Application of Fatigue Theories to Seismic Compression Estimation and the Evaluation of Liquefaction Potential

Samuel James Lasley

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Civil Engineering

Russell A. Green, Co-Chair
Adrian Rodriguez-Marek, Co-Chair
Martin C. Chapman
Matthew R. Eatherton

July 11, 2015
Blacksburg, Virginia

Keywords: earthquakes, dissipated energy, liquefaction, fatigue, seismic compression

Application of Fatigue Theories to Seismic Compression Estimation and the Evaluation of Liquefaction Potential

Samuel James Lasley

ABSTRACT

Earthquake-induced liquefaction of saturated soils and seismic compression of unsaturated soils are major sources of hazard to infrastructure, as attested by the wholesale condemnation of neighborhoods surrounding Christchurch, New Zealand. The hazard continues to grow as cities expand into liquefaction- and seismic compression-susceptible areas hence accurate evaluation of both hazards is essential.

The liquefaction evaluation procedure presented herein is based on dissipated energy and an SPT liquefaction/no-liquefaction case history database. It is as easy to implement as existing stress-based simplified procedures. Moreover, by using the dissipated energy of the entire loading time history to represent the demand, the proposed procedure melds the existing stress-based and strain-based liquefaction procedures in to a new, robust method that is capable of evaluating liquefaction susceptibility from both earthquake and non-earthquake sources of ground motion.

New relationships for stress reduction coefficient (r_d) and number of equivalent cycles (n_{eq}) are also presented herein. The r_d relationship has less bias and uncertainty than other common stress reduction coefficient relationships, and both the n_{eq} and r_d relationships are proposed for use in active tectonic and stable continental regimes. The n_{eq} relationship proposed herein is based on an alternative application of the Palmgren-Miner damage hypothesis, shifting from the existing high-cycle, low-damage fatigue framework to a low-cycle framework more applicable to liquefaction analyses.

Seismic compression is the accrual of volumetric strains caused by cyclic loading, and presented herein is a "non-simplified" model to estimate seismic compression. The proposed model is based on a modified version of the Richart-Newmark non-linear cumulative damage hypothesis, and was calibrated from the results of drained cyclic simple shear tests. The proposed

model can estimate seismic compression from any arbitrary strain time history. It is more accurate than other "non-simplified" seismic compression estimation models over a greater range of volumetric strains and can be used to compute number-of-equivalent shear strain cycles for use in "simplified" seismic compression models, in a manner consistent with seismic compression phenomenon.

This work received support from the National Science Foundation (NSF) grants CMMI-1030564 and CMMI-1435494 and US Army Engineer Research and Development Center (ERDC) grant W912HZ-13-C-0355. This support is gratefully acknowledged. Any opinions, findings, and conclusions or recommendations expressed in this research are those of the author and do not necessarily reflect the views of NSF or ERDC.

Acknowledgments

My academic and professional skills have been greatly advanced by the patient guidance and instruction of my advisors, Dr. Russell Green and Dr. Adrian Rodriguez-Marek. Their pursuit of excellence has been a powerful example for me, as has their kindness.

My years spent at Virginia Tech would have been far less productive and enjoyable if not for time spent in the company of top-notch students. Within the geotech program, the friendships and examples of Ashly Cabas, Haitham Dawood, Kevin Foster, Brett Maurer, Manisha Rai, Alexander Reeb, Craig Schillaber, Melis Sutman, and Daniel Vanden Berge have been especially meaningful to me. Outside the program, I have been uplifted and inspired by my interactions with Jacob Delimont, Derek Haderlie, Michael Irwin, Johnn Judd, Ron Kent, and DJ Seo.

The financial support provided by the Via Fellowship, the National Science Foundation (via grants CMMI-1030564 and CMMI-1435494), and ERDC (grant W912HZ-13-C-0355) is gratefully acknowledged. My extended involvement in this research would have been very difficult without it.

Finally, the constant and devoted support of my wife, Amber, has enabled me to succeed and given me greater desires to do so.

Contents

1 Introduction **1**

1.1 Soil Liquefaction 2

1.2 Seismic Compression 3

1.3 Objectives 4

1.4 Organization 4

1.5 Significance 6

1.6 References 7

1.7 Figures 9

2 Literature Review **11**

2.1 Introduction 11

2.2 Soil Testing 11

2.2.1 Laboratory Testing 12

Cyclic Triaxial 13

Cyclic Simple Shear 14

Hollow Cylinder Torsional Shear 15

Failure Criteria in Liquefaction Testing 15

2.2.2 In Situ Testing 16

Standard Penetration Test 16

Cone Penetrometer Test 17

2.3 Seismic Compression 18
2.3.1 Whitman and Ortigosa (1969) 18
2.3.2 Silver and Seed (1971) 18
2.3.3 Youd (1972) 19
2.3.4 Seed and Silver (1972) 19
2.3.5 Pyke et al. (1975) 20
2.3.6 Tokimatsu and Seed (1987) 20
2.3.7 Byrne (1991) 21
2.3.8 Chu and Vucetic (1992) 21
2.3.9 Pradel (1998) 21
2.3.10 Stewart et al. (2001) 22
2.3.11 Whang (2001) 22
2.3.12 Stewart et al. (2002) 22
2.3.13 Stewart and Whang (2003) 23
2.3.14 Hsu and Vucetic (2004) 24
2.3.15 Whang et al. (2004) 25
2.3.16 Stewart et al. (2004) 25
2.3.17 Sawada et al. (2006) 26
2.3.18 Duku et al. (2006) 26
2.3.19 Duku et al. (2008) 27
2.3.20 Yi (2010) 27
2.4 Equivalent-Linear Site Response Analysis 28
2.5 Stress-Based Simplified Liquefaction Evaluation Methods 29
2.6 Energy-Based Liquefaction Evaluation Methods 31
2.6.1 Field-Based Capacity Curves 32
Davis and Berrill (1982) 32
Berrill and Davis (1985) 33

Law et al. (1990) 33
Trifunac (1995) 34
Kayen and Mitchell (1997) 34
Green (2001) 35
Mayfield (2007) 36
Jafarian et al. (2014) 36
2.6.2 Laboratory-Based Capacity Curves 37
Case Western 37
Davis and Berrill (1996) 38
Jafarian et al. (2012) 38
Kokusho and Mimori (2015) 39
2.6.3 Summary of Energy-based Procedures 39
2.7 References 40
2.8 Figures 48

3 Manuscript #1: An Approach for Estimating Seismic Compression Using Site Response Analyses 54

3.1 Abstract 56
3.2 Introduction 57
3.3 Fatigue Models 60
3.3.1 Palmgren-Miner (P-M) Model 61
3.3.2 Richart-Newmark (R-N) Model 62
3.3.3 Byrne (1991) Model 63
3.4 Laboratory Testing Procedure 64
3.5 Modification and Calibration of the R-N Fatigue Model 65
3.5.1 Model Modification 65
3.5.2 Model Calibration 66

3.6 Accuracy of the Fatigue Models 67
3.6.1 Variable-Amplitude Sinusoidal Loading 67
3.6.2 Earthquake Loading 68
3.6.3 Overall Prediction Results 69
3.7 Case History 70
3.8 Summary and Conclusions 71
3.9 Acknowledgements 72
3.10 References 72
3.11 Tables 77
3.12 Figures 81

4 Manuscript #2: Number of Equivalent Cycles for Liquefaction Evaluations in Active Tectonic and Stable Continental Regimes 94

4.1 Abstract 96
4.2 Introduction 97
4.3 The Palmgren-Miner (P-M) Fatigue Theory 98
4.4 Previous n_{eq} Studies for Liquefaction Evaluations 100
4.4.1 Seed et al. (1975) 100
4.4.2 Liu et al. (2001) 102
4.4.3 Biondi et al. (2004) 103
4.4.4 Green and Terri (2005) 103
4.4.5 Lee (2009) 105
4.5 Approach Used to Develop Proposed Correlations 106
4.5.1 Earthquake Databases 106
4.5.2 Soil Profiles 107
4.5.3 Multidirectional Shaking 108
4.5.4 Regression Analysis 109

4.5.5 Proposed n_{eq} Correlations . . . 110
4.6 Comparison of n_{eq} Correlations . . . 112
4.6.1 Green and Terri (2005) vs. Seed et al. (1975) Approaches 112
4.6.2 Correlation with Other Ground Motion Duration Metrics 113
4.6.3 Comparison of n_{eq} Predictive Equations . . . 114
4.7 Summary and Conclusions . . . 115
4.8 Acknowledgments . . . 116
4.9 References . . . 116
4.10 Tables . . . 121
4.11 Figures . . . 125
4.12 Electronic Supplement . . . 136
4.12.1 NGA Database (WUS) . . . 136
4.12.2 McGuire et al. (2001) Database (CEUS) . . . 147

5 Manuscript #3: A New Stress Reduction Coefficient Relationship for Liquefaction Triggering Analyses 156
5.1 Abstract . . . 158
5.2 Introduction . . . 159
5.3 Earthquake Motions . . . 160
5.4 Soil Profiles . . . 161
5.5 Proposed r_d Relationship . . . 161
5.6 Discussion . . . 163
5.7 Conclusion . . . 165
5.8 References . . . 165
5.9 Tables . . . 168
5.10 Figures . . . 170
5.11 Electronic Supplement . . . 177
5.11.1 NGA Database (WUS) . . . 177
5.11.2 McGuire et al. (2001) Database (CEUS) . . . 188

6 Manuscript #4: An SPT Liquefaction Evaluation Procedure Based on Dissipated Energy 197

6.1 Abstract . . . 199
6.2 Introduction . . . 200
6.3 Existing Stress-based Methods . . . 201
6.4 Dissipated Energy and Liquefaction . . . 202
6.5 Laboratory Approach . . . 203
6.6 Case History-Based Approach . . . 204
6.6.1 Methodology . . . 205
6.6.2 Definition of the Limit State Curve . . . 208
6.6.3 Regression Approach . . . 210
6.6.4 Performance of the Proposed Procedure . . . 210
6.6.5 Performance Compared to Boulanger and Idriss (2012) 213
6.6.6 Behavior at High Effective Overburden Pressures . . . 215
6.6.7 Application with Site Response Analyses . . . 217
6.7 Comparison of Lab- and Field-Based Capacity Curves . . . 218
6.8 Conclusion . . . 219
6.9 Acknowledgements . . . 220
6.10 References . . . 220
6.11 Tables . . . 226
6.12 Figures . . . 230
6.13 Electronic Supplement . . . 249
6.13.1 Sample Calculations . . . 264
Calculate Small-Strain Shear Modulus . . . 265
Calculate Stress-Reduction Coefficient . . . 266
Calculate the Dissipated Energy in One Equivalent Cycle 267
Calculate Number of Equivalent Cycles . . . 269

Calculate Total Dissipated Energy 269
Calculate the Limit-State Value, g 269
Calculate Probability of Liquefaction 270
Calculate the Dissipated Energy to Liquefaction 270
Calculate the Factor of Safety 271
6.13.2 Procedure to Evaluate Liquefaction Potential 272
Simplified Procedure 272
Non-Simplified Procedure 276
References 278
Figures 280

7 Conclusions **283**
7.1 Summary of Research 283
7.1.1 Seismic Compression Method 283
7.1.2 Number of Equivalent Stress Cycles Correlation 284
7.1.3 Stress Reduction Coefficient Correlation 285
7.1.4 Dissipated-Energy-Based Liquefaction Evaluation . . . 286
7.2 Recommendations for Future Work 286
7.3 References 287

A Contents of Appendices **289**

B Seismic Compression Lab Test Results **290**
B.1 Constant-Amplitude Results 290
B.2 Irregular-Amplitude Results 313

C Cycle Counting Methods for Seismic Compression **354**
C.1 References 355
C.2 Tables 356
C.3 Figures 357

D Cyclic Simple Shear Lab Test Results **360**

D.1 Introduction 360

D.2 Setup and Equipment 360

D.3 Specimen Preparation 361

D.4 Testing Procedure 361

D.5 Materials 362

D.6 Results 362

D.6.1 Stress-Controlled Sinusoidal Tests 362

D.6.2 Earthquake Loading Tests 363

D.6.3 Strain-Controlled Tests 364

D.6.4 Comparison Between Stress- and Strain-Controlled Tests 365

D.6.5 Comparison With Other Test Programs 366

D.7 Summary 367

D.8 References 367

D.9 Tables 369

D.10 Figures 370

D.11 Sinusoidal Loading Results 390

D.12 Earthquake Loading Results 392

D.13 Strain-Controlled Sinusoidal Loading Results 394

E Soil Profiles of the Equivalent-Linear Analyses **395**

E.1 Profile000 - Balboa Boulevard-Northridge Earthquake (1994) . . . 395

E.2 Profile001 - Malden Street-Northridge Earthquake (1994) 397

E.3 Profile002 - Wynne Avenue-Northridge Earthquake (1994) 399

E.4 Profile003 - MBARI NO:3 EB-1-Loma Prieta Earthquake (1989) . 401

E.5 Profile004 - MBARI NO:3 EB-5-Loma Prieta Earthquake (1989) . 403

E.6 Profile005 - MBARI NO:3 UC-B-10-Loma Prieta Earthquake (1989) 405

E.7 Profile006 - Treasure Island-Loma Prieta Earthquake (1989) . . . 407
E.8 Profile007 - SFOBB-1-Loma Prieta Earthquake (1989) 409
E.9 Profile008 - SFOBB-2-Loma Prieta Earthquake (1989) 411
E.10 Profile009 - POO7-2-Loma Prieta Earthquake (1989) 413
E.11 Profile010 - POO7-3-Loma Prieta Earthquake (1989) 415
E.12 Profile011 - Woodward Marine UC-B4-Loma Prieta Earthquake (1989) 417
E.13 Profile012 - Moss Landing UC-B1-Loma Prieta Earthquake (1989) 419
E.14 Profile013 - Moss Landing UC-B2-Loma Prieta Earthquake (1989) 421
E.15 Profile014 - Marine Lab. B1-Loma Prieta Earthquake (1989). . . 423
E.16 Profile015 - Marine Lab. B2-Loma Prieta Earthquake (1989). . . 425
E.17 Profile016 - Miller Farm CMF 3-Loma Prieta Earthquake (1989) . 427
E.18 Profile017 - Miller Farm CMF 5-Loma Prieta Earthquake (1989) . 429
E.19 Profile018 - Miller Farm CMF 8-Loma Prieta Earthquake (1989) . 431
E.20 Profile019 - Miller Farm CMF 10-Loma Prieta Earthquake (1989). 433
E.21 Profile020 - Richmond City Hall-Loma Prieta Earthquake (1989) . 435
E.22 Profile021 - Port of Richmond POR-2-Loma Prieta Earthquake (1989) 437
E.23 Profile022 - Port of Richmond POR-3-Loma Prieta Earthquake (1989) 439
E.24 Profile023 - Port of Richmond POR-4-Loma Prieta Earthquake (1989) 441
E.25 Profile024 - Miller Farm-Loma Prieta Earthquake (1989) 443
E.26 Profile025 - Farris Farm-Loma Prieta Earthquake (1989) 445
E.27 Profile026 - Wildlife Site-Elmore Ranch (1987) 447
E.28 Profile027 - Wildlife Site-Superstition Hills (1987) 448
E.29 Profile028 - Radio Tower B1-Superstition Hills (1987) 449
E.30 Profile029 - Radio Tower B2-Superstition Hills (1987) 451

E.31 Profile030 - McKim Ranch A-Superstition Hills (1987) 453
E.32 Profile031 - Kornbloom-Superstition Hills (1987) 455
E.33 Profile032 - River Park A & C-Superstition Hills (1987) 457
E.34 Profile033 - Heber Road A1-Superstition Hills (1987) 459
E.35 Profile034 - Heber Road A2-Superstition Hills (1987) 461
E.36 Profile035 - Heber Road A3-Superstition Hills (1987) 463
E.37 Profile036 - Kornbloom-Westmorland (1981) 465
E.38 Profile037 - McKim Ranch-Westmorland (1981) 467
E.39 Profile038 - Radio Tower B1-Westmorland (1981) 469
E.40 Profile039 - Radio Tower B2-Westmorland (1981) 471
E.41 Profile040 - River Park A & C-Westmorland (1981) 473
E.42 Profile041 - Wildlife-Westmorland (1981) 475
E.43 Profile042 - Heber Road A1-Imperial Valley (1979) 477
E.44 Profile043 - Heber Road A2-Imperial Valley (1979) 479
E.45 Profile044 - Heber Road A3-Imperial Valley (1979) 481
E.46 Profile045 - Kornbloom-Imperial Valley (1979) 483
E.47 Profile046 - Radio Tower B1-Imperial Valley (1979) 485
E.48 Profile047 - Radio Tower B2-Imperial Valley (1979) 487
E.49 Profile048 - River Park A & C-Imperial Valley (1979) 489
E.50 Profile049 - Wildlife Site-Imperial Valley (1979) 491
E.51 Profile Summary Table 493

F Earthquake Motions **495**
F.1 NGA Earthquake Database 495
F.2 McGuire et al. (2001) Earthquake Motions (CEUS) 600

G Cyclic Simple Shear Testing Manual **680**

G.1 Introduction 680

G.1.1 Physical Setup 680

G.1.2 The Cell 682

G.1.3 The Pressure Panel 682

G.1.4 Other Parts 685

G.2 The Testing Cycle 685

G.2.1 Sample Preparation 685

Placement of Bottom Platen, Membrane, and Mold . . 685

Dry Pluviation 691

G.2.2 Assembly of Equipment 696

G.2.3 GCTS Control of Testing 700

G.2.4 Cleanup/Tear Down 701

G.3 Data Reduction 704

G.3.1 Void Ratio and Relative Density 704

G.3.2 Number of Cycles to Liquefaction 705

G.3.3 Calculation of Dissipated Energy 705

G.3.4 Plotting Results 705

G.4 Maintenance/Preparation 707

G.4.1 Creating a Cyclic Simple Shear Test File 707

Arbitrary (Earthquake) Loading 720

G.4.2 Friction Baseline Tests 721

G.4.3 Sensor Calibration 722

Load Cells 723

Pressure Sensors 727

G.4.4 Pore Pressure Servo Calibration 727

G.4.5 Loading & Saving Configuration Files 728

H Cyclic Triaxial Testing Manual **730**

H.1 Introduction 730

H.2 Setup and Equipment 730

H.2.1 Physical Setup 730

H.2.2 The Cell 732

H.2.3 The Pressure Panel 732

H.2.4 Other Parts 735

H.2.5 The GCTS Software 735

H.3 The Test Cycle 744

H.3.1 Setup 744

H.3.2 Sample Preparation 745

Placement of Bottom Platen, Membrane, and Mold . . 745

Wet Pluviation 746

H.3.3 Assembly of Equipment 749

Normal Actuator Connection 749

H.3.4 Saturation 750

Flushing the Specimen 751

Manual Back-Pressure Saturation 752

Switching From Manual to Servo Pressure Control . . . 754

Automatic Back-Pressure Saturation 756

H.3.5 GCTS Control of Testing 756

H.3.6 Cleanup/Tear Down 757

H.4 Data Reduction 759

H.4.1 Void Ratio and Relative Density 759

H.4.2 Number of Cycles to Liquefaction 760

H.4.3 Calculation of Dissipated Energy 760

H.4.4 Plotting Results 761

H.5 Maintenance/Preparation 762
H.5.1 Creating a Triaxial Test File 762
Arbitrary (Earthquake) Loading 771
H.5.2 Friction Baseline Tests 772
H.5.3 Sensor Calibration 773
Load Cells 774
Pressure Sensors 778
H.5.4 Pore Pressure Servo Calibration 778
H.5.5 Loading & Saving Configuration Files 778

I Conference Paper: Verification of ShakeVT2 781

J Code 793

J.1 License 793
J.2 Equivalent-Linear: ShakeVT2 794
J.3 Shear Modulus Reduction and Damping Curves 853
J.4 Batch Site Response Analyses 856
J.5 Regression of n_{eq} 884
J.6 Regression of r_d 889
J.7 Fit of Limit-State Curves 898
J.8 Data Reduction of CSS Lab Data 945
J.9 Data Reduction of Seismic Compression Lab Data 998
J.10 Implementation of ASTM 'Cycle Counting' Methods 1037

List of Figures

1.1 Liquefaction during the 2010 M_w7.0 earthquake caused damage at the Port-au-Prince seaport, hampering relief efforts. (gCaptain 2010) [Fair Use] 9

1.2 Liquefaction during the Christchurch earthquake sequence led to the condemnation of entire neighborhoods (NZDF 2011). [Fair Use] 10

2.1 Triaxial specimen of Monterey 0/30 sand with vacuum applied. 48

2.2 A cyclic simple shear specimen in preparation. 49

2.3 Volumetric strain as a function of shear strain and a) relative density (D_r), or b) SPT blow count. (Tokimatsu and Seed, 1987) [Fair use.] 50

2.4 Shear modulus and damping degradation as a function of strain and overburden stress (Darendeli and Stokoe 2001). [Fair use.] . 51

2.5 Cyclic stress ratio versus corrected SPT blow counts for liquefaction and non-liquefaction case histories for M7.5 earthquakes. (Seed et al. 1985) [Fair use.] 52

2.6 Stress-strain hysteresis loops of a stress-controlled cyclic simple shear test. 53

3.1 Relationship between cumulative damage (D) and the number of applied load cycles (n) having amplitude S. Failure occurs when D=1, with the corresponding number of applied cycles designated as N. (Adapted from Stallmayer and Walker, 1968) [Fair use.] . 81

3.2 Damage-cycle relationship for various load amplitudes. (Adapted from Stallmeyer and Walker, 1968) [Fair use.] 82

3.3 Damage-cycle ratio relationships for a) load-independent material, and b) load-dependent material. (Adapted from Stallmeyer and Walker, 1969) [Fair use.] 83

3.4 The left plot shows the damage accumulation path according to the Palmgren-Miner hypothesis. Two different cyclic loading time histories will follow the same damage accumulation path, as shown in the center and right plots. This is because the P-M hypothesis is load-independent. 83

3.5 At left, various load accumulation paths predicted by the Richart-Newmark hypothesis for a range of r values. Right and center, the R-N hypothesis is load-dependent, meaning that the ordering of load cycles affects the damage accumulation path and the amount of predicted damage. 84

3.6 The GCTS testing apparatus used for this study with triaxial specimen. (Modifications not shown.) 84

3.7 The particle size gradation of the quartz sand used in this study. 85

3.8 On the left, confining rings with pluviated sand. Right, specimen with top platen in place and under vacuum. (Note: the pictured sand was not used in this study.) 85

3.9 Examples of the constant-amplitude sinusoidal, variable-amplitude sinusoidal, and earthquake-type loadings used for the cyclic simple shear testing. 86

3.10 Typical results from a constant-amplitude strain-controlled cyclic simple shear test on dry sand. 87

3.11 Shear strain time history of a variable-amplitude sinusoidal test. 88

3.12 Predicted and actual damage paths for a variable-amplitude sinusoidal loading function. 89

3.13 Boxplots of the percent error in prediction of volumetric strains for irregular-sinusoid loadings. Negative values signify overprediction. 90

3.14 At top, a *shock-type*, and, at bottom, a *vibration-type* motion used as the loading function. 91

3.15 Actual and predicted volumetric strains for a *vibration-type* motion. 92

3.16 Boxplots of the percent error in prediction of volumetric strains for earthquake motion loadings. 93

4.1 The sequence of steps used by Seed et al. (1975) to normalize a CSR-$N_{liq.}$ curve. Step 1: Designate the CSR required to induce liquefaction in one cycle as CSR_1; Step 2: Normalize the ordinate values by dividing CSR by CSR_1; Step 3: Apply a FS to CSR_1 (i.e., divide CSR_1 by FS) and determine the number of cycles required to induce liquefaction (N_{ref}) corresponding to the normalized peak amplitude (α) of the uniform cyclic load; and Step 4: Normalize the abscissa values as N_{ref}/N_{liq}. 125

4.2 Normalized CSR-N_{liq} curve ($FS = 1.5$) used by Seed et al. (1975) to compute n_{eq}. Such curves are commonly referred to as weighting factor (WF) curves. The CSR-$N_{liq.}$ curve used to develop the WF curve came from De Alba et al. (1975): large-scale, undrained, cyclic simple shear tests performed on reconstituted Monterey No. 0 sand specimens having a relative density of approximately 65%. (adapted from Seed et al. 1975) 126

4.3 Magnitude versus distance distribution of the motions used in this study. 127

4.4 WUS profiles used in this study. The number in the corner of each subplot is the number of profiles in that grouping. 128

4.5 CEUS profiles used in this study. The number in the corner of each subplot is the number of profiles in that grouping. 129

4.6 Scatterplots of a_{max}, M_w, and R versus $n_{eq,A1}$ from the WUS database. The number of equivalent cycles has a much higher correlation to the a_{max} at the ground surface than to M_w. . . . 130

4.7 Scatterplots of a_{max}, M_w, and R versus $n_{eq,A1}$ from the CEUS database. The number of equivalent cycles has a much higher correlation to the a_{max} at the ground surface than to M_w. . . . 130

4.8 Values of n_{eq} predicted by the correlations proposed herein. The top plots show the relationship using Eqn. 4.15 and the bottom figures are for Eqn. 4.17. The n_{eq} correlations corresponding to the WUS are to the left; CEUS correlations are plotted on the right. 131

4.9 Comparison of n_{eq} values computed using Seed et al. (1975) and Liu et al. (2001) (i.e., $n_{eq,\text{Sea75}}$ and $n_{eq,\text{Lea01}}$, respectively) with values computed using Approaches 1, 2, and 3 for 0 to 4 m depths (i.e., $n_{eq,\text{A1}}$, $n_{eq,\text{A1}}$, and $n_{eq,\text{A1}}$, respectively) for WUS. 132

4.10 Comparison of n_{eq} values computed using Seed et al. (1975) and Liu et al. (2001) (i.e., $n_{eq,\text{Sea75}}$ and $n_{eq,\text{Lea01}}$, respectively) with values computed using Approaches 1, 2, and 3 for 0 to 4 m depths (i.e., $n_{eq,\text{A1}}$, $n_{eq,\text{A1}}$, and $n_{eq,\text{A1}}$, respectively) for CEUS. 132

4.11 Plots of $n_{eq,\text{A1}}$ for 0 to 4 m depths versus other ground motion duration metrics: bracketed duration ($\pm 0.05g$ threshold), 5-95% significant duration, and 5-75% significant duration for WUS. . 133

4.12 Plots of $n_{eq,\text{A1}}$ for 0 to 4 m depths versus other ground motion duration metrics: bracketed duration ($\pm 0.05g$ threshold), 5-95% significant duration, and 5-75% significant duration for CEUS. . 133

4.13 Comparison of n_{eq} correlations for $M = 5.5$. "WUS" and "CEUS" refer to the WUS and CEUS correlations developed herein from approach 1 ($n_{eq,\text{A1}}$ values). 134

4.14 Comparison of n_{eq} correlations for $M = 7.5$. "WUS" and "CEUS" refer to the WUS and CEUS correlations developed herein from approach 1 ($n_{eq,\text{A1}}$ values). 134

4.15 Comparison of functional form #1 (approach 1, $n_{eq,\text{A1}}$) and the Biondi et al. (2004) n_{eq} correlations. 135

5.1 Stress reduction coefficient (r_d) is defined as the ratio of the maximum seismically induced shear stress at the base of a flexible soil column to that at the base of a rigid soil column, both having length z. 170

5.2 Magnitude versus distance distribution of the motions used in this study. 171

5.3 WUS profiles used in this study. The number in the corner of each subplot is the number of profiles in that grouping. 172

5.4 CEUS profiles used in this study. The number in the corner of each subplot is the number of profiles in that grouping. 173

5.5 The proposed models at magnitudes of 5.5 and 7.5, compared with Idriss (1999), Liao and Whitman (1986), and Cetin (2000). 174

5.6 Predicted r_d values from several predictive equations above a heat map showing the density of $r_{d,\text{ Actual}}$ values from the WUS dataset. The value E is the Nash-Sutcliffe (1970) model efficiency coefficient. 175

5.7 Predicted r_d values above a heat map showing the density of $r_{d,\text{ Actual}}$ values from the CEUS dataset. The value E is the Nash-Sutcliffe (1970) model efficiency coefficient. 176

6.1 Results of a constant-volume cyclic simple shear test: Left: Shear stress versus shear strain hysteresis loops, Right: Shear stress versus vertical stress. 230

6.2 Results of a constant-volume cyclic simple shear test: Dissipated energy increasing to liquefaction. A best fit line over the linear portion of the dissipated energy curve ($\Delta W_{\text{Effect.}}$) is used to obtain the dissipated energy corresponding to total stress analyses (ΔW_{Total}). The difference between the two measures of dissipated energy is illustrated here and can be attributed to the softening of the soil with increasing pore water pressure. 231

6.3 Results of cyclic simple shear tests with contours of equal dissipated energy to liquefaction. 232

6.4 Results of cyclic simple shear tests plotted with normalized dissipated energy to liquefaction (corrected to remove softening effects) versus relative density. 233

6.5 Direct Method to estimate $\Delta W_{1eq}/G_{max}$ (after Tokimatsu and Seed 1987) using the Darendeli and Stokoe (2001) degradation curves. Lines correspond to a clean sand ($FC = 0\%$, $PI = 0$). To use the figure, calculate the value on the x-axis, read up to the appropriate curve, and obtain the corresponding value from the y-axis. 234

6.6 Direct Method to estimate $\Delta W_{1eq}/G_{max}$ (after Tokimatsu and Seed 1987) using the Darendeli and Stokoe (2001) degradation curves. Lines correspond to "sands with high fines content." To use the figure, calculate the value on the x-axis, read up to the appropriate curve, and obtain the corresponding value from the y-axis. 235

6.7 Direct Method to estimate $\Delta W_{1eq}/G_{max}$ (after Tokimatsu and Seed 1987) using the Darendeli and Stokoe (2001) degradation curves. Lines correspond to a low-plasticity silt. To use the figure, calculate the value on the x-axis, read up to the appropriate curve, and obtain the corresponding value from the y-axis. 236

6.8 Proposed limit state curve (scenario #1) with case history data. 237

6.9 Proposed limit state curve (scenario #1) with case history data and normalized dissipated energy on a log axis. 238

6.10 Proposed limit state curve (scenario #1, $P = 50\%$) with case history data for various ranges of fines content. 239

6.11 Proposed limit state curve (scenario #1, $P = 50\%$) with case history data for various ranges of initial effective overburden stress. 240

6.12 Proposed limit state curve (scenario #1, $P = 50\%$) with case history data for various ranges of peak ground acceleration. . . 241

6.13 Proposed limit state curve (scenario #1, $P = 50\%$) with case history data for various ranges of moment magnitude. 242

6.14 Comparison of proposed procedure and the Boulanger and Idriss (2012) procedure. Solid lines indicate the 15, 50, and 85% probabilities of liquefaction. Large dots signify case history data points for which the models do not agree; the color of the point indicates whether the case history site liquefied. White squares indicate points on which the two procedures agree correctly, and the stars indicate points where the two procedures agree incorrectly. . . 243

6.15 Comparison of proposed procedure and Boulanger and Idriss (2012). Points are separated on the y-axis by whether the case history liquefied or not. The shape of the points indicate which of the procedures predicted the case history correctly. The numbers in the left-most pane indicate the number of case histories in each row. 244

6.16 Overburden correction factor developed for stress-based methods assuming $\Delta W_{liq}/\sigma'_{v0}$ is insensitive to extreme overburden stresses. 245

6.17 Demand, capacity, and factor of safety as a function of initial effective overburden stress for a M_w5.5, $a_{max} = 0.15g$ event. The ground water table is located at a shallow depth (0.5 m) for all initial effective overburden stresses. 246

6.18 Demand, capacity, and factor of safety as a function of initial effective overburden stress for a M_w7.5, $a_{max} = 0.35g$ event. The ground water table is located at a shallow depth (0.5 m) for all initial effective overburden stresses. 247

6.19 Proposed limit state curve (scenario #1, P = 15, 50, & 85%) compared with the laboratory data. 248

6.20 Graphical method to obtain $\Delta W_{1,eq}$ using Darendeli and Stokoe's shear modulus and damping degradation curves. 268

6.21 Direct Method to estimate $\Delta W_{1eq}/G_{max}$ (after Tokimatsu and Seed 1987) using the Darendeli and Stokoe (2001) degradation curves. Lines correspond to a clean sand ($FC = 0\%$, $PI = 0$). To use the figure, calculate the value on the x-axis, read up to the appropriate curve, and obtain the corresponding value from the y-axis. 280

6.22 Direct Method to estimate $\Delta W_{1eq}/G_{max}$ (after Tokimatsu and Seed 1987) using the Darendeli and Stokoe (2001) degradation curves. Lines correspond to "sands with high fines content." To use the figure, calculate the value on the x-axis, read up to the appropriate curve, and obtain the corresponding value from the y-axis. 281

6.23 Direct Method to estimate $\Delta W_{1eq}/G_{max}$ (after Tokimatsu and Seed 1987) using the Darendeli and Stokoe (2001) degradation curves. Lines correspond to a low-plasticity silt. To use the figure, calculate the value on the x-axis, read up to the appropriate curve, and obtain the corresponding value from the y-axis. 282

B.1 Test results for irregular-loading seismic compression test *139.csv*. 314

B.2 Test results for irregular-loading seismic compression test *140.csv*. 315

B.3 Test results for irregular-loading seismic compression test *141.csv*. 316

B.4 Test results for irregular-loading seismic compression test *143.csv*. 317

B.5 Test results for irregular-loading seismic compression test *144.csv*. 318

B.6 Test results for irregular-loading seismic compression test *145.csv*. 319

B.7 Test results for irregular-loading seismic compression test *146.csv*. 320

B.8 Test results for irregular-loading seismic compression test *162.csv*. 321

B.9 Test results for irregular-loading seismic compression test *163.csv*. 322

B.10 Test results for irregular-loading seismic compression test *164.csv*. 323

B.11 Test results for irregular-loading seismic compression test *178.csv*. 324

B.12 Test results for irregular-loading seismic compression test *179.csv*. 325

B.13 Test results for irregular-loading seismic compression test *180.csv*. 326

B.14 Test results for irregular-loading seismic compression test *181.csv*. 327

B.15 Test results for irregular-loading seismic compression test *142.csv*. 328

B.16 Test results for irregular-loading seismic compression test *147.csv*. 329

B.17 Test results for irregular-loading seismic compression test *177.csv*. 330

B.18 Test results for irregular-loading seismic compression test *100.csv*. 331

B.19 Test results for irregular-loading seismic compression test *101.csv*. 332

B.20 Test results for irregular-loading seismic compression test *102.csv*. 333

B.21 Test results for irregular-loading seismic compression test *105.csv*. 334

B.22 Test results for irregular-loading seismic compression test *106.csv*. 335

B.23 Test results for irregular-loading seismic compression test *107.csv*. 336

B.24 Test results for irregular-loading seismic compression test *108.csv*. 337
B.25 Test results for irregular-loading seismic compression test *110.csv*. 338
B.26 Test results for irregular-loading seismic compression test *111.csv*. 339
B.27 Test results for irregular-loading seismic compression test *112.csv*. 340
B.28 Test results for irregular-loading seismic compression test *113.csv*. 341
B.29 Test results for irregular-loading seismic compression test *115.csv*. 342
B.30 Test results for irregular-loading seismic compression test *116.csv*. 343
B.31 Test results for irregular-loading seismic compression test *117.csv*. 344
B.32 Test results for irregular-loading seismic compression test *118.csv*. 345
B.33 Test results for irregular-loading seismic compression test *120.csv*. 346
B.34 Test results for irregular-loading seismic compression test *97.csv*. 347
B.35 Test results for irregular-loading seismic compression test *98.csv*. 348
B.36 Test results for irregular-loading seismic compression test *99.csv*. 349
B.37 Test results for irregular-loading seismic compression test *104.csv*. 350
B.38 Test results for irregular-loading seismic compression test *109.csv*. 351
B.39 Test results for irregular-loading seismic compression test *114.csv*. 352
B.40 Test results for irregular-loading seismic compression test *119.csv*. 353

C.1 An illustration of the *mean crossing peak counting* algorithm. . 357
C.2 Two one-half cycles as determined by the *range counting* algorithm. 358
C.3 The *rainflow counting* algorithm. 359

D.1 GCTS equipment with CSS specimen and modifications. . . . 370
D.2 Part of the top platen mount that has been modified. The silver-colored block is new; it replaces a larger (taller) block to allow room for the internal normal pressure cell (shown). 371
D.3 Flask, funnel, and stopper used for dry pluviation. 372
D.4 Removing excess sand from a liquefaction-evaluation specimen. 373

D.5 Gradations of the sands used in cyclic simple shear testing. . . 374

D.6 Typical results for a stress-controlled sinusoidal test on a loose specimen. 375

D.7 Typical results for a stress-controlled sinusoidal test on a dense specimen. 376

D.8 Cyclic stress ratio, CSR, versus number of cycles to liquefaction, $N_{liq.}$ for stress-controlled sinusoidal tests. 377

D.9 Normalized dissipated energy, $\Delta W_{\text{Effect.}}/\sigma'_{v0}$, versus relative density, D_r, for stress-controlled sinusoidal tests. 378

D.10 Typical results for a stress-controlled earthquake-loading test. . 379

D.11 Normalized dissipated energy, $\Delta W_{\text{Effect.}}/\sigma'_{v0}$, versus relative density, D_r, for stress-controlled earthquake-loading tests. 380

D.12 Typical results for a strain-controlled sinusoidal test on a loose specimen. 381

D.13 Typical results for a strain-controlled sinusoidal test on a dense specimen. 382

D.14 Amplitude of cyclic shear strain, γ, versus number of cycles to liquefaction, $N_{liq.}$ for strain-controlled sinusoidal tests. 383

D.15 Normalized dissipated energy, $\Delta W_{\text{Effect.}}/\sigma'_{v0}$, versus relative density, D_r, for strain-controlled sinusoidal tests. 384

D.16 Normalized dissipated energy, $\Delta W_{\text{Effect.}}/\sigma'_{v0}$, versus relative density, D_r, for all tests from this study. 385

D.17 Normalized dissipated energy, $\Delta W_{\text{Effect.}}/\sigma'_{v0}$, versus relative density, D_r, for cyclic triaxial tests performed on Monterey sand (from Green 2001). 386

D.18 Normalized dissipated energy, $\Delta W_{\text{Effect.}}/\sigma'_{v0}$, versus relative density, D_r, for all tests performed as part of the liquefaction triggering program. 387

D.19 Normalized dissipated energy, $\Delta W_{\text{Effect.}}/\sigma'_{v0}$, versus relative density, D_r, for cyclic triaxial tests performed on Monterey #0 sand (Riemer et al. 1994). 388

D.20 Normalized dissipated energy, $\Delta W_{\text{Effect.}}/\sigma'_{v0}$, versus relative density, D_r, for all laboratory testing (this study; Green 2001; Riemer et al. 1994). 389

E.1 Shear wave velocities of Profile000 396
E.2 Shear wave velocities of Profile001 398
E.3 Shear wave velocities of Profile002 400
E.4 Shear wave velocities of Profile003 402
E.5 Shear wave velocities of Profile004 404
E.6 Shear wave velocities of Profile005 406
E.7 Shear wave velocities of Profile006 408
E.8 Shear wave velocities of Profile007 410
E.9 Shear wave velocities of Profile008 412
E.10 Shear wave velocities of Profile009 414
E.11 Shear wave velocities of Profile010 416
E.12 Shear wave velocities of Profile011 418
E.13 Shear wave velocities of Profile012 420
E.14 Shear wave velocities of Profile013 422
E.15 Shear wave velocities of Profile014 424
E.16 Shear wave velocities of Profile015 426
E.17 Shear wave velocities of Profile016 428
E.18 Shear wave velocities of Profile017 430
E.19 Shear wave velocities of Profile018 432
E.20 Shear wave velocities of Profile019 434
E.21 Shear wave velocities of Profile020 436
E.22 Shear wave velocities of Profile021 438
E.23 Shear wave velocities of Profile022 440
E.24 Shear wave velocities of Profile023 442

E.25 Shear wave velocities of Profile024 444
E.26 Shear wave velocities of Profile025 446
E.27 Shear wave velocities of Profile026 447
E.28 Shear wave velocities of Profile027 448
E.29 Shear wave velocities of Profile028 450
E.30 Shear wave velocities of Profile029 452
E.31 Shear wave velocities of Profile030 454
E.32 Shear wave velocities of Profile031 456
E.33 Shear wave velocities of Profile032 458
E.34 Shear wave velocities of Profile033 460
E.35 Shear wave velocities of Profile034 462
E.36 Shear wave velocities of Profile035 464
E.37 Shear wave velocities of Profile036 466
E.38 Shear wave velocities of Profile037 468
E.39 Shear wave velocities of Profile038 470
E.40 Shear wave velocities of Profile039 472
E.41 Shear wave velocities of Profile040 474
E.42 Shear wave velocities of Profile041 476
E.43 Shear wave velocities of Profile042 478
E.44 Shear wave velocities of Profile043 480
E.45 Shear wave velocities of Profile044 482
E.46 Shear wave velocities of Profile045 484
E.47 Shear wave velocities of Profile046 486
E.48 Shear wave velocities of Profile047 488
E.49 Shear wave velocities of Profile048 490
E.50 Shear wave velocities of Profile049 492

F.1 WUS 1: *1011_WON095* and *1011_WON185*. 496

F.2 WUS 2: *1012_LA0000* and *1012_LA0090*. 496
F.3 WUS 3: *1021_L04000* and *1021_L04090*.. 497
F.4 WUS 4: *1023_L09090* and *1023_L09000*.. 497
F.5 WUS 5: *1027_LV1000* and *1027_LV1090*. 498
F.6 WUS 6: *1029_LV3000* and *1029_LV3090*. 498
F.7 WUS 7: *1033_LIT090* and *1033_LIT180*. 499
F.8 WUS 8: *1041_MTW000* and *1041_MTW090*.. 499
F.9 WUS 9: *1050_PAC265* and *1050_PAC175*. 500
F.10 WUS 10: *1051_PUL104* and *1051_PUL194*. 500
F.11 WUS 11: *1060_CUC090* and *1060_CUC180*. 501
F.12 WUS 12: *1074_SAN090* and *1074_SAN180*. 501
F.13 WUS 13: *1078_5108-090* and *1078_5108-360*.. 502
F.14 WUS 14: *1091_VAS000* and *1091_VAS090*.. 502
F.15 WUS 15: *1096_WWJ090* and *1096_WWJ180*. 503
F.16 WUS 16: *1142_IZ1090* and *1142_IZ1000*. 504
F.17 WUS 17: *1154_BRS090* and *1154_BRS180*. 505
F.18 WUS 18: *1159_ERG090* and *1159_ERG180*. 505
F.19 WUS 19: *1165_IZT090* and *1165_IZT180*. 506
F.20 WUS 20: *1168_MNS000* and *1168_MNS090*. 506
F.21 WUS 21: *1169_MSK000* and *1169_MSK090*. 507
F.22 WUS 22: *1172_TKR090* and *1172_TKR180*. 507
F.23 WUS 23: *124_A-FLT000* and *124_A-FLT270*. 508
F.24 WUS 24: *1257_HWA003-W* and *1257_HWA003-N*. 508
F.25 WUS 25: *126_GAZ000* and *126_GAZ090*. 509
F.26 WUS 26: *133_B-SRO000* and *133_B-SRO270*. 509
F.27 WUS 27: *1347_ILA063-N* and *1347_ILA063-W*. 510
F.28 WUS 28: *1352_KAU003-N* and *1352_KAU003-W*.. 510

F.29 WUS 29: *139_DAY-LN* and *139_DAY-TR*. 511
F.30 WUS 30: *143_TAB-LN* and *143_TAB-TR*. 511
F.31 WUS 31: *1440_TAP065-E* and *1440_TAP065-N*. 512
F.32 WUS 32: *1446_TAP077-N* and *1446_TAP077-W*. 513
F.33 WUS 33: *146_G01230* and *146_G01320*. 513
F.34 WUS 34: *150_G06230* and *150_G06320*. 514
F.35 WUS 35: *1518_TCU085-E* and *1518_TCU085-N*. 514
F.36 WUS 36: *1529_TCU102-E* and *1529_TCU102-N*. 515
F.37 WUS 37: *155_F-BEV-EW* and *155_F-BEV-NS*.. 515
F.38 WUS 38: *1551_TCU138-N* and *1551_TCU138-W*.. 516
F.39 WUS 39: *156_F-CSC-EW* and *156_F-CSC-NS*. 516
F.40 WUS 40: *1577_TTN025-E* and *1577_TTN025-N*. 517
F.41 WUS 41: *1585_TTN040-N* and *1585_TTN040-W*.. 517
F.42 WUS 42: *1587_TTN042-N* and *1587_TTN042-W*.. 518
F.43 WUS 43: *1613_1060-E* and *1613_1060-N*. 518
F.44 WUS 44: *1618_531-E* and *1618_531-N*. 519
F.45 WUS 45: *1619_MDR000* and *1619_MDR090*. 519
F.46 WUS 46: *1626_212V5180* and *1626_212V5090*. 520
F.47 WUS 47: *164_H-CPE147* and *164_H-CPE237*. 521
F.48 WUS 48: *1645_mtwin000* and *1645_mtwin090*. 521
F.49 WUS 49: *1649_vquez000* and *1649_vquez090*. 522
F.50 WUS 50: *1691_ANACA000* and *1691_ANACA270*. 522
F.51 WUS 51: *1696_HOW060* and *1696_HOW330*. 523
F.52 WUS 52: *1709_0GPN00E* and *1709_0GPN90W*. 523
F.53 WUS 53: *1715_WON095* and *1715_WON185*. 524
F.54 WUS 54: *1718_LITTL090* and *1718_LITTL360*. 524
F.55 WUS 55: *1720_MCS025* and *1720_MCS115*. 525

F.56 WUS 56: *1727_RANCH180* and *1727_RANCH090*. 525
F.57 WUS 57: *1741_Lsm2000* and *1741_Lsm2270*. 526
F.58 WUS 58: *1745_Lsm6000* and *1745_Lsm6270*. 526
F.59 WUS 59: *1747_Lsm8270* and *1747_Lsm8000*. 527
F.60 WUS 60: *1767_12674090* and *1767_12674360*. 527
F.61 WUS 61: *1786_22T04090* and *1786_22T04180*. 528
F.62 WUS 62: *1795_12647090* and *1795_12647180*. 529
F.63 WUS 63: *1836_22161090* and *1836_22161360*. 529
F.64 WUS 64: *1917_02467360* and *1917_02467090*. 530
F.65 WUS 65: *1942_12116036* and *1942_12116126*. 530
F.66 WUS 66: *1961_13095026* and *1961_13095116*. 531
F.67 WUS 67: *1972_RVB090* and *1972_RVB360*. 531
F.68 WUS 68: *2017_2017090* and *2017_2017360*. 532
F.69 WUS 69: *2019_47006067* and *2019_47006337*. 532
F.70 WUS 70: *2021_57383090* and *2021_57383360*. 533
F.71 WUS 71: *2032_2014090* and *2032_2014180*. 533
F.72 WUS 72: *2046_0438090* and *2046_0438360*. 534
F.73 WUS 73: *2091_ps07039* and *2091_ps07309*. 534
F.74 WUS 74: *2118_1727090* and *2118_1727360*. 535
F.75 WUS 75: *222_B-LMO265* and *222_B-LMO355*. 535
F.76 WUS 76: *225_PFT045* and *225_PFT135*. 536
F.77 WUS 77: *226_TVY045* and *226_TVY135*. 537
F.78 WUS 78: *2296_ILA063-N* and *2296_ILA063-W*. 537
F.79 WUS 79: *23_GGP010* and *23_GGP100*. 538
F.80 WUS 80: *2336_TAP077-N* and *2336_TAP077-W*. 538
F.81 WUS 81: *2367_TCU045-E* and *2367_TCU045-N*. 539
F.82 WUS 82: *2396_TCU085-E* and *2396_TCU085-N*. 539

F.83 WUS 83: *2423_TCU129-E* and *2423_TCU129-N*. 540
F.84 WUS 84: *2427_TCU138-N* and *2427_TCU138-W*. 540
F.85 WUS 85: *2439_TTN025-E* and *2439_TTN025-N*. 541
F.86 WUS 86: *2445_TTN040-N* and *2445_TTN040-W*. 541
F.87 WUS 87: *2447_TTN042-N* and *2447_TTN042-W*. 542
F.88 WUS 88: *2601_TCU045-E* and *2601_TCU045-N*. 542
F.89 WUS 89: *2633_TCU085-E* and *2633_TCU085-N*. 543
F.90 WUS 90: *2640_TCU102-E* and *2640_TCU102-N*. 543
F.91 WUS 91: *265_CPE045* and *265_CPE315*. 544
F.92 WUS 92: *2658_TCU129-E* and *2658_TCU129-N*. 545
F.93 WUS 93: *2661_TCU138-W* and *2661_TCU138-N*. 545
F.94 WUS 94: *2677_TTN025-E* and *2677_TTN025-N*. 546
F.95 WUS 95: *2685_TTN040-W* and *2685_TTN040-N*. 546
F.96 WUS 96: *2687_TTN042-N* and *2687_TTN042-W*. 547
F.97 WUS 97: *2805_KAU003-N* and *2805_KAU003-W*. 547
F.98 WUS 98: *283_A-ARI000* and *283_A-ARI270*. 548
F.99 WUS 99: *284_A-AUL000* and *284_A-AUL270*. 548
F.100 WUS 100: *285_A-BAG000* and *285_A-BAG270*. 549
F.101 WUS 101: *286_A-BIS000* and *286_A-BIS270*. 549
F.102 WUS 102: *2877_TCU102-E* and *2877_TCU102-N*. 550
F.103 WUS 103: *2897_TCU138-N* and *2897_TCU138-W*. 550
F.104 WUS 104: *2919_TTN025-E* and *2919_TTN025-N*. 551
F.105 WUS 105: *292_A-STU000* and *292_A-STU270*. 551
F.106 WUS 106: *2927_TTN040-N* and *2927_TTN040-W*. 552
F.107 WUS 107: *2929_TTN042-W* and *2929_TTN042-N*. 553
F.108 WUS 108: *293_A-TDG000* and *293_A-TDG270*. 553
F.109 WUS 109: *295_B-AUL000* and *295_B-AUL270*. 554

F.110 WUS 110: *296_B-BAG000* and *296_B-BAG270*. 554
F.111 WUS 111: *297_B-BIS000* and *297_B-BIS270*.. 555
F.112 WUS 112: *2996_HWA003-N* and *2996_HWA003-W*.. 555
F.113 WUS 113: *303_B-STU000* and *303_B-STU270*. 556
F.114 WUS 114: *3139_TAP077-N* and *3139_TAP077-W*. 556
F.115 WUS 115: *3172_TCU045-E* and *3172_TCU045-N*.. 557
F.116 WUS 116: *3194_TCU085-E* and *3194_TCU085-N*.. 557
F.117 WUS 117: *3202_TCU102-E* and *3202_TCU102-N*.. 558
F.118 WUS 118: *3217_TCU129-E* and *3217_TCU129-N*.. 558
F.119 WUS 119: *3220_TCU138-N* and *3220_TCU138-W*. 559
F.120 WUS 120: *3241_TTN025-E* and *3241_TTN025-N*.. 559
F.121 WUS 121: *3249_TTN040-N* and *3249_TTN040-W*. 560
F.122 WUS 122: *3251_TTN042-W* and *3251_TTN042-N*. 561
F.123 WUS 123: *3325_HWA003-N* and *3325_HWA003-W*.. 561
F.124 WUS 124: *3390_ILA063-N* and *3390_ILA063-W*. 562
F.125 WUS 125: *3479_TCU085-E* and *3479_TCU085-N*.. 562
F.126 WUS 126: *3489_TCU102-E* and *3489_TCU102-N*.. 563
F.127 WUS 127: *3507_TCU129-E* and *3507_TCU129-N*.. 563
F.128 WUS 128: *3509_TCU138-N* and *3509_TCU138-W*. 564
F.129 WUS 129: *3532_TTN025-E* and *3532_TTN025-N*.. 564
F.130 WUS 130: *3540_TTN040-N* and *3540_TTN040-W*. 565
F.131 WUS 131: *3542_TTN042-N* and *3542_TTN042-W*. 565
F.132 WUS 132: *369_H-SCN045* and *369_H-SCN315*.. 566
F.133 WUS 133: *43_CSM095* and *43_CSM185*.. 566
F.134 WUS 134: *443_CEM000* and *443_CEM090*. 567
F.135 WUS 135: *444_HAU000* and *444_HAU090*.. 567
F.136 WUS 136: *45_DCF090* and *45_DCF180*.. 568

F.137 WUS 137: *454_GIL067* and *454_GIL337*. 569
F.138 WUS 138: *455_G01230* and *455_G01320*. 569
F.139 WUS 139: *459_G06000* and *459_G06090*. 570
F.140 WUS 140: *476_LOB050* and *476_LOB320*. 570
F.141 WUS 141: *49_SAD003* and *49_SAD273*. 571
F.142 WUS 142: *495_S1010* and *495_S1280*. 571
F.143 WUS 143: *496_S2240* and *496_S2330*. 572
F.144 WUS 144: *497_S3270* and *497_S3360*. 572
F.145 WUS 145: *501_D-SG3205* and *501_D-SG3295*. 573
F.146 WUS 146: *511_ARM270* and *511_ARM360*. 573
F.147 WUS 147: *512_ATL270* and *512_ATL360*. 574
F.148 WUS 148: *525_LMR252* and *525_LMR162*. 574
F.149 WUS 149: *528_H01000* and *528_H01090*. 575
F.150 WUS 150: *536_ARS270* and *536_ARS360*. 575
F.151 WUS 151: *537_SIL000* and *537_SIL090*. 576
F.152 WUS 152: *541_H02000* and *541_H02090*. 577
F.153 WUS 153: *585_CPE161* and *585_CPE251*. 577
F.154 WUS 154: *59_CSM095* and *59_CSM185*. 578
F.155 WUS 155: *63_FTR056* and *63_FTR326*. 578
F.156 WUS 156: *631_A-CHL030* and *631_A-CHL120*. 579
F.157 WUS 157: *643_A-WON075* and *643_A-WON165*. 579
F.158 WUS 158: *661_A-ANG000* and *661_A-ANG090*. 580
F.159 WUS 159: *663_A-MTW000* and *663_A-MTW090*. 580
F.160 WUS 160: *67_ISD014* and *67_ISD284*. 581
F.161 WUS 161: *703_A-VAS000* and *703_A-VAS090*. 581
F.162 WUS 162: *715_B-MTW000* and *715_B-MTW090*. 582
F.163 WUS 163: *72_L04111* and *72_L04201*. 582

F.164 WUS 164: *73_L09021* and *73_L09291*. 583
F.165 WUS 165: *763_GIL337* and *763_GIL067*. 583
F.166 WUS 166: *765_G01000* and *765_G01090*. 584
F.167 WUS 167: *769_G06000* and *769_G06090*. 585
F.168 WUS 168: *77_PUL164* and *77_PUL254*. 585
F.169 WUS 169: *782_MCH000* and *782_MCH090*. 586
F.170 WUS 170: *788_PJH045* and *788_PJH315*. 586
F.171 WUS 171: *789_PTB207* and *789_PTB297*. 587
F.172 WUS 172: *791_SG3261* and *791_SG3351*. 587
F.173 WUS 173: *795_PHT270* and *795_PHT360*. 588
F.174 WUS 174: *797_RIN000* and *797_RIN090*. 588
F.175 WUS 175: *798_TLH000* and *798_TLH090*. 589
F.176 WUS 176: *801_SJTE225* and *801_SJTE315*. 589
F.177 WUS 177: *804_SSF115* and *804_SSF205*. 590
F.178 WUS 178: *809_UC2000* and *809_UC2090*. 590
F.179 WUS 179: *810_LOB000* and *810_LOB090*. 591
F.180 WUS 180: *813_YBI000* and *813_YBI090*. 591
F.181 WUS 181: *828_PET000* and *828_PET090*. 592
F.182 WUS 182: *87_SAD003* and *87_SAD273*. 593
F.183 WUS 183: *879_LCN260* and *879_LCN345*. 593
F.184 WUS 184: *89_TEH090* and *89_TEH180*. 594
F.185 WUS 185: *891_SIL000* and *891_SIL090*. 594
F.186 WUS 186: *897_29P000* and *897_29P090*. 595
F.187 WUS 187: *922_PPC090* and *922_PPC180*. 595
F.188 WUS 188: *925_RCD090* and *925_RCD180*. 596
F.189 WUS 189: *934_SVP090* and *934_SVP360*. 596
F.190 WUS 190: *938_WBR090* and *938_WBR360*. 597

F.191 WUS 191: *943_ACI000* and *943_ACI270*. 597
F.192 WUS 192: *946_ATB000* and *946_ATB090*. 598
F.193 WUS 193: *957_HOW060* and *957_HOW330*. 598
F.194 WUS 194: *989_CHL070* and *989_CHL160*. 599
F.195 WUS 195: *994_0141-270* and *994_0141-360*. 599
F.196 CEUS 1: *1125A54E* and *1125A54N*. 600
F.197 CEUS 2: *1125A61E* and *1125A61N*. 601
F.198 CEUS 3: *1125A64E* and *1125A64N*. 601
F.199 CEUS 4: *1125S08L* and *1125S08T*. 602
F.200 CEUS 5: *1125S16L* and *1125S16T*. 602
F.201 CEUS 6: *1125S17L* and *1125S17T*. 603
F.202 CEUS 7: *1125S20L* and *1125S20T*. 603
F.203 CEUS 8: *29P000* and *29P090*. 604
F.204 CEUS 9: *A-CSH000* and *A-CSH090*. 604
F.205 CEUS 10: *A-GRN180* and *A-GRN270*. 605
F.206 CEUS 11: *A-GRV060* and *A-GRV330*. 605
F.207 CEUS 12: *A-HMC180* and *A-HMC270*. 606
F.208 CEUS 13: *A-LVL000* and *A-LVL090*. 606
F.209 CEUS 14: *A-MAL180* and *A-MAL270*. 607
F.210 CEUS 15: *A-ORR000* and *A-ORR090*. 607
F.211 CEUS 16: *A-RIV180* and *A-RIV270*. 608
F.212 CEUS 17: *A-SON033* and *A-SON303*. 609
F.213 CEUS 18: *A-VIR200* and *A-VIR290*. 609
F.214 CEUS 19: *A54–N* and *A54–E*. 610
F.215 CEUS 20: *ABY000* and *ABY090*. 610
F.216 CEUS 21: *AMB–X* and *AMB–Y*.. 611
F.217 CEUS 22: *ARC000* and *ARC090*. 611

F.218 CEUS 23: *ARS270* and *ARS360*. 612
F.219 CEUS 24: *ATL270* and *ATL360*. 612
F.220 CEUS 25: *B-A3E146* and *B-A3E236*. 613
F.221 CEUS 26: *B-KOD180* and *B-KOD270*. 613
F.222 CEUS 27: *BOL000* and *BOL090*. 614
F.223 CEUS 28: *BRN000* and *BRN090*. 614
F.224 CEUS 29: *C-ATC270* and *C-ATC360*. 615
F.225 CEUS 30: *C-ATP270* and *C-ATP360*. 615
F.226 CEUS 31: *C-OLC270* and *C-OLC360*. 616
F.227 CEUS 32: *C-TSM270* and *C-TSM360*. 617
F.228 CEUS 33: *C08050* and *C08320*. 617
F.229 CEUS 34: *CAD250* and *CAD340*. 618
F.230 CEUS 35: *CHY080-N* and *CHY080-W*. 618
F.231 CEUS 36: *CLS000* and *CLS090*. 619
F.232 CEUS 37: *CLS000* and *CLS090*. 619
F.233 CEUS 38: *CLS220* and *CLS310*. 620
F.234 CEUS 39: *CPE045* and *CPE315*. 620
F.235 CEUS 40: *CPM000* and *CPM090*. 621
F.236 CEUS 41: *CSM095* and *CSM185*. 621
F.237 CEUS 42: *CSM095* and *CSM185*. 622
F.238 CEUS 43: *CUC090* and *CUC180*. 622
F.239 CEUS 44: *CUC090* and *CUC180*. 623
F.240 CEUS 45: *CYC195* and *CYC285*. 623
F.241 CEUS 46: *D-OLC270* and *D-OLC360*. 624
F.242 CEUS 47: *D-PLM270* and *D-PLM360*. 625
F.243 CEUS 48: *DAY-LN* and *DAY-TR*. 625
F.244 CEUS 49: *DCF090* and *DCF180*. 626

F.245 CEUS 50: *FER-L1* and *FER-T1*. 626
F.246 CEUS 51: *FSD172* and *FSD262*. 627
F.247 CEUS 52: *G06000* and *G06090*. 627
F.248 CEUS 53: *G06000* and *G06090*. 628
F.249 CEUS 54: *GAZ000* and *GAZ090*. 628
F.250 CEUS 55: *GAZ000* and *GAZ090*. 629
F.251 CEUS 56: *GBZ000* and *GBZ270*. 629
F.252 CEUS 57: *GIL067* and *GIL337*. 630
F.253 CEUS 58: *GRN180* and *GRN270*. 630
F.254 CEUS 59: *GYN000* and *GYN090*. 631
F.255 CEUS 60: *H-PG3000* and *H-PG3090*. 631
F.256 CEUS 61: *H-SUP045* and *H-SUP135*. 632
F.257 CEUS 62: *H-TM2000* and *H-TM2090*. 633
F.258 CEUS 63: *H-VC4000* and *H-VC4090*. 633
F.259 CEUS 64: *H-XCV075* and *H-XCV165*. 634
F.260 CEUS 65: *H-Z11000* and *H-Z11090*. 634
F.261 CEUS 66: *H01000* and *H01090*. 635
F.262 CEUS 67: *HCP045* and *HCP135*. 635
F.263 CEUS 68: *HWA023-N* and *HWA023-W*. 636
F.264 CEUS 69: *HWA026-N* and *HWA026-W*. 636
F.265 CEUS 70: *HWA056-N* and *HWA056-W*. 637
F.266 CEUS 71: *ILA031-N* and *ILA031-W*. 637
F.267 CEUS 72: *ILA051-N* and *ILA051-W*. 638
F.268 CEUS 73: *ILA063-N* and *ILA063-W*. 638
F.269 CEUS 74: *ISD014* and *ISD284*. 639
F.270 CEUS 75: *IZN180* and *IZN090*. 639
F.271 CEUS 76: *IZT180* and *IZT090*. 640

F.272 CEUS 77: *KAU078-N* and *KAU078-W*. 641
F.273 CEUS 78: *KBU000* and *KBU090*. 641
F.274 CEUS 79: *KJM000* and *KJM090*. 642
F.275 CEUS 80: *KJM000* and *KJM090*. 642
F.276 CEUS 81: *KSH-L1* and *KSH-T1*.. 643
F.277 CEUS 82: *L-BPL070* and *L-BPL160*. 643
F.278 CEUS 83: *L04111* and *L04201*. 644
F.279 CEUS 84: *LCN260* and *LCN345*.. 644
F.280 CEUS 85: *LGP000* and *LGP090*.. 645
F.281 CEUS 86: *LGP000* and *LGP090*.. 645
F.282 CEUS 87: *LMR162* and *LMR252*. 646
F.283 CEUS 88: *MA1130* and *MA1220*. 646
F.284 CEUS 89: *MA2130* and *MA2220*. 647
F.285 CEUS 90: *MA3130* and *MA3220*. 647
F.286 CEUS 91: *MCD000* and *MCD090*. 648
F.287 CEUS 92: *MDR000* and *MDR090*. 649
F.288 CEUS 93: *MEL090* and *MEL180*. 649
F.289 CEUS 94: *MEL090* and *MEL180*. 650
F.290 CEUS 95: *MEL090* and *MEL180*. 650
F.291 CEUS 96: *NSK-E* and *NSK-N*.. 651
F.292 CEUS 97: *PAC175* and *PAC265*.. 651
F.293 CEUS 98: *PKC090* and *PKC360*. 652
F.294 CEUS 99: *PLC000* and *PLC090*.. 652
F.295 CEUS 100: *PLC258* and *PLC348*. 653
F.296 CEUS 101: *PNG-E* and *PNG-N*.. 653
F.297 CEUS 102: *PUL104* and *PUL194*. 654
F.298 CEUS 103: *RAN000* and *RAN090*. 654

F.299 CEUS 104: *RAN000* and *RAN090*. 655
F.300 CEUS 105: *RIV180* and *RIV270*. 655
F.301 CEUS 106: *RIV180* and *RIV270*. 656
F.302 CEUS 107: *RIV180* and *RIV270*. 657
F.303 CEUS 108: *S1010* and *S1280*. 657
F.304 CEUS 109: *S3270* and *S3360*. 658
F.305 CEUS 110: *SER000* and *SER270*. 658
F.306 CEUS 111: *SER000* and *SER270*. 659
F.307 CEUS 112: *SER000* and *SER270*. 659
F.308 CEUS 113: *SHL000* and *SHL090*. 660
F.309 CEUS 114: *SIL000* and *SIL090*. 660
F.310 CEUS 115: *SKR180* and *SKR090*. 661
F.311 CEUS 116: *SLO234* and *SLO324*. 661
F.312 CEUS 117: *SLO234* and *SLO324*. 662
F.313 CEUS 118: *SOD015* and *SOD285*. 662
F.314 CEUS 119: *SOD015* and *SOD285*. 663
F.315 CEUS 120: *SON033* and *SON303*. 663
F.316 CEUS 121: *SOR225* and *SOR315*. 664
F.317 CEUS 122: *SOR225* and *SOR315*. 665
F.318 CEUS 123: *SRO-NS* and *SRO-WE*. 665
F.319 CEUS 124: *SUL230* and *SUL320*. 666
F.320 CEUS 125: *TAP035-N* and *TAP035-W*. 666
F.321 CEUS 126: *TAP036-N* and *TAP036-W*. 667
F.322 CEUS 127: *TAP059-N* and *TAP059-W*. 667
F.323 CEUS 128: *TAP060-N* and *TAP060-W*. 668
F.324 CEUS 129: *TAP067-N* and *TAP067-W*. 668
F.325 CEUS 130: *TAP069-N* and *TAP069-W*. 669

F.326 CEUS 131: *TAP072-N* and *TAP072-W*. 669
F.327 CEUS 132: *TAP075-N* and *TAP075-W*. 670
F.328 CEUS 133: *TAP078-N* and *TAP078-W*. 670
F.329 CEUS 134: *TCU015-N* and *TCU015-W*. 671
F.330 CEUS 135: *TCU025-N* and *TCU025-W*. 671
F.331 CEUS 136: *TCU046-N* and *TCU046-W*. 672
F.332 CEUS 137: *TCU047-N* and *TCU047-W*. 673
F.333 CEUS 138: *TCU087-N* and *TCU087-W*. 673
F.334 CEUS 139: *TCU089-N* and *TCU089-W*. 674
F.335 CEUS 140: *TCU095-N* and *TCU095-W*. 674
F.336 CEUS 141: *TCU120-N* and *TCU120-W*. 675
F.337 CEUS 142: *TCU128-N* and *TCU128-W*. 675
F.338 CEUS 143: *TCU136-E* and *TCU136-N*. 676
F.339 CEUS 144: *TFS000* and *TFS090*. 676
F.340 CEUS 145: *TMB205* and *TMB295*. 677
F.341 CEUS 146: *VIR200* and *VIR290*. 677
F.342 CEUS 147: *WTW115* and *WTW205*. 678
F.343 CEUS 148: *WWJ090* and *WWJ180*. 678
F.344 CEUS 149: *WWJ090* and *WWJ180*. 679

G.1 The physical layout of some of the GCTS equipment. 682
G.2 The base of the cell with the shear carriage installed and two posts removed. Notice the back pressure sensor mounted at the front. . 683
G.3 Other parts of the cell. Clockwise from bottom left: the top cap, two of the four columns, the cell wall, and the top ring. 684
G.4 The pressure panel. 686
G.5 Clockwise from left: top platen, flask with stopper and spout, brass confining rings, membrane, o-rings, and, at center, bottom platen. 687

G.6 Clockwise from the top left: the internal LVDT assembly, the internal shear LVDT reaction plate and bracket, the normal piston, large and small shaft-locking collars, and the normal actuator contact plate. In the center is the top-platen-to-normal-piston block. 687

G.7 The normal track assembly with the mini-load cell on top. . . . 688

G.8 Mounting the bottom platen onto the shear carriage. 689

G.9 Membrane attached to the bottom platen. 689

G.10 Using the mold as an o-ring stretcher. 690

G.11 Bottom platen with membrane and confining rings. 690

G.12 Bottom bender-element platen with rings and 'fence.' 691

G.13 Flask with sand, stopper, and funnel. 692

G.14 Sample after dry pluviation sand placement. Notice the heaped sand. 693

G.15 Scraping the excess sand off of the top of the confining rings. . . 694

G.16 The mini-vacuum that I use to clean up sand. 694

G.17 The top platen with the top-platen-to-normal-piston block on top of the specimen-in-preparation 695

G.18 Using the beam to straighten the orientation of the top platen. . 696

G.19 Partial assembly of the internal cell components including the black normal-movement guides, the silver-colored stiffener plate, and the internal LVDT reaction plates and other parts. 698

G.20 Bolting the normal track assembly onto the top platen. 699

G.21 The hysteresis loops of a stress versus strain plot. 705

G.22 An example of an output plot. 706

G.23 The 'Test Setup' dialog box. 707

G.24 The 'Simple Shear Program Definition' dialog box. 708

G.25 The 'Test Data Save Options' dialog box. 708

G.26 The new stage dialog box. 709

G.27 The universal program selector dialog box. 710
G.28 The universal programs dialog box. 710
G.29 The duration tab of the Ramp phase dialog box. 711
G.30 The data acquisition tab of the Ramp phase dialog box. . . . 711
G.31 The normal actuator definition dialog box for the Ramp phase. 712
G.32 The shear actuator definition dialog box for the Ramp phase. . 713
G.33 The dialog box for the Consolidation phase. 714
G.34 The shear actuator definition dialog box for the Consolidation phase. 715
G.35 The normal actuator definition dialog box for the Consolidation phase. 716
G.36 The duration tab for the Cyclic phase. 717
G.37 The shear actuator definition dialog box for the Cyclic phase. . 718
G.38 Results of an air friction baseline test. The relationship between friction load and displacement can be approximated with a bi-linear fit. The points on this plot are each from the peaks of the sinusoidal loading. 722
G.39 The 'Analog Inputs' dialog. 723
G.40 The 'Editing Analog Inputs . . . ' window for the normal load cell. 724
G.41 Dialog box for choosing the calibration method. 724
G.42 The 2 point calibration method dialog. 725
G.43 The multiple point calibration method dialog. 726

H.1 The physical layout of some of the GCTS equipment. 732
H.2 The base of the cell with the shear carriage installed and two posts removed. Notice the back pressure sensor mounted at the front.. 733
H.3 Other parts of the cell. Clockwise from bottom left: the top cap, two of the four columns, the cell wall, and the top ring. 734
H.4 The pressure panel.. 736

H.5 Assorted other parts used in a triaxial test. Clockwise from the left: sand-filled flask with spout, 2.8-inch triaxial mold, o-rings, black triaxial pedestal, additional pedestal for 2.8-inch specimens, the top and bottom platens, a latex membrane, a piece of filter paper, the normal piston, a pi tape, the normal actuator contact plate and mating collar, and a mold extender. 737

H.6 The right toolbar in the GCTS main screen allows access to various sensor readouts. 739

H.7 The top toolbar in the GCTS main screen contains the tools needed to interact with the GCTS system. 739

H.8 The Test Setup dialog/window allows you to create and edit test instructions. 739

H.9 The Projects dialog/window allows you to create and view test results. 740

H.10 The Inputs Offset dialog/window allows you to set the values of the sensors. 740

H.11 The PID dialog/window allows you to set the sensitivity of the feedback mechanisms. 741

H.12 The Outputs Offset dialog/window allows you to control the actuators and cell and back pressures. 742

H.13 The Digital Outputs dialog/window allows you to turn the servos on or off. 742

H.14 The Hydraulic dialog/window allows you to turn the hydraulic pump on and off. 743

H.15 The hysteresis loops of a stress versus strain plot. 760

H.16 An example of an output plot. 761

H.17 The 'Test Setup' dialog box. 762

H.18 The 'Simple Shear Program Definition' dialog box. 763

H.19 The 'Test Data Save Options' dialog box. 763

H.20 The new stage dialog box. 764

H.21 The consolidation stage dialog box. 764

H.22 The universal programs dialog box. 765
H.23 Creating a new universal program. 766
H.24 The duration tab of the phase dialog box. 766
H.25 The data acquisition tab of the phase dialog box. 767
H.26 The normal actuator definition dialog box. 767
H.27 The cell pressure definition dialog box. 768
H.28 The back pressure definition dialog box. 769
H.29 The digit outputs section of the phase dialog box. 770
H.30 Results of an air friction baseline test. The relationship between friction load and velocity can be approximated with a tri-linear fit. 773
H.31 The 'Analog Inputs' dialog. 774
H.32 The 'Editing Analog Inputs . . . ' window for the normal load cell. 775
H.33 Dialog box for choosing the calibration method. 775
H.34 The 2 point calibration method dialog. 776
H.35 The multiple point calibration method dialog. 777

List of Tables

3.1 Index properties of the quartz sand used in this study. 77

3.2 Summary of Sinusoidal Tests 77

3.3 Median and Interquartile Range of the Percent Error Prediction for Variable-Amplitude Sinusoidal Loadings 78

3.4 Median and Interquartile Range of the Percent Error Prediction for Earthquake Motions 78

3.5 Median and Interquartile Range of the Percent Error Prediction for All Irregular Motions 78

3.6 Soil Profile at Location B-2 of Site A (after Stewart et al. 2002). 79

3.7 Earthquake Recordings Used in This Case History Study . . . 80

3.8 Estimated Settlements (cm) Across the Entire Profile for Each Input Ground Motion 80

4.1 Regression Coefficients for Lee's (2009) n_{eq} Correlation 121

4.2 Regression Coefficients for n_{eq} Form #1 121

4.3 Standard Deviations of the Regression Coefficients for n_{eq} Form #1 122

4.4 Inter-event, Intra-event and Total Errors from Eqn. 4.15 . . . 122

4.5 Regression Coefficients for n_{eq} Form #2 122

4.6 Standard Deviations of the Regression Coefficients for n_{eq} Form #2 123

4.7 Inter-event, Intra-event and Total Errors from Eqn. 4.17 . . . 123

4.8 Strength of Correlation Between Various Measures of Duration and n_{eq} (WUS Dataset) 123
4.9 Strength of Correlation Between Various Measures of Duration and n_{eq} (CEUS Dataset) 124
4.10 WUS (NGA) Earthquake Motions 136
4.11 Central-Eastern United States (CEUS) Earthquake Motions . . 147

5.1 Regression Coefficients of the r_d Relationship for the WUS Database 168
5.2 Regression Coefficients of the r_d Relationship for the CEUS Database 168
5.3 Values of E for the WUS Dataset 168
5.4 Values of E for the CEUS Dataset 169
5.5 WUS (NGA) Earthquake Motions 177
5.6 Central-Eastern United States (CEUS) Earthquake Motions . . 188

6.1 Regression Coefficients for r_d (Lasley et al. 2015b) 226
6.2 Regression Coefficients for the Fit of the $\Delta W_{1,eq}/G_{max}$ Curves (Clean Sand) 226
6.3 Regression Coefficients for the Fit of the $\Delta W_{1,eq}/G_{max}$ Curves (Sand with High Fines Content) 226
6.4 Regression Coefficients for n_{eq} (Lasley et al. 2015a) 226
6.5 Values Used in the First-Order Approximation of Uncertainty. μ denotes the mean, σ denotes the standard deviation, and δ is the coefficient of variation or σ/μ 228
6.6 Regression Coefficients for the Proposed Limit State Curve . . 228
6.7 Success Rate of Prediction of the Occurrence of Liquefaction for the Case History Database 229
6.8 Regression Coefficients for the Proposed Limit State Curve Using an Alternate G_{max}-$N_{1,60}$ Correlation. The regression coefficients given in Table 6.6 are recommended for forward analyses using SPT blow count. 229

6.9 Success Rate of Prediction of the Occurrence of Liquefaction of Limit State Curves Using Alternative G_{max}-$N_{1,60}$ Correlation . 229
6.10 SPT Case History Database- Earthquake and Site Info 249
6.11 SPT Case History Database-Calculations 258
6.12 Regression Coefficients for n_{eq} (Lasley et al. 2015a) 274
6.13 Regression Coefficients for r_d (Lasley et al. 2015b) 275
6.14 Regression Coefficients for the Fit of the $\Delta W_{1,eq}/G_{max}$ Curves (Clean Sand) 275
6.15 Regression Coefficients for the Fit of the $\Delta W_{1,eq}/G_{max}$ Curves (Sand with High Fines Content) 276

B.1 Details and Calibration Coefficients of All Constant-Amplitude CSS Tests 290
B.2 Cycles to Various Levels of Volumetric Strain 301
B.3 Summary of Irregular-loading Seismic Compression CSS Tests . 313

C.1 Stepwise Illustration of the Rainflow Counting Algorithm . . . 356
C.2 Comparison of Cycle Counting Methods As Used with Seismic Compression Methods 356

D.1 Index Properties of the Sands Tested. 369
D.2 Mean and Standard Deviations of Regression Coefficients for the $\Delta W_{\text{Effect.}}/\sigma'_{v0}$-$D_r$ Relationship Obtained Via Bootstrapping . . 369
D.3 Results of Welch's t-tests 369
D.4 Results of stress-controlled sinusoidal CV-CSS tests. 390
D.5 Results of Stress-Controlled Earthquake-Loading Constant Volume Cyclic Simple Shear Tests 392
D.6 Results of Strain-Controlled CV-CSS Tests 394

Chapter 1

Introduction

Loose, granular soils have a tendency to contract when cyclically loaded (e.g. as in an earthquake). In saturated soils, this tendency can lead to the generation of excess pore water pressure and the consequent loss of shear strength. This process is called cyclic softening and, if the process continues, the soil may liquefy. In unsaturated soils, this same tendency to contract leads to seismic compression, a decrease in soil volume. Various methodologies have been proposed to evaluate the potential for liquefaction and seismic compression, many of which are loosely based on fatigue theories. The objective of the research presented herein is the more rigorous and consistent application of specific fatigue theories to these important geotechnical earthquake engineering phenomena. In particular: 1. The development of a method to estimate seismic compression using a modified version of the non-linear Richart-Newmark (Richart and Newmark 1948) cumulative damage hypothesis, and 2. The development of a dissipated-energy-based liquefaction evaluation procedure based on a low-cycle fatigue application of the Palmgren-Miner (Palmgren 1924; Miner 1945) hypothesis. This chapter introduces the research presented in this dissertation and details the objectives of this research. Throughout this chapter and dissertation, the terms "simplified" and "non-simplified" are used to describe seismic compression and liquefaction evaluation procedures. In this context, a "non-simplified" procedure requires a site response analysis to compute the induced stresses/strains in the soil profile where a "simplified" procedure does not require these site-specific analyses.

1.1 Soil Liquefaction

Liquefaction occurs when the soil skeleton of a loose, saturated sand contracts (e.g., as a result of earthquake shaking), with the self-weight of the soil particles and the overburden load being transferred to the pore water. As the overburden load is transferred to the pore water, the pore water pressure increases and the shear strength of the soil is temporarily reduced. Possible consequences of this loss of strength include bearing capacity failures of foundations, slope failures, and settlements; the safety of buildings, bridges, dams, and many other structures is often affected.

The damaging effects of liquefaction were realized during the 1964 Niigata, Japan earthquake. During this event, liquefaction caused apartment buildings to tilt dangerously on their sides, underground tanks to rise to the surface, and costly damage to other buildings and infrastructure (Seed and Idriss 1967). More recently, at the Port-au-Prince, Haiti seaport, liquefaction-damaged piers (via lateral-spreading) hampered the delivery of relief supplies (Rathje et al. 2010; see Figure 1.1). In most large earthquakes, liquefaction inflicts heavy economic damages (see Figure 1.2) and potentially threatens lives.

The current state-of-practice for assessing the potential for liquefaction is to use in-situ index test data (e.g. SPT blow counts, CPT logs, shear wave velocity profiles) and expected earthquake loading in conjunction with the stress-based simplified procedure. The stress-based simplified procedure was proposed by Whitman (1971) and Seed and Idriss (1971), and subsequently modified by many others (e.g. Youd et al. 2001). At its core, it is based on the Palmgren-Miner (Palmgren 1924; Miner 1945) cumulative damage hypothesis, a well-used fatigue theory in earthquake engineering.

While the stress-based simplified procedure is useful in liquefaction potential evaluations, it has several limitations. For example, in implementing the procedure in a non-simplified form (i.e. computing the induced stresses using a site response analysis) only the estimation of the induced amplitude of the load is refined, not the duration of the load. Also, applicability of the procedure for evaluating motions other than active shallow crustal earthquakes (e.g. subduction zone or stable continental earthquakes), non-tectonic earthquake motions (e.g. induced seismicity), or non-earthquake motions (e.g. blasting) is questionable.

The energy-based liquefaction evaluation procedure presented herein implements an established low-cycle metal fatigue theory (an alternative implementation of the Palmgren-Miner hypothesis; Green and Terri 2005) that is easily calibrated for a range of soils, stress conditions, and earthquake ground motions. The damage metric for the theory is dissipated energy, the cumulative area bound by stress-strain hysteresis curves, and dissipated energy has been shown to be a good indicator of excess pore pressure and the resulting liquefaction (Nemat-Nasser and Shokooh 1979) irrespective of the shape of the loading history (Liang et al. 1995). By using this well-established low-cycle fatigue theory with dissipated energy as the damage metric, the energy-based liquefaction evaluation procedure proposed herein can be applied in tectonic settings that are under-represented in the current liquefaction case history database or for applications dealing with non-earthquake cyclic loading, such as the design of ground improvement programs.

1.2 Seismic Compression

Seismic compression is the accrual of volumetric strain that occurs when a loose, unsaturated sand is cyclically loaded (e.g. from earthquake shaking). Compared to liquefaction, damages due to seismic compression are relatively small. This is because shear strengths are never completely lost in the soil, and seismic compressions occur only in the relatively shallow depths above the water table. However, Stewart et al. (2001) documented hundreds of cases of permanent vertical and horizontal ground displacements in structural fills. They suggested that many of the these displacements were caused by seismic compression, and concluded that seismic compression poses a significant risk to engineered structural fills in particular.

Currently, several methods are available to estimate the magnitude of seismic compression (e.g. Tokimatsu and Seed 1987; Duku 2008). These are mainly "simplified" methods that rely on a number-of-equivalent-cycles correlation to represent duration. In many cases, the number of equivalent cycles is computed inconsistently with the underlying mechanics of seismic compression. Proposed herein is a "non-simplified" method that was developed using the Richart-Newmark (1948) cumulative damage hypothesis in conjunction with the results of cyclic direct simple shear tests performed at Virginia Tech. The proposed method can predict seismic compression for any arbitrary cyclic

loading.

1.3 Objectives

As stated previously, the overall objectives of the research presented herein are: 1. the development of a seismic compression estimation method based the Richart-Newmark (R-N; Richart and Newmark 1948) cumulative damage hypothesis, and 2. the development of an energy-based liquefaction evaluation procedure based on low-cycle fatigue principles.

In order to achieve the first objective, over 450 cyclic simple shear laboratory tests have been performed on dry sand specimens at various relative densities and overburden pressures. From the results of sinusoidal tests, the proposed R-N-based model was calibrated. From other cyclic simple shear test performed with variable-amplitude sinusoidal and earthquake loading functions, the calibrated model was validated.

For the second objective, equivalent-linear site response analysis were conducted using 50 soil profiles and 228 pairs of earthquake motions (representing two different tectonic regimes). From the resulting site responses, a number of equivalent cycles correlation was developed based on low-cycle fatigue and dissipated energy. A stress reduction coefficient correlation was also developed from the site response analyses. With the number of equivalent cycles correlation, the stress reduction coefficient correlation, and a liquefaction/no-liquefaction case history database, the energy-based liquefaction evaluation procedure was developed. Finally, the results of constant-volume cyclic simple shear tests support the form of the liquefaction evaluation procedure.

1.4 Organization

This dissertation is organized as a collection of four manuscripts with three introductory chapters and one closing chapter. Chapter 2 gives a brief literature review. A large portion of the chapter gives a historical overview of seismic compression research and outlines procedures that are currently in use for the evaluation of seismic compression. Another large portion of the chapter provides a historical overview of energy-based liquefaction evaluation

procedures. Lesser sections of the chapter provide background information for the subsequent chapters and discuss topics such as field and laboratory soil testing, the stress-based simplified procedure, and equivalent-linear site response analyses.

Chapter 3 is the first of four manuscripts that have been or will soon be submitted for publication. It presents the aforementioned method to estimate seismic compression using the Richart-Newmark (Richart and Newmark 1948) cumulative damage hypothesis. It also contains a discussion on fatigue hypotheses including the Richart-Newmark and Palmgren-Miner (1924; 1945) hypotheses. The results of the method presented therein are compared with those of the Byrne (1991) and Palmgren-Miner methods, and the newly proposed method provides the best overall results of the three methods.

Chapter 4 contains the second manuscript. It presents a new number of equivalent cycles (n_{eq}) correlation which is a function of the peak ground acceleration and earthquake magnitude. The proposed n_{eq} correlation is based on the Palmgren-Miner cumulative damage hypothesis, like many other n_{eq} correlations in geotechnical earthquake engineering. However, instead of assuming high-cycle fatigue conditions (many cycles, little damage in each cycle), the proposed correlation assumes low-cycle fatigue conditions (few cycles to failure; significant damage in each cycle) and uses dissipated energy as the damage metric. Correlations are presented for both shallow-crustal tectonic regimes (e.g. the western United States) and stable continental regimes (e.g. the central-eastern United States).

Chapter 5 is a technical note that introduces a new stress reduction coefficient (r_d) relationship. The stress reduction coefficient allows for the computation of the seismically induced stresses at depth in a soil profile without the need to perform numerical site response analyses. As such, r_d is an integral part of the simplified liquefaction evaluation procedure. The stress reduction coefficient relationship proposed therein yields values having less bias and uncertainty than those of other commonly-used r_d relationships. Also, relationships are presented for both shallow-crustal tectonic regimes (e.g. the western United States) and stable continental regimes (e.g. the central-eastern United States).

Chapter 6 presents a simplified liquefaction evaluation procedure based on dissipated energy and the n_{eq} correlation described in Chapter 4. The pro-

posed procedure is a function of peak ground acceleration, SPT blow count, earthquake magnitude and fines content. It is presented in a manner that will be familiar to current users of the stress-based simplified procedure. For the liquefaction case history database from which it was derived, the proposed procedure is as successful at predicting liquefaction/no-liquefaction as other stress-based simplified procedures. Additionally, because the proposed liquefaction evaluation procedure is based on dissipated energy, it can more easily be applied to sites outside the tectonic regime from which it was derived or used to evaluate liquefaction potential for non-earthquake loadings.

A summary of the significance of the research findings and conclusions are given in Chapter 7. Additional research details are given in the numerous appendices.

1.5 Significance

Earthquakes present a serious risk to life and property for large portions of the world's population. A significant part of this risk can be attributed to the possible effects of liquefaction and seismic compression. The research presented herein brings powerful fatigue theories to bear on the evaluations of liquefaction potential and seismic compression. By providing more accurate and versatile evaluation methods, limited resources can be applied in the best possible manner to reduce this risk.

Both the seismic compression and the liquefaction evaluation procedures proposed herein can be used in a "non-simplified" manner. Barriers to the application of numerical site response analyses (such as the equivalent-linear procedure) have been reduced in recent years. Cost-effective methods for site characterization for dynamic analyses are becoming more prevalent as are protocols for selecting input motions and performing site response analyses (e.g. Kottke and Rathje 2008; Bradley 2012; Stewart and Kwok 2008). As these site response analyses become more common, the usefulness of procedures that can compatibly perform both "simplified" and "non-simplified" evaluations grows. Although not developed as part of this research, the modified R-N model developed for seismic compression estimation can be used to develop a "simplified" procedure with a consistent damage metric (i.e. volumetric strain).

In addition to providing better earthquake-induced liquefaction and seismic compression evaluations, the procedures proposed herein can be used with any cyclic loading for which stress and strain time histories can be obtained: from machine foundation design to remedial ground densification.

1.6 References

Bradley B.A. (2012). "A ground motion selection algorithm based on the generalized conditional intensity measure." *Soil Dynamics and Earthquake Engineering*, 40, 48-61.

Byrne, P. (1991). "A cyclic shear-volume coupling and pore pressure model for sand." *Proceedings of the Second International Conference on Recent Advances in Geotechnical Earthquake Engineering and Soil Dynamics*, St. Louis, MO, March, Paper, 47–55.

Duku, P. M., Stewart, J. P., Whang, D. H., and Yee, E. (2008). "Volumetric strains of clean sands subject to cyclic loads." Journal of geotechnical and geoenvironmental engineering, 134(8), 1073–1085.

gCaptain. (2010). "Port Au Prince Haiti, Earthquake Damage – Incident Photo Of The Week." http://gcaptain.com/port-prince-earthquake-damage/

Green, R. A., and Terri, G. A. (2005). "Number of equivalent cycles concept for liquefaction evaluations—Revisited." *Journal of Geotechnical and Geoenvironmental Engineering*, 131(4), 477–488.

Kottke, A. and Rathje, E.M. (2008). "A semi-automated procedure for selecting and scaling recorded earthquake motions for dynamic analysis." *Earthquake Spectra*, 24 (4), 911-932.

Liang, L., Figueroa, J. L., and Saada, A. S. (1995). "Liquefaction under random loading: Unit energy approach." *Journal of Geotechnical Engineering*, 121(11), 776–781.

Miner, M. A. (1945). "Cumulative Damage in Fatigue." *Journal of Applied Mechanics*, 12(3), A159-A164.

Nemat-Nasser, S., and Shokooh, A. (1979). "A unified approach to densification and liquefaction of cohesionless sand in cyclic shearing." *Canadian Geotechnical Journal*, 16(4), 659–678.

New Zealand Defence Force. (2011). "CHCH City - Streets1." https://www.flickr.com/photos/nzdefenceforce/5469816702/.

Palmgren, A. (1924). "Die lebensdauer von kugellagern (Life Length of Roller Bearings, In German)." *Zeitschrift des Vereins Deutscher Ingenieure*, 68(14), 339–341.

Rathje, E., J. Bachhuber, B. Cox, J. French, R. A. Green, S. Olson, Glenn Rix, D. Wells, and O. Suncar. (2010). "Geotechnical Reconnaissance of the 2010 Haiti Earthquake." http://www.geerassociation.org/, http://www.geerassociation.org/.

Richart, F.E. and Newmark, N.M. (1948). "An hypothesis for determination of cumulative damage in fatigue," *ASTM Proceedings*, 48: 767-800.

Seed, H. B., and I. M. Idriss. (1967). "Analysis of Soil Liquefaction: Niigata Earthquake." *Journal of the Soil Mechanics and Foundations Division*, 93(SM3), 83–108.

Seed, H. B., and Idriss, I. M. (1971). "Simplified Procedure for Evaluating Soil Liquefaction Potential." *Journal of the Soil Mechanics and Foundations Division*, 97(9), 1249–73.

Stewart, J. P., and Kwok, A. O. (2008). "Nonlinear seismic ground response analysis: Code usage protocols and verification against vertical array data." *Geotechnical Earthquake Engineering and Soil Dynamics IV* (D. Zeng, M. T. Manzari, and D. R. Hiltunen, eds.), ASCE, Reston, VA, 1-24.

Tokimatsu, K., and Seed, H. B. (1987). "Evaluation of settlements in sands due to earthquake shaking." *Journal of Geotechnical Engineering*, 113(8), 861–878.

Whitman, R. V. (1971). "Resistance of Soil to Liquefaction and Settlement." *Soils and Foundations*, 11(4), 59–68.

Youd, T. L., Idriss, I. M., Andrus, R. D., Arango, I., Castro, G., Christian, J. T., Dobry, R., Finn, W. D. L., Harder, L. F., Hynes, M. E., Ishihara, K., Koester, J. P., Liao, S. S. C., Marcuson, W. F., Martin, G. R., Mitchell, J. K., Moriwaki, Y., Power, M. S., Robertson, P. K., Seed, R. B., and Stokoe, K. H. (2001). "Liquefaction Resistance of Soils: Summary Report from the 1996 NCEER and 1998 NCEER/NSF Workshops on Evaluation of Liquefaction Resistance of Soils." *Journal of Geotechnical and Geoenvironmental Engineering*, 127(10), 817–833.

1.7 Figures

Figure 1.1: Liquefaction during the 2010 M_w7.0 earthquake caused damage at the Port-au-Prince seaport, hampering relief efforts. (gCaptain 2010) [Fair Use]

Figure 1.2: Liquefaction during the Christchurch earthquake sequence led to the condemnation of entire neighborhoods (NZDF 2011). [Fair Use]

Chapter 2

Literature Review

2.1 Introduction

This chapter presents an overview of and background information on the concepts central to the subsequent manuscripts. Some of the topics are introduced in detail in the manuscripts themselves; this chapter contains the subjects that are not covered in sufficient detail in other chapters. This chapter is organized as follows: first, a brief introduction on laboratory and in-situ soil testing is given followed by a detailed review of historic seismic compression research and evaluation methods. The equivalent-linear site response analysis is then presented. Next, a brief introduction to stress-based liquefaction evaluation methods is presented. Finally, a historic overview of energy-based liquefaction evaluation procedures is given.

2.2 Soil Testing

The testing of soils in order to characterizing their engineering behavior is generally divided into two categories: in-situ testing and laboratory testing. In-situ testing provides information about the current state of a soil deposit and, under some conditions, provides information on engineering parameters of soils. Laboratory testing is used to characterize the behavior of the soil under various loading conditions. This information is required to make competent design decisions and allow for modeling and predictions. Geotech-

nical earthquake engineers are mainly concerned with the composition of the soil (e.g. grain size distribution and plasticity), the in-situ density, the strain-dependent stiffness and damping of the soil, and ultimately the cyclic strength of the soil. This section provides an overview of the soil testing that are relevant to characterizing the seismic compression and liquefaction behavior of soils. Laboratory testing will be discussed first, followed by in-situ testing.

2.2.1 Laboratory Testing

Laboratory tests are performed on soil samples obtained from field investigations and can be generally placed in one of two categories: index testing and characterization of the stress-strain behavior of the soil. Index testing provides information on the inherent properties of the aggregate soil particles, irrespective of the packing of the particles. For example, common index tests determine grain size distribution, Atterberg limits (liquid limit and plastic limit), minimum and maximum void ratios (void ratio is defined as the volume of voids over the volume of solids), particle angularity, and particle composition. Since index tests measure inherent properties of the aggregate soil, they can be performed on disturbed soil samples.

Characterization of stress-strain behavior involves the evaluation of the stress-strain behavior of a soil under a given loading condition and drainage condition. Geotechnical earthquake engineers are interested in the cyclic strength of soils under undrained conditions for liquefaction analysis and under drained conditions for seismic compression analysis. The soils most susceptible to liquefaction and seismic compression are sands, silts, and gravels. The cyclic strength of these soils is evaluated by applying a dynamic loading to the soil specimen until failure occurs. In most cases, this loading is sinusoidal, but several other load shapes are also used in practice and research.

The fabric of the soil can have a profound effect on the results of these strength tests (Ladd 1974; Seed et al. 1975; Mulilis et al. 1977), so undisturbed soil samples give the most reliable results. However, undisturbed sampling of liquefaction- and seismic compression-susceptible soils is, in most cases, prohibitively expensive and, under some circumstances, not possible. Where undisturbed samples are unavailable, soil test specimens must be re-

constituted from disturbed soil samples. For liquefaction evaluations, soil specimens are commonly reconstituted by dry or wet tamping (soil is placed into a mold then tamped to density; Ladd 1978), dry or wet pluviation (soil is pored into the mold; Vaid and Negussey 1984; Vaid et al. 1999), or, for soils with fines, slurry deposition (Kuerbis and Vaid 1988). Sample preparation should aim to mimic natural soil deposition and soil fabric.

In laboratory testing, the relative density is often used as a measure of the soil density. It is defined as:

$$D_r = \frac{e_{max} - e}{e_{max} - e_{min}} \times 100\% \tag{2.1}$$

where e, e_{max}, and e_{min} are the void ratio, the maximum void ratio, and the minimum void ratio, respectively. e_{max} and e_{min} are determined using standardized index tests.

A number of different laboratory tests are used to evaluate the cyclic strength of a soil. These tests vary in the ways that static and dynamic stresses are applied to a specimen and each have their own strengths and weaknesses. In this section, cyclic triaxial, cyclic simple shear, and cyclic hollow cylinder torsional shear tests will be briefly detailed. Finally, a few common cyclic failure criteria are enumerated.

Cyclic Triaxial

Triaxial tests have a long history in geotechnical engineering (Holtz et al. 2010). Triaxial soil specimens are cylindrically-shaped with a height roughly twice that of the diameter (ASTM 2013, Standard D5311). The specimens are enclosed in a rubber membrane (see Figure 2.1), and confining stress (outside the membrane) and pore pressure (inside the membrane) can be controlled independently. Cyclic triaxial tests are predominantly used for liquefaction evaluations (not for seismic compression), and for reliable results, the specimen must be saturated with no sizeable air voids present. This is often obtained via back-pressure saturation. In back pressure saturation, the samples are first flushed with CO_2 and both the confining stress and pore pressure are increased until the air voids are sufficiently small to have little effect. While the back pressure saturation occurs, a positive effective confining stress (confining stress minus pore pressure) is

maintained. After the specimen is saturated, it is allowed to consolidate under a target effective confining stress.

Once consolidation has completed, a cyclic axial stress is applied at the top of the specimen until failure occurs (failure criteria are discussed in a subsequent section). Drainage from the specimen is not allowed to occur during the cyclic loading phase (i.e. the pore water pressure is allowed to rise), and the pore water pressure is recorded throughout the cyclic phase of the test. For more information, see Appendix H.

Cyclic Simple Shear

In a simple shear test, a stout cylindrical specimen is enclosed by rings or a reinforced membrane (see Figure 2.2). The diameter of the specimen should be at least 2.5 times greater than the height (ASTM 2007). A few variants of the cyclic simple shear (CSS) test are possible. For an ordinary CSS test for liquefaction, a saturated sample is prepared and a confining stress is applied via a normal confining stress. Consolidation is allowed to occur; drainage lines are left open while consolidation continues. After consolidation, drainage lines are closed and the normal confining stress is held constant while a cyclic shear load is applied to the top or the bottom of the specimen, depending on the configuration of the testing apparatus. Cyclic loading continues until failure occurs, and pore water pressures are recorded throughout the cyclic phase of the test.

Another variant of the CSS is the constant-volume (C-V) cyclic simple shear test (Finn and Vaid 1977). A C-V CSS test can be performed on a wet or dry specimen. Like an ordinary CSS test, a normal consolidation stress is applied to consolidate the specimen with drainage lines open. After consolidation, however, the normal displacement is held constant while the cyclic shear loading is applied and drainage lines are left open. Instead of measuring the pore water pressure, the normal stress is recorded and allowed to change during the cyclic phase of the test. The test is based on the premise that the decrease in the normal stress during loading is equal to the increase in pore water pressure for an ordinary CSS test (Finn and Vaid 1977; Finn et al. 1979; Dyvik et al. 1987).

One benefit of performing constant-volume cyclic simple shear tests is that the specimen does not need to be saturated and no enclosing cell is needed.

This allows for specimens to be prepared and tested in much less time compared to the ordinary CSS and triaxial tests. For more information on specific test procedures used during this research, see Appendix G.

The induced shear stresses and strains of cyclic simple shear tests are more representative of earthquake-induced stresses and shear strains (Kramer 1996) than those of cyclic triaxial tests because the CSS apparatus can directly apply shear stresses to the horizontal planes of a specimen. However, due to the confining rings or reinforced membranes of conventional (and C-V) CSS tests, the normal confining stresses applied to the sides of the specimen are not controlled; only the normal vertical stresses that are applied to the top and the bottom of the specimen can be controlled.

Hollow Cylinder Torsional Shear

Hollow cylinder torsional shear tests are performed on specimens that are shaped like a hollow cylinder, open at the top and the bottom. The inside and outside of the hollow specimen are membrane covered, allowing for any combination of normal confining stresses to the top and sides of the specimen. During cyclic loading shear stresses are imposed on horizontal planes of the specimen via a torque applied at the top (Holtz et al. 2010). Like cyclic triaxial tests, most hollow cylinder tests are performed on saturated specimens and drainage is closed during the cyclic loading phase of the test. These tests are sometimes simply called 'torsional shear tests.' These are the least common of the three laboratory tests discussed in section.

Failure Criteria in Liquefaction Testing

Several different failure criteria are used in laboratory testing to define the onset of liquefaction. These liquefaction criteria can be categorized as follows (Wu et al. 2004):

1. Pore Water Pressure-Based: Initial liquefaction is considered to have occurred when the pore water stress is equal to the confining stress. This can also be stated as when the residual excess pore water pressure is equal to the initial effective confining stress or when the residual excess pore water pressure ratio (r_u) is equal to 1.

2. Strength-Based Criteria: Liquefaction is considered to have occurred when a significant amount of strength is lost. The threshold value of strength will vary depending on application and judgment.

3. Strain-Based Criteria: Liquefaction is considered to have occurred when a certain amplitude of strain or deformation has occurred. This criteria is similar to the strength-based criteria because strain increases as strength decreases. As with the strength-based criteria, the threshold strain value varies between researchers with some using 3% single-amplitude and others using up to 20% double-amplitude strain as the liquefaction initiation criteria.

For the cyclic simple shear tests discussed in Chapter 6 of this work, a strain-based criteria of 3.75% single-amplitude strain was used (Vaid and Sivathayalan 1996; Sivathayalan and Ha 2011). The failure criteria are not necessary for drained seismic compression tests because catastrophic loss of strength does not occur during cyclic loading. Instead, volumetric strains are the damage metric and are recorded throughout the cyclic phase. Seismic compression tests are halted when designated volumetric strains have occurred.

2.2.2 In Situ Testing

Because undisturbed soil specimens are difficult and costly to obtain, in-situ soil tests have an important role in evaluating liquefaction and seismic compression susceptibility. In-situ tests provide information such as soil type stratigraphy, in-situ density, and depth to the water table. From these details an engineer can identify liquefaction- and seismic compression-susceptible soil layers in the profile. Many in-situ tests have been proposed and used over the years. This section will detail two common in-situ tests: the standard penetration test and the cone penetrometer test.

Standard Penetration Test

The standard penetration test, or SPT, is a test commonly used for standard geotechnical site investigations and liquefaction potential evaluation. It is standardized by ASTM standard 1586 (2011). To perform the test, a borehole must first be drilled from the ground surface to the depth of interest. Next,

a 'split-spoon' (or 'split barrel') sampler is lowered to the bottom of the borehole by means of driving rods. The split-spoon sampler has an outer diameter of 5.1 cm (2 inches) and inner diameter of 3.5 cm (1.375 inches). The sampler is driven into the bottom of the borehole by dropping a 0.623 kN (140 pound) weight a distance of 76.2 cm (30 inches). It is driven a total of 45.7 cm (18 inches) into the bottom of the borehole, and the number of 'blows' required to drive the sampler the last 30.5 cm (12 inches) is the measured SPT blow count N_m.

The earliest stress-based simplified procedures (e.g. Whitman 1971) used SPT blow count as a proxy for soil density, and it remains a common in-situ test for liquefaction evaluations. Besides being ubiquitous, the SPT also returns a disturbed soil sample which is useful for soil type identification and index testing. Because the test is common, many correlations exist for the SPT blow count and most engineers are familiar with it. However, the site profiling from SPT testing is discrete (or non-continuous); thin layers may be missed entirely because testing is generally performed at depth increments of 0.076, 0.1524, or 3.048 m (2.5, 5 or 10 ft).

Cone Penetrometer Test

The cone penetrometer test (CPT) has become common in recent years and rivals the SPT in popularity for geotechnical site investigations in some regions of the United States. It is performed by pushing a cone-tipped rod into the soil at a constant rate of 20 mm/s (ASTM 2012). The cone has an apex angle of 60 degrees and a cross-sectional area of 10 cm^2. As the cone progresses through the profile, the tip resistance (q_c) and sleeve resistance (f_s) are measured in units of stress. The sleeve is located directly behind the cone. The tip resistance and sleeve resistance can be correlated to soil type, and for soils that are determined to be liquefiable, the tip resistance can be used in many stress-based liquefaction evaluation procedures.

The CPT has some benefits over the SPT: it provides a nearly continuous profile and is better at revealing thin soil layers. Also, the CPT is more repeatable and less prone to operator error (Robertson and Cabal 2007). However, it does not return a soil sample, is still less common than the SPT, and is often not able to be used to characterize profiles having gravel layers or dense soil layers.

2.3 Seismic Compression

Research in the area of seismic compression of unsaturated sands began in earnest in the late 1960s and early 1970s. Early studies (Whitman 1969; Silver and Seed 1971; Youd 1972) established the relationship between the amplitude of the horizontal shear strain and the amount of seismic compression. Other studies recognized the effect of two dimensional motions (Pyke et al. 1975) and that fine-grained soils are less susceptible (Chu and Vucetic 1992) to large-magnitude volume changes. Several methods have been proposed to estimate the magnitude of seismic compression (Seed and Silver 1972; Tokimatsu and Seed 1987; Byrne 1991; Pradel 1998; Stewart and Whang 2003). Most of these procedures are "simplified" procedures that rely on number-of-equivalent-cycles correlations.

More recently, a great quantity of laboratory testing (Whang 2001; Hsu and Vucetic 2004; Stewart et al. 2004; Duku et al. 2006, 2008) has been performed to better understand soil compositional factors that affect the magnitude and accumulation of seismic compression and to improve the existing procedures (Duku et al. 2008; Yi 2010). In this section, a brief historical review of seismic compression research is given in chronological order.

2.3.1 Whitman and Ortigosa (1969)

Whitman and Ortigosa (1969) imposed vertical accelerations of various magnitudes on sand samples using a shaking table. They found that vertical accelerations alone did not cause appreciable settlement for accelerations less than about 1 g. Assuming it unlikely that vertical accelerations would exceed 1 g in an earthquake, they concluded that the effects of vertical accelerations could be neglected.

2.3.2 Silver and Seed (1971)

Silver and Seed (1971) performed strain-controlled cyclic simple shear tests using an NGI-type device. The tests were performed on dry crystal silica (#20) sand at target relative densities of 45, 60, and 80%. The cyclic shear strain amplitudes imposed on the samples ranged from 0.01 to 0.5%. From their test data they derived what would be known as a "volumetric strain

material model" (Stewart et al. 2004) for No. 20 crystal silica sand. They found that vertical strain increases with increasing shear strain amplitude, increasing number of strain cycles, and decreasing relative density of the sand. Vertical strain rate falls off exponentially as the number of cycles increases; most straining occurs in the first few cycles. Additionally, their tests showed that overburden stress had little effect on the total vertical strain. However, they discovered that the cyclic shear strain needed to produce a vertical strain in the sample (hereafter known as the threshold shear strain) increases with increasing overburden stress.

2.3.3 Youd (1972)

Youd (1972) performed strain-controlled tests on saturated and air-dried Ottawa sand with an NGI-type cyclic simple shear apparatus. The relative densities of the test specimens ranged from 70-80%. Youd's experiments confirmed Silver and Seed's (1971) results that vertical strain is a function of cyclic shear strain amplitude and that overburden stress does not affect vertical strain above a certain amplitude of cyclic shear strain. Additionally, Youd showed that there is not a significant difference in the behavior of dry sands and saturated sands under drained cyclic shear strain. Finally, Youd confirmed that no vertical strain occurs below a limiting or threshold shear strain.

2.3.4 Seed and Silver (1972)

Seed and Silver (1972) reported on a few sites that experienced seismic compression during the 1971 San Fernando earthquake, and proposed a semi-empirical method to estimate seismic compression. The method first requires a response analysis of the soil profile to estimate earthquake-induced shear strains. Next, representative samples from the profile are collected and subjected to the estimated vertical effective stress and an equivalent number of shear strain cycles (determined from the design earthquake magnitude). The vertical strain measured from testing each sample is integrated over the corresponding layer of the soil profile, yielding the total settlement. Seed and Silver (1972) validated their method by predicting settlements for shake table tests on dry sand. For these tests, their prediction error was less than

50%. They also provided a field case history where they predicted 6.35 cm (2.5 inches) of settlement for a profile similar to that of a site that had experienced settlements of 10 to 15 cm (4 to 6 inches). The method assumes vertically propagating shear waves in the soil profile.

2.3.5 Pyke et al. (1975)

Pyke et al. (1975) introduced the Jensen Filtration Plant case history in which a compacted fill settled up to 10 cm (4 inches). Pyke et al. (1975) used the procedure outlined by Seed and Silver (1972) but calculated a settlement of about one-third of the recorded settlement. They hypothesized that the uni-directional nature of the simple shear apparatus did not accurately reflect the multi-directional nature of earthquake motions. To study the issue, Pyke et al. performed stress-controlled tests using dry Monterey No. 0 sand on a combined pair of shaking tables whose motions were perpendicular.

Tests reported in Pyke et al. (1975) were performed with one-, two-, and three-directional random motions. The authors concluded that settlements from simultaneous perpendicular horizontal motions are nearly the same as the sum of the settlements of the motions acting separately. As a result, Pyke et al. (1975) recommended that computed settlements based on uni-directional tests should be doubled to more accurately represent field conditions. Additionally, contrary to the conclusions of Whitman and Ortigosa (1969), Pyke et al. concluded that vertical accelerations acting with horizontal accelerations may "cause a marked increase in settlements."

2.3.6 Tokimatsu and Seed (1987)

Tokimatsu and Seed (1987) provided an updated method to compute seismic compression; their seismic compression method is a simplified version of Seed and Silver's (1972) method. Where Seed and Silver require a site response analysis to be performed on the soil profile in order to estimate shear strains, Tokimatsu and Seed employ an equation similar to the equation for CSR (see Eqn. 2.3) in the simplified liquefaction evaluation procedure. Relationships between shear strain and volumetric strain are then used to calculate settlements (see Figure 2.3). Tokimatsu and Seed recommend multiplying calculated settlements by a factor of 2 per Pyke et al.'s (1975) suggestion.

This has been the customary practice since then. Tokimatsu and Seed (1987) represents a breakthrough for simple seismic compression estimations. However, it is a graphical procedure and was intended for clean sand only.

2.3.7 Byrne (1991)

Byrne (1991) proposed an alternative version of Martin et al.'s (1975) incremental shear-volume strain coupling equation for seismic compression or pore-water pressure increases. Byrne's equation can be used to estimate volumetric strains in a "non-simplified" manner. A "non-simplified" procedure in this context requires site response analyses and the resultant strain time history to estimate incremental and total volume changes. As such, it does not rely on a number of equivalent cycles correlation, but requires a numerical site response analysis to be performed. Byrne's model will be discussed in more depth in Chapter 3.

2.3.8 Chu and Vucetic (1992)

Chu and Vucetic (1992) performed strain-controlled cyclic simple shear tests (Marshall Silver-type device with NGI-type membranes) on brown, low plasticity clay (CL, $PI = 10.5$) at various moisture contents. Similar to the behavior of sands, they discovered that volumetric strains occur according to the amplitude of cyclic shear strain and the applied number of strain cycles. At small strains, like those considered in an earthquake, settlements are virtually independent of water content. Compared to sands, the threshold shear strain of the clay samples was an order of magnitude greater.

2.3.9 Pradel (1998)

In order to make Tokimatsu and Seed's (1987) procedure less tedious, Pradel (1998) developed equations that follow Tokimatsu and Seed's charts, enabling the procedure to be easily implemented in a spreadsheet or with other programming. Pradel's method shows good agreement with Tokimatsu and Seed's (1987) and Seed and Silver's (1972) methods for two case histories. However, the procedure is still only intended for use with clean sands and offers only a quick, rough estimate of seismic compression.

2.3.10 Stewart et al. (2001)

Stewart et al. (2001) document cases of permanent vertical and horizontal ground displacements in structural fills caused by the 1994 Northridge earthquake. While several different modes of failure may have occurred, Stewart et al. conclude that many of the displacements were caused by seismic compression in the compacted fills. Stewart et al. (2001) hoped to bring attention to the fact that seismic compression is a serious risk in engineered fills and cuts.

2.3.11 Whang (2001)

In an effort to better understand how compositional factors affect seismic compression, Whang (2001) performed bi-directional, strain-controlled cyclic simple shear tests on four different sands and seven different fill soils. The fill soils contained plastic and/or non-plastic fines.

In general, clean sands were found to undergo more volumetric strain than soils with fines. The clean sands with broad gradations and coarse particles showed about half the volumetric strain of sands with fine grain-particles and narrow gradations. Soils with non-plastic fines performed similarly to clean sands: seismic compression increased with decreasing relative density and was not affected by degree of saturation. The greatest volumetric strains in soils with plastic fines occurred at low compaction energy and low saturation. Whang hypothesized that this was caused by inter-clod void spaces that exist in pre-sheared plastic soils. Greater compaction energy and water content appeared to breakdown the proposed clod structure, thus limiting the seismic compression of the soil. Whang also proposed revisions to Tokimatsu and Seed's (1987) procedure. These suggestions are treated in Stewart and Whang (2003).

2.3.12 Stewart et al. (2002)

Stewart et al. (2002) performed a comprehensive case study on two occurrences of seismic compression from the 1994 Northridge earthquake. Both cases of settlement occurred on compacted fills; one experienced settlements of up to 18 cm, and the other site experienced settlements of less than 6

cm. The researchers endeavored to explain the recorded settlements using an updated version of Seed and Silver's (1972) procedure and to quantify the sensitivity of their results. To that end, they collected and tested many samples, performed a detailed site investigation, analyzed recorded ground motions, and performed site response analyses using 1-D and 2-D methods.

From the results of strain-controlled cyclic simple shear testing, Stewart et al. (2002) produced new material models for soils with varying amounts of fines and saturation levels. They found that soils with significant amounts of non-plastic fines behave in a similar manner as clean sands (e.g. as relative compaction increases, seismic compression decreases), but undergo less seismic compression. Similarly, for soils with significant amounts of low-plasticity fines ($PI \approx 15$), as relative compaction or saturation increases, the seismic compression decreases. At low saturation, volumetric strains are similar to those for clean sands at the same relative compaction, but at high saturation, the volumetric strains are one fifth to one half that of clean sands. These findings are in line with Whang's (2001) inter-clod void hypothesis.

Another important finding of Stewart et al. (2002) is that shear strains in the soil profile near slopes may be much greater than those obtained from a traditional 1-D analysis; a 2-D site response analysis usually provides better results.

2.3.13 Stewart and Whang (2003)

Striving to improve upon Tokimatsu and Seed (1987) and Pradel (1998), Stewart and Whang (2003) suggested four improvements:

1. The stress reduction factor (r_d) given by Tokimatsu and Seed is originally from Seed and Idriss (1971). Recent studies by Seed et al. (2001) show this factor to be generally high; new factors are suggested that incorporate the shear wave velocity in the top 12 meters of the soil.
2. Like many methods, Tokimatsu and Seed's procedure is based on 15 uniform strain cycles. Fifteen cycles is the assumed number of cycles from a magnitude 7.5 earthquake. For a magnitude different than a 7.5, Tokimatsu and Seed present a chart with correction factors for the volumetric strain. In Stewart and Whang's method, Liu et al.'s (2001) regression equation is used to estimate the number of cycles. In

this equation, the number of equivalent cycles is a function of earthquake magnitude, source-to-site distance, and site conditions. With the equivalent number of cycles from Liu et al.'s equation, a correction factor can be obtained from a *material model*.

3. Modulus reduction curves (shear modulus degradation curves) give the shear modulus as a function of shear strain and are an important part of Tokimatsu and Seed's method. Newer curves from Darendeli and Stokoe (2001) are offered to replace the older curves. Darendeli and Stokoe's curves are a function not only of shear strain, but also of plasticity index and effective overburden stress. Additionally, other shear modulus degradation curves have been published and may be more applicable.
4. An essential part of Tokimatsu and Seed's procedure is the volumetric strain material model from Silver and Seed (1971) for No. 20 crystal silica sand. Stewart and Whang (2003) suggest using a material model that incorporates the type of sand, the fines content, and the saturation level.

Stewart and Whang's (2003) suggestions aim to give better results from the same framework. With these suggestions, Tokimatsu and Seed's basic procedure is augmented to better estimate settlements in soils with fines and with varying levels of saturation.

2.3.14 Hsu and Vucetic (2004)

Hsu and Vucetic (2004) performed multistage, NGI-type direct simple shear tests on seven different soils. The plasticity index (PI) of the soils ranged from 0 to 34, and the saturation of the samples varied. Their results support earlier findings that no volumetric strain and virtually no change in the soil fabric occurs when the applied cyclic shear strain is less than the threshold shear strain, γ_{tv}. When the applied cyclic shear strains are above the threshold strain, permanent changes to the soil fabric occur, resulting in a volumetric strain. These authors found that the value of the threshold shear strain can be correlated with the plasticity index of the soil.

2.3.15 Whang et al. (2004)

Focusing on the performance of compacted fills, Whang et al. (2004) continued in the direction of Whang (2001) by performing strain-controlled cyclic simple shear tests on four different fill soils. All soils contained a significant amount of fines; some of the fines were non-plastic and others were of moderate plasticity. Many tests were performed on the soils at varying levels of saturation and relative compaction.

Altogether, Whang et al.'s (2004) results reiterate those of Whang (2001). In general, soils with fines experienced less settlement than clean sands. The effect of saturation was important for soils of medium-plasticity fines (greater volumetric strains occurred at lower saturations), but negligible for soils with non-plastic fines. The occurrence of clod formation in plastic soils enhanced the susceptibility of seismic compression. For construction of new fills, Whang et al. suggest minimizing clod formation by compacting plastic soils to greater than 90% modified proctor relative compaction (RC) and at wetter than optimum water contents.

2.3.16 Stewart et al. (2004)

Expanding on the work of Whang (2001) and Whang et al. (2004), Stewart et al. (2004) performed strain-controlled cyclic simple shear tests on 14 different clean sands, 8 silty sands, and one moderately plastic ($PI = 27$) clay. Results of testing did not show that the volumetric strain of clean sands is affected by the compositional factors of particle size (D_{50}), angularity, and gradation ($C_U = D_{60}/D_{10}$). For silty sands, increasing fines content resulted in greater seismic compression for a given relative compaction. This result is seemingly at odds with the conclusions of Whang et al. (2004). However, since Stewart et al. used non-plastic fines, they concluded that the addition of non-plastic fines to clean sand at the same relative compaction increases seismic compressions, but plastic fines tend to decrease seismic compression. The degree of saturation was found to make a significant difference in the amount of seismic compression in soils with fines. At 30% saturation, seismic compressions were lowest, with volumetric strains increasing as the saturation increased or decreased. It was hypothesized that matric suction in the soil increases the stiffness of the specimen, decreases the amount of compression, and is greatest at a saturation of 30%. The degree of saturation showed

little effect for clean sands. As mentioned, Stewart et al. (2004) tested one plastic clay ($FC = 77\%$ and $PI = 27$). Volumetric strains in the plastic clay were approximately one-half of those experienced by low plasticity clays ($PI = 15$). Otherwise, the clay performed similarly to low plasticity clays ($PI = 15$, see Whang (2001) and Whang et al. (2004)). Several new material models were produced that correlate volumetric strain to relative compaction, fines content, saturation, and plasticity index. Conveniently, the new material models are defined by variables in a standard equation, allowing easy use in spreadsheet functions.

2.3.17 Sawada et al. (2006)

Sawada et al. (2006) performed stress-controlled cyclic triaxial tests on Ohgishima sand at 50, 75, and 100 percent saturation. The sands contained a significant proportion of non-plastic fines ($FC = 21.8\%$), and were subjected, in the triaxial test, to irregular motions obtained from actual earthquake motions. Their results showed that for maximum shear strains less than 10%, total volumetric strains increased with decreasing saturation. At face value, this result runs counter to those of Stewart et al. (2004). However, the differences between settlements in the case of Sawada et al. were minimal, and Stewart et al. (2004) used soil mixtures that were 50% fines. Similar to other tests on clean sands, the final volumetric strain for soils with non-plastic fines was best indicated by the maximum shear strain, reiterating results from earlier investigations.

2.3.18 Duku et al. (2006)

Duku et al. (2006) studied the effect of aging on seismic compression using one non-plastic and three plastic unsaturated soils. An overburden stress was applied to the samples for different amounts of time (two minutes or two hours) before the samples were subjected to strain-controlled cyclic simple shear testing. For the non-plastic soil, aging made no significant difference in the amount of seismic compression that occurred. The plastic soil samples that experienced the two hours of applied overburden stress before testing demonstrated 50-70% less volumetric strain than the samples that experienced only two minutes. This finding may explain why few, if any, cases of

seismic compression have been recorded in plastic soils.

2.3.19 Duku et al. (2008)

Using strain-controlled cyclic simple shear tests, Duku et al. (2008) examined the effects of compositional and environmental factors on the seismic compression of sixteen clean sands. As expected, of all compositional and environmental factors, relative density had the greatest effect on the seismic compression of the sands. Other compositional factors (mineralogy, soil fabric, gradation, void ratio "breadth" ($e-e_{min}$), and particle angularity) showed little influence on the seismic compression susceptibility of the sands. Among the environmental factors, the degree of saturation and age did not influence the seismic compression of the sand. Contrary to the results of Silver and Seed (1971) and Youd (1972), Duku et al. (2008) found that overburden stress can have a significant effect with seismic compression decreasing as overburden increases. Additionally, the stress history of the sands was found to be an influencing environmental factor. It was found that sands with greater OCR experience less volumetric strain. However, Duku et al. concluded that the effects of stress history can be neglected during seismic compression analysis of clean sand fills because most fills become normally consolidated with depth. Besides examining the effects of compositional and environmental factors, Duku et al. (2008) produced several new material models to add to those produced by Stewart et al. (2004).

2.3.20 Yi (2010)

Based on the methods of Tokimatsu and Seed (1987) and Pradel (1998), Yi (2010) developed a shear wave velocity-based procedure to estimate seismic compression. Using a new relationship between relative density and shear wave velocity, Yi's method estimates volumetric strain from Tokimatsu and Seed (1987)-derived plots (see Figure 2.3).

2.4 Equivalent-Linear Site Response Analysis

The 1D equivalent-linear site response analysis first introduced by Schnabel et al. (1972) is a well-known and widely-used method to estimate the amplification (or de-amplification) of seismic waves as they propagate through a soil profile. In the years since it was first introduced, it has been shown to give a good approximation of real-world conditions. It assumes the vertical propagation of shear waves from a uniform half-space through the horizontal layers of a soil profile (Idriss and Sun 1992). These calculations are performed in the frequency domain. In order to model the non-linear response of soil, an iterative procedure is used.

For each layer in a soil profile, the equivalent-linear procedure begins by defining shear modulus reduction and damping curves for a range of expected shear strains. Figure 2.4 shows examples of these shear modulus reduction and damping curves for a range of overburden stresses. For the first iteration of the algorithm, the response of the soil profile is calculated using the initial conditions of shear modulus and damping. Generally these initial conditions are those corresponding to very small strains. From the shear strain time history response of each layer, a representative shear strain, γ_{eff}, is chosen from the strain time history of each layer (typically, $\gamma_{eff} = 0.65 \cdot \gamma_{max}$). This representative strain is used to pick values of shear modulus and damping using the aforementioned curves, and the process iterates until modulus and damping values reasonably agree between iterations.

Historically, the ratio of effective shear strain to maximum shear strain ($\gamma_{eff}/\gamma_{max}$) has taken a range of values in equivalent-linear site response analyses. For example, the SHAKE91 manual (Idriss and Sun 1992) defines the ratio of effective to maximum shear strain as:

$$\frac{\gamma_{eff}}{\gamma_{max}} = \frac{M-1}{10} \tag{2.2}$$

where M is the magnitude of the earthquake. However, the computed site response has been shown to not be particularly sensitive to this value (within a certain range), so it is generally taken as 0.65 (Kramer 1996).

Several software packages to perform equivalent-linear site response have

been developed over the years. SHAKE91, ShakeVT, SHAKE2000, and ProShake are all closely based on the original SHAKE code (Schnabel et al. 1972b). More recently, other equivalent-linear site response software have been developed, namely: DEEPSOIL (Hashash et al. 2010), Strata (Rathje and Kottke 2010), and ShakeVT2 (Lasley et al. 2014). For more information on the equivalent-linear procedure, in general, and the code written for this research, in particular, see Appendices I and J.

2.5 Stress-Based Simplified Liquefaction Evaluation Methods

The "simplified" liquefaction evaluation procedure was proposed separately by Whitman (1971) and Seed and Idriss (1971). This simplified procedure did not require site-specific response analysis, nor an earthquake time history to determine induced shear stresses (i.e. demand), hence the name "simplified" procedure. The core of the simplified procedure is the cyclic stress ratio (CSR):

$$CSR = \frac{\tau_{avg}}{\sigma'_{v0}} = 0.65 \frac{a_{max}}{g} \frac{\sigma_v}{\sigma'_{v0}} r_d \tag{2.3}$$

where τ_{avg} is the average amplitude of the seismically-induced shear stress, σ'_{v0} is the initial effective overburden stress, 0.65 is an arbitrary ratio of average shear stress to maximum shear stress (τ_{avg}/τ_{max}) of the earthquake motion, a_{max} is the peak ground acceleration at the surface of the profile, g is the acceleration due to gravity, σ_v is the total overburden stress, and r_d is the stress reduction factor to account for the non-rigid response of the soil column. By plotting CSR versus SPT blow count (or the results of any other in-situ test for soil state) for each liquefaction and non-liquefaction site in a case history database, a capacity (i.e. limit state) curve to divide liquefaction and non-liquefaction sites can be formed (see Figure 2.5). This curve defines the cyclic resistance ratio (CRR).

The cyclic resistance ratio is the cyclic stress ratio at which liquefaction occurs in a specific number of cycles (generally 15 cycles for a M_w7.5 earthquake) and is a function of the soil density. Traditionally, a factor of safety

against liquefaction is calculated as the ratio of cyclic resistance ratio and cyclic stress ratio (Eqn. 2.3).

$$FS = \frac{CRR}{CSR} \tag{2.4}$$

Over the years, the procedure has undergone a number of improvements and additions (e.g. Youd et al. 2001). The most notable of these changes are the additions of magnitude scaling factor (MSF), overburden correction factor (K_σ), and the correction for static shear stresses (K_α). The magnitude scaling factor corrects for the durations of earthquakes with magnitudes other than 7.5. The K_σ and K_α correction factors were needed because the cyclic stress ratio required to cause liquefaction changes with effective overburden stress and in the presence of an initial static shear stress (i.e. non-level ground conditions). Thus, the factor of safety now includes these corrections:

$$FS = \frac{CRR}{CSR} \cdot MSF \cdot K_\sigma \cdot K_\alpha \tag{2.5}$$

where CRR is the smallest cyclic stress ratio at which liquefaction occurs for a M_w7.5 earthquake and under one atmosphere of initial effective overburden stress. The cyclic resistance ratio is commonly correlated to SPT blow count corrected for fines content, but CRR curves that are correlated to CPT tip resistance or shear wave velocity are also common.

Following on the work of others (e.g. Liao et al. 1988; Youd and Noble 1997; Toprak et al. 1999), Cetin et al. (2004) developed a probabilistic, stress-based liquefaction evaluation procedure using SPT case histories. The probabilistic procedure allows for the incorporation of uncertainties (in both input parameters and the location of the CRR curve) into the analysis. Cetin et al. (2004) expanded the case history database and corrected or removed erroneous cases. Additionally, they developed a new stress reduction coefficient and incorporated fines content, effect of overburden stress, and duration weighting factors into the formulation of the CRR curve using Bayesian updating.

Idriss and Boulanger (2008) presented deterministic, stress-based simplified liquefaction evaluation procedures for the SPT and CPT. Their procedures retain the same general form of the Youd et al. (2001) procedure and provide updated relationships for r_d, MSF, K_σ, and the overburden correction factor

for SPT blow count, C_N. Building on that foundation and a reevaluation of the SPT case history database (Boulanger et al. 2012), Boulanger and Idriss (2012) presented a probabilistic update to the 2008 deterministic SPT procedure.

The energy-based liquefaction evaluation procedure presented herein uses the SPT case history database revised by Boulanger et al. (2012) and some of the probabilistic methods of Cetin et al. (2004) and Boulanger and Idriss (2012) (see Chapter 6).

2.6 Energy-Based Liquefaction Evaluation Methods

Dissipated energy per unit volume of soil is the cumulative area bound by stress-strain hysteresis loops. Figure 2.6 shows the stress-strain hysteresis loops from a stress-controlled cyclic simple shear test. Nemat-Nasser and Shokooh (1979) were the first to suggest the use of dissipated energy to estimate pore water pressure increases in saturated, undrained sands. They recognized that densification of drained sands or liquefaction in undrained sands involves the rearrangement of sand particles which, in turn, requires an expenditure of energy. Using dissipated energy, they accurately predicted pore pressure increases for the results of cyclic laboratory tests on undrained sands.

Following the pioneering work of Nemat-Nasser and Shokooh (1979), dissipated energy has been implemented as the damage metric in a number of liquefaction evaluation methods. Several of these methods are detailed below. They are primarily organized here by the manner in which the capacity (i.e. limit state) curve was developed. Presented first are the methods whose capacity curves are obtained from field case histories with in-situ measurements of soil properties. Next, procedures that define the capacity curves using the results of laboratory tests are presented. Not all of the liquefaction evaluation methods included in this section are based on dissipated energy; several are based on other measures of energy. They all aim to account for the damaging effects of the entire earthquake motion.

2.6.1 Field-Based Capacity Curves

The earliest liquefaction evaluation procedures used the Gutenburg-Richter (1956) relationship and some attenuation relationship to obtain the energy at the site of interest. Other procedures use site-response analyses or number of equivalent cycles correlations to estimate the energy. In all procedures described in this section, energy is determined for each case history in a database, and the capacity curve is estimated directly from this data without relying on laboratory results. For forward analysis of these procedures, the demand is obtained in the same way as the energy was determined for regression of the capacity curve. While there may be a discrepancy between the actual and estimated dissipated energies, the estimated demand energy of these procedures is consistent with the capacity curve energy. The most recent stress-based simplified procedures are developed in the same way; laboratory results have an effect on the development of the stress-based capacity curves (e.g. the overburden stress correction factor), but are not directly used for capacity curve regression.

Davis and Berrill (1982)

Beginning with Davis and Berrill (1982) several energy-based liquefaction methods have used the Gutenberg-Richter (1956) energy relationship to estimate the dissipated energy leading to liquefaction. The Gutenberg-Richter relationship is:

$$E_0 = 10^{1.5M+1.8} \tag{2.6}$$

where E_0 is the total radiated energy (kJ) from a magnitude M earthquake.

Davis and Berrill (1982) begin with the Gutenberg-Richter total radiated energy at the source and apply a geometric spreading attenuation to obtain the corresponding energy at the site. The portion of this reduced energy arriving at the site is dissipated as a function of the effective overburden stress and SPT blow count. Thus, the dissipated energy at the site and layer of interest is a function of magnitude, site-to-source distance, SPT blow count, and effective overburden stress. Using 57 liquefaction/no-liquefaction case histories, Davis and Berrill (1982) proposed a capacity curve that is a function of the corrected SPT blow count and uses the dissipated energy described

above as the damage metric. They were quick to point out some issues with their proposed approach, for example: 1. the model doesn't take into account directivity effects or other non-uniform energy radiation patterns; 2. site-to-source distances, in many cases, are difficult to establish with accuracy; 3. their SPT case history database was sparse and only contained results for fine sands. A more significant issue that Davis and Berrill (1982) did not mention is the application of a global measure of energy to a single layer of soil which is a crude approximation.

Berrill and Davis (1985)

Berrill and Davis (1985) aimed to improve upon the previous work. They retained the Gutenberg-Richter (1956) relationship but allowed for material attentuation as well as geometric spreading. In addition, they used a new dissipated energy-excess pore pressure model with a non-linear relationship (the Davis and Berrill 1982 method uses a linear relationship). Using 90 liquefaction/no-liquefaction case histories and the improved demand term, Berrill and Davis (1985) again obtained a capacity curve that is a function of corrected SPT blow count.

Law et al. (1990)

Following the general outline of Davis and Berrill (1982), Law et al. (1990) defined an energy-based demand term using the Gutenberg-Richter relationship that is a function of magnitude and distance and a capacity term that is a function of corrected SPT blow count. They fit their capacity curve using the results of 136 liquefaction/no-liquefaction case histories and provided separate curves for sand and silty sand. The form of their demand term allows for regime-specific attenuation by changing the value of B:

$$T = \frac{10^{1.5M}}{R^B} \tag{2.7}$$

where T is the demand term, M is the magnitude of the earthquake, R is the site-to-source distance, and B is the energy-attenuation parameter which they specify to be 4.3 ± 0.5 for the 'highly-fractured rock' of the western United States and Canada.

Trifunac (1995)

Based on the 90 case histories from Berrill and Davis (1985), Trifunac (1995) proposed five capacity curves that are a function of corrected SPT blow count and various other parameters. The first of these capacity curves uses the Gutenberg-Richter energy relationship and is simply a modification of the Davis and Berrill (1982) capacity curve. The second one of these obtains the demand energy term from the Fourier amplitude spectrum of the earthquake motion (via a regression model for the amplitude spectrum). The third Trifunac (1995) capacity curve estimates the demand energy at the site using the peak velocity and the duration, the fourth uses the Fourier amplitude of velocity at a period of 0.39 seconds, and the final uses the peak velocity, duration, and the Lamé constant μ. In all five cases, the demand energy was calculated for each case history and the capacity curve was defined as a function of SPT blow count.

Kayen and Mitchell (1997)

Kayen and Mitchell (1997) developed a procedure based on Arias intensity. Arias intensity is defined as (Arias 1970):

$$I_a = \frac{2\pi}{g}\left[\int a_x^2(t)dt + \int a_y^2(t)dt\right] \tag{2.8}$$

where a_x and a_y are horizontal and orthogonal components of an acceleration recording, and g is the acceleration due to gravity in the same units of acceleration. Kayen and Mitchell (1997) calculated the Arias intensity for several liquefaction/no-liquefaction case histories that had nearby ground motion recordings. After correcting the Arias intensity for depth in the profile and the SPT blow count for fines content, Kayen and Mitchell (1997) fit a capacity curve to the case history data. They also fit a curve for CPT tip resistance. For forward analyses, the Arias intensity demand can be calculated using recorded motions from similar magnitude-distance scenarios or estimated using a regressed relationship by Kayen (1993, unpublished PhD dissertation) which is a function of magnitude and site-to-source distance. Thus, the Kayen and Mitchell (1997) procedure either depends on site-specific earthquake time histories or an energy attenuation relationship.

Green (2001)

Green (2001) proposed a liquefaction evaluation procedure based on an SPT liquefaction/no-liquefaction case history database. The dissipated energy for each case history was estimated using the hysteretic definition of damping ratio, shear modulus reduction and damping curves, and the number of equivalent cycles concept. In Green's procedure, the normalized energy demand (NED) is estimated for a given earthquake scenario using inputs similar to those of the stress-based procedure:

$$NED = \frac{2\pi D_\gamma}{\sigma'_{m0} G_\gamma} \cdot \left[0.65 \frac{a_{max}}{g} \sigma_v r_d \right]^2 \cdot n_{eq} \tag{2.9}$$

where the squared quantity, $0.65 \cdot a_{max}/g \cdot \sigma_v \cdot r_d$, yields an approximation of the average shear stress, τ_{avg}, in the same way as the stress-based simplified procedures (Eqn. 2.3). The variable n_{eq} is the number of equivalent cycles (the basis for the stress-based MSF). The values of D_γ and G_γ are obtained from shear modulus reduction and damping curves and correspond to the shear strain developed by τ_{avg}. Since this general form is used in the procedure proposed herein, the method of obtaining D_γ and G_γ will be explained in Chapter 6. Green's (2001) normalized energy capacity term, NEC, is defined as:

$$NEC = 1.195 \cdot 10^{-4} \exp(0.185 \cdot N_{1,60cs}) \tag{2.10}$$

where $N_{1,60cs}$ is the SPT blow count corrected for overburden stress, energy, and fines content, among other things. Both NED and NEC are terms of dissipated energy normalized by the mean initial effective confining stress.

Green (2001) also showed that the dissipated energy from laboratory testing cannot be directly related to the dissipated energy from equivalent-linear site response analyses. The reason is that equivalent-linear site response analyses assume a constant value of shear modulus and damping for the entire loading time history, neglecting the additional dissipated energy that results from the softening of the soil at high excess pore pressures. In order to compare laboratory dissipated energies to equivalent-linear dissipated energies, Green proposed a correction factor be applied to the laboratory energies.

Mayfield (2007)

Mayfield (2007) proposed a liquefaction evaluation procedure based on CAV_5, the cumulative absolute velocity with a threshold of 5 cm/s. It is defined as (Kramer and Mitchell 2006):

$$CAV_5 = \int \chi \cdot |a(t)| \; dt \tag{2.11}$$

where

$$\chi = \begin{cases} 0 & \text{if } |a| < 5 \; cm/s^2 \\ 1 & \text{if } |a| \geq 5 \; cm/s^2 \end{cases} \tag{2.12}$$

and a is an acceleration time history. For a liquefaction/no-liquefaction case history database, Mayfield calculated CAV_5 from nearby ground motions, where available, or using an attenuation relationship developed by Kramer and Mitchell (2006). The resulting capacity curve is a function of both corrected SPT blow count and CAV_5. For forward analysis, the Mayfield procedure relies on the availability of earthquake recordings or the CAV_5 attenuation relationship.

Jafarian et al. (2014)

Jafarian et al. (2014) presented a liquefaction capacity curve based on cumulative kinetic energy density (CKED). For a given situation, CKED is a function of the effective density of the soil and the specific energy density (SED) of the incoming motion. SED, in turn, is a defined as:

$$\text{SED} = \int v^2 \; dt \tag{2.13}$$

where v is the velocity time history at the surface of the profile. Jafarian et al. (2014) developed a regression equation for SED using 1436 acceleration time histories from 60 earthquakes and the soil profiles of Cetin (2000). Their proposed regression equation for SED is a function of magnitude, site-to-source distance, fault type, and the mean shear wave velocity of the upper

30 meters of the profile. Using this regression equation and liquefaction/no-liquefaction case history databases, Jafarian et al. developed their liquefaction capacity curve which is a function of CKED, effective overburden stress, corrected SPT blow count, and fines content.

In many ways, the Jafarian et al. (2014) procedure is similar to the procedure proposed herein (Chapter 6): equivalent-linear site response analyses are used to develop an equation to predict dissipated energy in a more simplified manner (i.e. a number of equivalent cycles correlation is developed in Chapter 4).

2.6.2 Laboratory-Based Capacity Curves

Several energy-based capacity curves have been developed from the results of laboratory testing. In most cases, the resulting capacity curve of the lab test is converted for use with an in-situ measurement such as SPT blow count. Despite the conversion to field test results, the original curve is generally regressed from the lab results without input from case history data. For forward analyses, this makes the demand energy difficult to determine in a manner consistent with the capacity energy.

Case Western

A large amount of research in energy-based liquefaction evaluation has been performed by researchers at Case Western Reserve University. This section discusses their work.

Based on the results of strain-controlled hollow cylinder torsional shear tests using sinusoidal loadings, Figueroa et al. (1994) provided an equation for the dissipated energy to liquefaction which is a function of effective confining stress and relative density of the sand. Their tests were performed on specimens with relative densities of 50, 60, and 70% and at effective confining stresses of 41, 83, and 124 kPa.

Similar to the work of Figueroa et al. (1994), Liang et al. (1995) performed hollow cylinder torsional shear tests with specimens at the same relative densities and under the same effective confining stresses, but these tests

used stress-controlled random earthquake-type loadings instead of strain-controlled sinusoidal loadings. They also provided an equation for the dissipated energy to liquefaction using the same functional form as Figueroa et al. (1994). They concluded that the differences between the regression coefficients for random loading and for sinusoidal loading were not statistically significant; in other words, the regressions can be considered to be the same. Both Figueroa et al. (1994) and Liang et al. (1995) suggest the use of their regressed equations in connection with forward analysis of liquefaction potential.

The proposed liquefaction evaluation procedure base on the Figueroa et al. (1994) and Liang et al. (1995) lab results is given in more detail in Liang (1995). The procedure requires torsional shear tests performed on recovered soil samples from the site of interest to determine the capacity dissipated energy. The demand is determined using site response analyses with a characteristic earthquake input motion. In the site response analyses, a nonlinear, lumped-mass model is used to account for soil softening as loading progresses and pore water pressures increase.

Davis and Berrill (1996)

Davis and Berrill (1996) use the results of equivalent-linear site response analyses to evaluate liquefaction potential. From the site response analyses, the frequency-domain displacement at the surface of the profile is used to calculate the dissipated energy demand. This dissipated energy is then used to estimate the excess pore water pressures using correlations obtained from laboratory data (Simcock et al. 1983). If the excess pore water pressures exceed the initial effective stress, liquefaction is considered to have occurred in the given layer. Unlike many of the other procedures, the Davis and Berrill (1996) procedure does not provide a limit state curve, but relies on an excess pore pressure model instead.

Jafarian et al. (2012)

Jafarian et al. (2012) performed a number of strain-controlled hollow cylinder cyclic torsional shear tests on Toyoura sand specimens. The specimens had relative densities ranging from 30 to 75% and experienced effective confining stresses ranging from 55 to 166 kPa. For each test, they calculated the

dissipated energy to liquefaction normalized by the initial effective vertical stress, and from these results they proposed a capacity curve that is a function of the at-rest earth pressure coefficient (K_0) and the overburden- and energy-corrected SPT blow count (via the relative density of the specimens). They verified their capacity curve using dissipated energies calculated from stress and strain time histories from centrifuge testing, shaking table testing, and downhole array data from the Superstition Hills earthquake. Thus, for forward analysis of liquefaction potential, non-linear site response analyses are needed.

Kokusho and Mimori (2015)

Kokusho and Mimori (Kokusho 2013; Kokusho and Mimori 2015) developed an energy-based procedure with a capacity curve derived from the results of stress-controlled cyclic triaxial tests. They performed these tests on Futtsu sand specimens with relative densities ranging from 30 to 70% and fines contents ranging from 0 to 20%. From the results they correlated the normalized dissipated energy to liquefaction (normalized by the effective confining stress) to the cyclic stress ratio required to cause liquefaction in 20 cycles. This correlation they combined with a CSR-based capacity curve to obtain a capacity energy curve that is a function of SPT blow counts. For forward analyses, the capacity energy is compared with demand energy obtained from a velocity time history of an equivalent-linear site response analyses.

2.6.3 Summary of Energy-based Procedures

Many energy-based liquefaction evaluation procedures have been proposed. The field-based procedures depend on earthquake motion time histories or attenuation relationships. The laboratory-based procedures either require laboratory testing using soil from the site or compare the dissipated energy from site response analyses with the dissipated energy required to cause liquefaction in laboratory tests. For engineers accustomed to the existing stress-based simplified procedures, the energy-based liquefaction evaluation procedures would be foreign, difficult to implement, and require more resources to complete. Additionally, neither the existing energy-based nor the stress-based procedures are easy to implement for tectonic regimes like the stable-continental regime of the central-eastern United States.

2.7 References

Arias, A. (1970). "A measure of earthquake intensity." R. J. Hansen, ed. *Seismic design for nuclear power plants*, MIT Press, Cambridge, Mass.

ASTM. (2007). "Standard test method for consollidated undrained direct simple shear testing of cohesive soils." Designation: D6528-07, ASTM International, West Conshohocken, PA.

ASTM. (2011). "Standard test method for standard penetration test (SPT) and split-barrel sampling of soils." Designation: D1586-11, ASTM International, West Conshohocken, PA.

ASTM. (2012). "Standard test method for electronic friction cone and piezocone penetration testing of soils." Designation: D5778-12, ASTM International, West Conshohocken, PA.

ASTM. (2013). "Standard test method for load controlled cyclic triaxial strength of soil." Designation: D5311/D5311M-13, ASTM International, West Conshohocken, PA.

Berrill, J. B., and Davis, R. O. (1985). "Energy dissipation and seismic liquefaction of sands: Revised model." *Soils and Foundations*, 25(2), 106–118.

Boulanger, R., and Idriss, I. (2012). "Probabilistic Standard Penetration Test–Based Liquefaction–Triggering Procedure." *Journal of Geotechnical and Geoenvironmental Engineering*, 138(10), 1185–1195.

Boulanger, R., Wilson, D., and Idriss, I. (2012). "Examination and Reevalaution of SPT-Based Liquefaction Triggering Case Histories." Journal of Geotechnical and Geoenvironmental Engineering, 138(8), 898–909.

Byrne, P. (1991). "A cyclic shear-volume coupling and pore pressure model for sand." *Proceedings of the Second International Conference on Recent Advances in Geotechnical Earthquake Engineering and Soil Dynamics*, St. Louis, MO, March, Paper, 47–55.

Cetin, K. Ö. (2000). "Reliability-based assessment of seismic soil liquefaction initiation hazard." Ph.D. Dissertation, University of California, Berkeley, Berkeley, CA.

Cetin, K. O., Seed, R. B., Der Kiureghian, A., Tokimatsu, K., Harder, L. F., Kayen, R. E., and Moss, R. E. S. (2004). "Standard Penetration Test-

Based Probabilistic and Deterministic Assessment of Seismic Soil Liquefaction Potential." *Journal of Geotechnical and Geoenvironmental Engineering*, 130(12), 1314–1340.

Chu, H.-H., and Vucetic, M. (1992). "Settlement of compacted clay in a cyclic direct simple shear device." *ASTM Geotechnical Testing Journal*, 15(4), 371–379.

Darendeli, M. B., and Stokoe, K. H. (2001). Development of a new family of normalized modulus reduction and material damping curves. Geotechnical Engineering Report GD01-1, University of Texas at Austin.

Davis, R. O., and Berrill, J. B. (1982). "Energy dissipation and seismic liquefaction in sands." *Earthquake Engineering & Structural Dynamics*, 10(1), 59–68.

Davis, R. O., and Berrill, J. B. (1996). "Liquefaction susceptibility based on dissipated energy: a consistent design methodology." *Bulletin of the New Zealand National Society for Earthquake Engineering*, 29(2), 83–91.

Duku, P. M., Stewart, J. P., and Whang, D. H. (2006). "Effect of post-compaction ageing on seismic compression of fine-grained soils." *Geotechnical Special Publication 152: Ground Modification and Seismic Mitigation*, ASCE, 152, 411.

Duku, P. M., Stewart, J. P., Whang, D. H., and Yee, E. (2008). "Volumetric strains of clean sands subject to cyclic loads." *Journal of Geotechnical and Geoenvironmental Engineering*, 134(8), 1073–1085.

Dyvik, R., Berre, T., Lacasse, S., and Raadim, B. (1987). "Comparison of truly undrained and constant volume direct simple shear tests." *Geotechnique*, 37(1), 3–10.

Figueroa, J. L., Saada, A. S., Liang, L., and Dahisaria, N. M. (1994). "Evaluation of soil liquefaction by energy principles." *Journal of Geotechnical Engineering*, 120(9), 1554–1569.

Finn, W. D. L., and Vaid, Y. P. (1977). "Liquefaction potential from drained constant volume cyclic simple shear tests." *Proceedings of the 6th World Conference on Earthquake Engineering*, New Delhi, India, 10–14.

Finn, W. D., Vaid, Y. P., and Bhatia, S. K. (1979). "Constant volume cyclic simple shear testing." *Proceedings of the Second International Conference*

on Microzonation for Safer Construction - Research and Application, 839 – 851.

Green, R. A. (2001). "Energy-Based Evaluation and Remediation of Liquefiable Soils." Ph.D., Virginia Polytechnic Institute and State University, Blacksburg, Virginia.

Gutenberg, B. and Richter, C. F. (1956). "Magnitude and energy of earthquakes." *Ann Geofis.*, 9, 1-15.

Hashash, Y.M.A, Groholski, D.R., Phillips, C. A., Park, D, Musgrove, M. (2011) "DEEPSOIL 5.0, User Manual and Tutorial." 107 p.

Holtz, R. D., Kovacs, W. D., and Sheahan, T. C. (2010). *An Introduction to Geotechnical Engineering*. Prentice Hall, Upper Saddle River, NJ.

Hsu, C.-C., and Vucetic, M. (2004). "Volumetric threshold shear strain for cyclic settlement." *Journal of Geotechnical and Geoenvironmental Engineering*, 130(1), 58–70.

Idriss, I. M., and Boulanger, R. W. (2008). *Soil Liquefaction During Earthquakes*. Earthquake Engineering Research Institute, Oakland, California.

Idriss, I. M., and Sun, J. I. (1992). *User's Manual for SHAKE91: A Computer Program for Conducting Equivalent Linear Seismic Response Analyses of Horizontally Layered Soil Deposits*. Center for Geotechnical Modeling, Department of Civil & Environmental Engineering, University of California.

Jafarian, Y., Towhata, I., Baziar, M. H., Noorzad, A., and Bahmanpour, A. (2012). "Strain energy based evaluation of liquefaction and residual pore water pressure in sands using cyclic torsional shear experiments." *Soil Dynamics and Earthquake Engineering*, 35, 13–28.

Jafarian, Y., Vakili, R., Abdollahi, A. S., and Baziar, M. H. (2014). "Simplified soil liquefaction assessment based on cumulative kinetic energy density: Attenuation law and probabilistic analysis." *International Journal of Geomechanics*, 14(2), 267–281.

Kayen, R., and Mitchell, J. (1997). "Assessment of liquefaction potential during earthquakes by arias intensity." *Journal of Geotechnical and Geoenvironmental Engineering*, 123(12), 1162–1174.

Kokusho, T. (2013). "Liquefaction potential evaluations: energy-based method versus stress-based method." *Canadian Geotechnical Journal*, 50(10), 1088–1099.

Kokusho, T., and Mimori, Y. (2015). "Liquefaction potential evaluations by energy-based method and stress-based method for various ground motions." *Soil Dynamics and Earthquake Engineering*, 75, 130–146.

Kramer, S. L. (1996). *Geotechnical Earthquake Engineering*. Prentice Hall, New Jersey.

Kramer, S. L., and Mitchell, R. A. (2006). "Ground Motion Intensity Measures for Liquefaction Hazard Evaluation." *Earthquake Spectra*, 22(2), 413–438.

Kuerbis, R., and Vaid, Y. P. (1988). "Sand sample preparation-the slurry deposition method." *Soils and Foundations*, 28(4), 107–118.

Ladd, R. S. (1974). "Specimen preparation and liquefaction of sands." *Journal of Geotechnical and Geoenvironmental Engineering*, 100(GT10), Proc. Paper 10857, 1180-1184.

Ladd, R. S. (1978). "Preparing test specimens using undercompaction." *Geotechnical Testing Journal*, ASTM 1(1), 16–23.

Lasley, S. J., Green, R. A., and Rodriguez-Marek, A. (2014). "Comparison of equivalent-linear site response analysis software." *Proceedings of the 10th U.S. National Conference on Earthquake Engineering*, Earthquake Engineering Research Institute, Anchorage, AK.

Law, K. T., Cao, Y. L., and He, G. N. (1990). "An energy approach for assessing seismic liquefaction potential." *Canadian Geotechnical Journal*, 27(3), 320–329.

Liang, L. (1995). "Development of an energy method for evaluating the liquefaction potential of a soil deposit." Ph.D. Dissertation, Case Western Reserve University.

Liang, L., Figueroa, J. L., and Saada, A. S. (1995). "Liquefaction under random loading: Unit energy approach." *Journal of Geotechnical Engineering*, 121(11), 776–781.

Liao, S. S. C., Veneziano, D., and Whitman, R. V. (1988). "Regression models for evaluating liquefaction probability." *Journal of Geotechnical Engineering*, 114(4), 389–411.

Liu, A. H., Stewart, J. P., Abrahamson, N. A., and Moriwaki, Y. (2001). "Equivalent number of uniform stress cycles for soil liquefaction analysis."

Journal of Geotechnical and Geoenvironmental Engineering, 127(12), 1017–1026.

Martin, G.R., Finn, W.D.L., and H.B. Seed (1975). "Fundamentals of liquefaction under cyclic loading." *Journal of the Geotechnical Engineering Division*, ASCE, 101(GT5): 423-438.

Mayfield, R. T. (2007). "The Return Period of Soil Liquefaction." Ph.D. Dissertation, University of Washington, Department of Civil and Environmental Engineering.

Mulilis, J. P., Arulanandan, K., Mitchell, J. K., Chan, C. K., and Seed, H. B. (1977). "Effects of sample preparation on sand liquefaction." *Journal of the Geotechnical Engineering Division*, 103(2), 91–108.

Nemat-Nasser, S., and Shokooh, A. (1979). "A unified approach to densification and liquefaction of cohesionless sand in cyclic shearing." *Canadian Geotechnical Journal*, 16(4), 659–678.

Pradel, D. (1998). "Procedure to evaluate earthquake-induced settlements in dry sandy soils." *Journal of Geotechnical and Geoenvironmental Engineering*, 124(4), 364–368.

Pyke, R. M., Chan, C. K., and Seed, H. B. (1975). "Settlement of sands under multidirectional shaking." *Journal of the Geotechnical Engineering Division*, 101(4), 379–398.

Rathje, E. M., and Kottke, A. (2010). "Strata." http://nees.org/resources/strata.

Robertson, P.K. and Cabal K.L. (2007). "Guide to Cone Penetration Testing for Geotechnical Engineering." 2nd Ed., Gregg Drilling & Testing, Inc., Signal Hill, CA.

Sawada, S., Tsukamoto, Y., and Ishihara, K. (2006). "Residual deformation characteristics of partially saturated sandy soils subjected to seismic excitation." *Soil Dynamics and Earthquake Engineering*, 26(2-4), 175–182.

Schnabel, P., Seed, H. B., and Lysmer, J. (1972). "Modification of seismograph records for effects of local soil conditions." *Bulletin of the Seismological Society of America*, 62(6), 1649–1664.

Schnabel, P.B., Lysmer, J., and Seed, H.B. (1972b). "SHAKE: a computer program for earthquake response analysis of horizontally layered sites," Report EERC 72-12, Earthquake Engineering Research Center, University of California, Berkeley.

Seed, H. B., and Idriss, I. M. (1971). "Simplified procedure for evaluating soil liquefaction potential." *Journal of the Soil Mechanics and Foundations Division*, ASCE, 97(9), 1249–1273.

Seed, H. B., and Silver, M. L. (1972). "Settlement of dry sands during earthquakes." *Journal of Soil Mechanics & Foundations Div*, 98(SM4), 381–397.

Seed, H. B., Arango, I., and Chan, C. K. (1975). "Evaluation of soil liquefaction potential during earthquakes, EERC 75-28." Earthquake Engineering Research Center, University of California, Berkeley.

Seed, H. B., Tokimatsu, K., Harder, L. F., and Chung, R. M. (1985). "Influence of SPT procedures in soil liquefaction resistance evaluations." *Journal of Geotechnical Engineering*, 111(12), 1425–1445.

Seed, R.B., Cetin, K.O., Moss, R.E.S., Kammerer, A.M., Wu, J., Pestana, J.M., and Riemer, M.F. (2001). "Recent advances in soil liquefaction engineering and seismic site response evaluation," *Proc. 4th Int. Conf. on Recent Advances in Geotech. Eqk. Engrg. Soil Dyn.*, Paper No. SPL-2.

Silver, M. L., and Seed, H. B. (1971). "Volume changes in sands during cyclic loading." *Journal of Soil Mechanics & Foundations Div*, 97(SM9), 1171–1182.

Sivathayalan, S., and Ha, D. (2011). "Effect of static shear stress on the cyclic resistance of sands in simple shear loading." *Canadian Geotechnical Journal*, 48(10), 1471–1484.

Stafford, P. J., Berrill, J. B., and Pettinga, J. R. (2009). "New predictive equations for Arias intensity from crustal earthquakes in New Zealand." *Journal of Seismology*, 13(1), 31–52.

Stewart, J. P., Bray, J. D., McMahon, D. J., Smith, P. M., and Kropp, A. L. (2001). "Seismic performance of hillside fills." *Journal of Geotechnical and Geoenvironmental Engineering*, 127(11), 905–919.

Stewart, J. P., Smith, P. M., Whang, D. H., and Bray, J. D. (2002). "Documentation and analysis of field case histories of seismic compression during the 1994 Northridge, California earthquake." Pacific Earthquake Engineering Research Center, University of California, Berkeley.

Stewart, J. P., and Whang, D. H. (2003). "Simplified procedure to estimate ground settlement from seismic compression in compacted soils." *Proceed-*

ings 2003 Pacific Conference on Earthquake Engineering, Christchurch, New Zealand, Paper.

Stewart, J. P., Whang, D. H., Moyneur, M., and Duku, P. (2004). "Seismic compression of as-compacted fill soils with variable levels of fines content and fines plasticity." Consortium of Universities for Research in Earthquake Engineering, Richmond, CA.

Tokimatsu, K., and Seed, H. B. (1987). "Evaluation of settlements in sands due to earthquake shaking." *Journal of Geotechnical Engineering*, 113(8), 861–878.

Toprak, S., Holzer, T. L., Bennett, M. J., and Tinsley, J. C. (1999). "CPT- and SPT-based probabilistic assessment of liquefaction potential." *Proc., 7th U.S.–Japan Workshop on Earthquake Resistant Design of Lifeline Facilities and Countermeasures Against Liquefaction*, Seattle.

Travasarou, T., J. D. Bray, and N. A. Abrahamson (2003). "Empirical attenuation relationship for Arias Intensity," *Earthquake Engineering and Structural Dynamics*, 32, 1133-1155.

Trifunac, M. D. (1995). "Empirical criteria for liquefaction in sands via standard penetration tests and seismic wave energy." *Soil Dynamics and Earthquake Engineering*, 14(6), 419–426.

Vaid, Y. P., and Negussey, D. (1984). "Relative density of pluviated sand samples." *Soils and Foundations*, 24(2), 101–105.

Vaid, Y. P., and Sivathayalan, S. (1996). "Static and cyclic liquefaction potential of Fraser Delta sand in simple shear and triaxial tests." *Canadian Geotechnical Journal*, 33(2), 281–289.

Vaid, Y. P., Sivathayalan, S., and Stedman, D. (1999). "Influence of specimen-reconstituting method on the undrained response of sand." *ASTM Geotechnical Testing Journal*, 22(3), 187–195.

Whang, D. H. (2001). "Seismic compression of compacted fills." Ph.D. Thesis, University of California, Los Angeles.

Whang, D. H., Stewart, J. P., and Bray, J. D. (2004). "Effect of compaction conditions on the seismic compression of compacted fill soils." ASTM.

Whitman, R. V., and Ortigosa, P.O. (1969). "Densification of sand by vertical vibrations." *Proceedings of the 4th World Conference on Earthquake Engineering*, Santiago, Chile, 29–37.

Whitman, Robert V. (1971). "Resistance of soil to liquefaction and settlement." *Soils and Foundations* 11(4), 59–68.

Wu, J., Kammerer, A. M., Riemer, M. F., Seed, R. B., and Pestana, J. M. (2004). "Laboratory study of liquefaction triggering criteria." *Proc. of the 13th World Conference on Earthquake Engineering*, Vancouver, BC, Canada, Paper.

Yi, F. (2010). "Procedure to evaluate seismic settlement in dry sand based on shear wave velocity." *Proceedings of the 9th U.S. National and 10th Canadian Conference on Earthquake Engineering (9USN/10CCEE)*, Toronto, Canada. Paper No. 760.

Youd, T. L. (1972). "Compaction of Sands by Repeated Shear Straining." *Journal of the Soil Mechanics and Foundations Division*, 98(7), 709–725.

Youd, T. L., Idriss, I. M., Andrus, R. D., Arango, I., Castro, G., Christian, J. T., Dobry, R., Finn, W. D. L., Harder, L. F., Hynes, M. E., Ishihara, K., Koester, J. P., Liao, S. S. C., Marcuson, W. F., Martin, G. R., Mitchell, J. K., Moriwaki, Y., Power, M. S., Robertson, P. K., Seed, R. B., and Stokoe, K. H. (2001). "Liquefaction resistance of soils: Summary report from the 1996 NCEER and 1998 NCEER/NSF workshops on evaluation of liquefaction resistance of soils." *Journal of Geotechnical and Geoenvironmental Engineering*, ASCE, 127(10), 817–833.

Youd, T. L., and Noble, S. K. (1997). "Liquefaction criteria based on statistical and probabilistic analyses." *Proc., NCEER Workshop on Evaluation of Liquefaction Resistance of Soils*, NCEER Technical Rep. No: NCEER-97-0022, 201–205.

2.8 Figures

Figure 2.1: Triaxial specimen of Monterey 0/30 sand with vacuum applied.

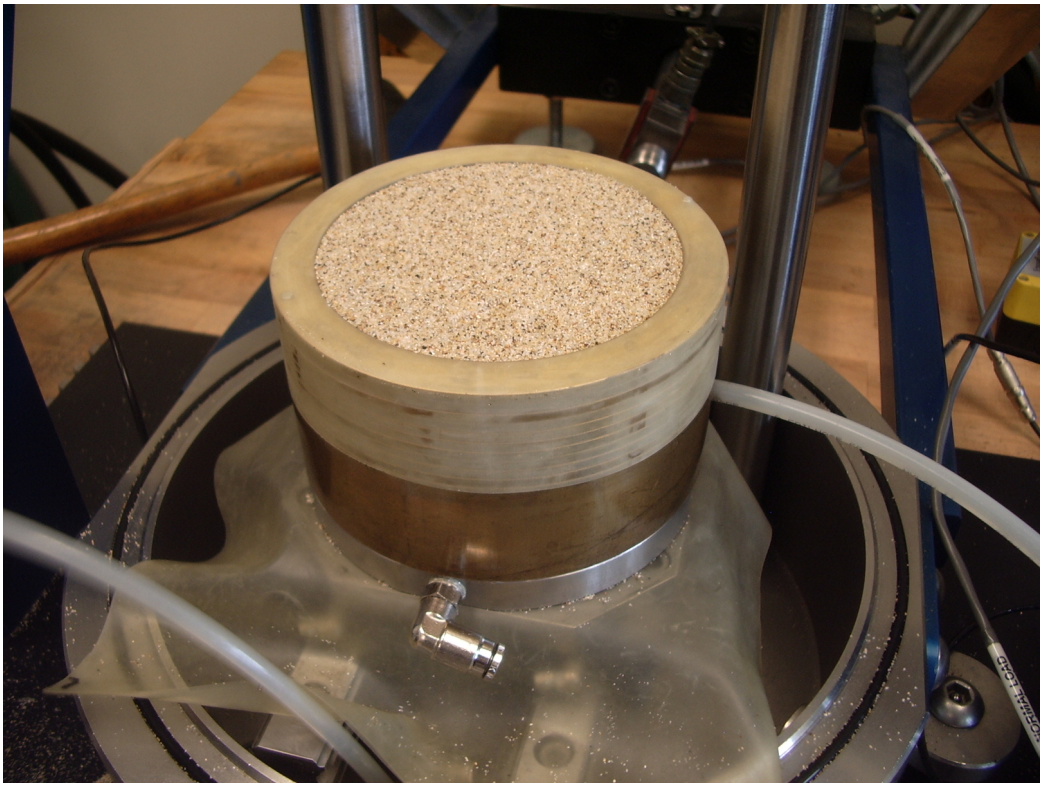


Figure 2.2: A cyclic simple shear specimen in preparation.

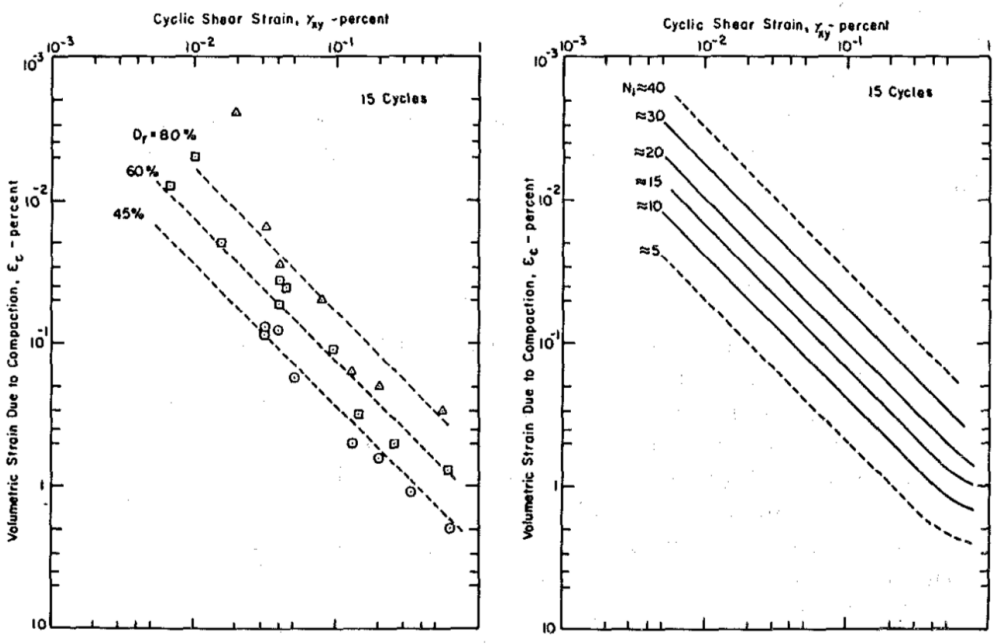


Figure 2.3: Volumetric strain as a function of shear strain and a) relative density (D_r), or b) SPT blow count. (Tokimatsu and Seed, 1987) [Fair use.]

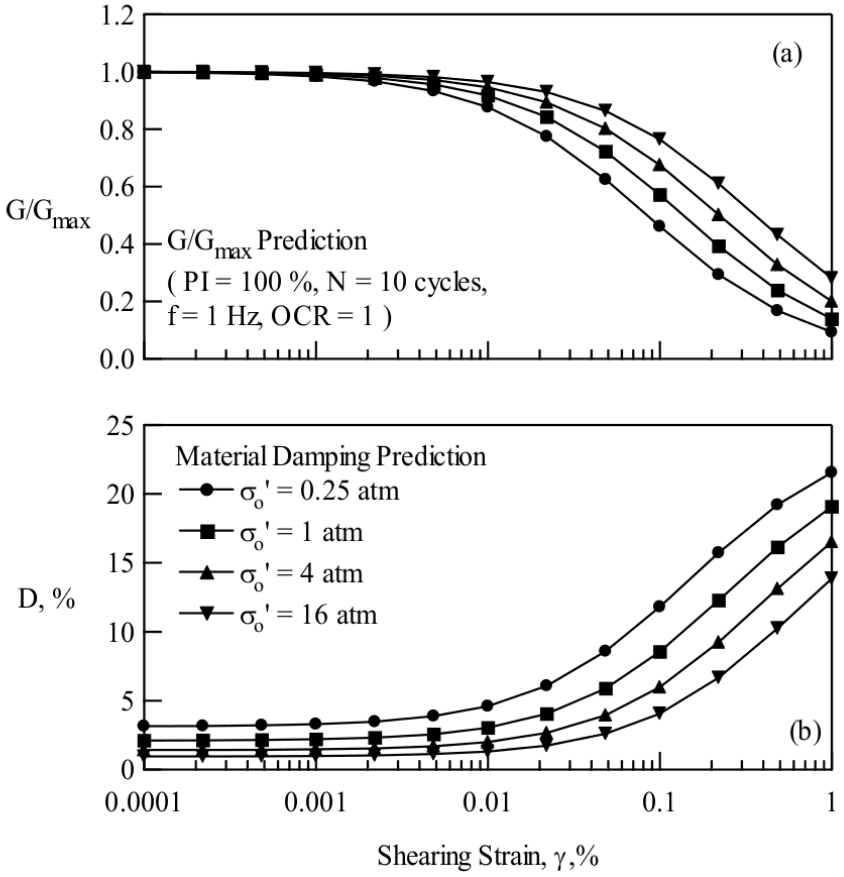


Figure 2.4: Shear modulus and damping degradation as a function of strain and overburden stress (Darendeli and Stokoe 2001). [Fair use.]

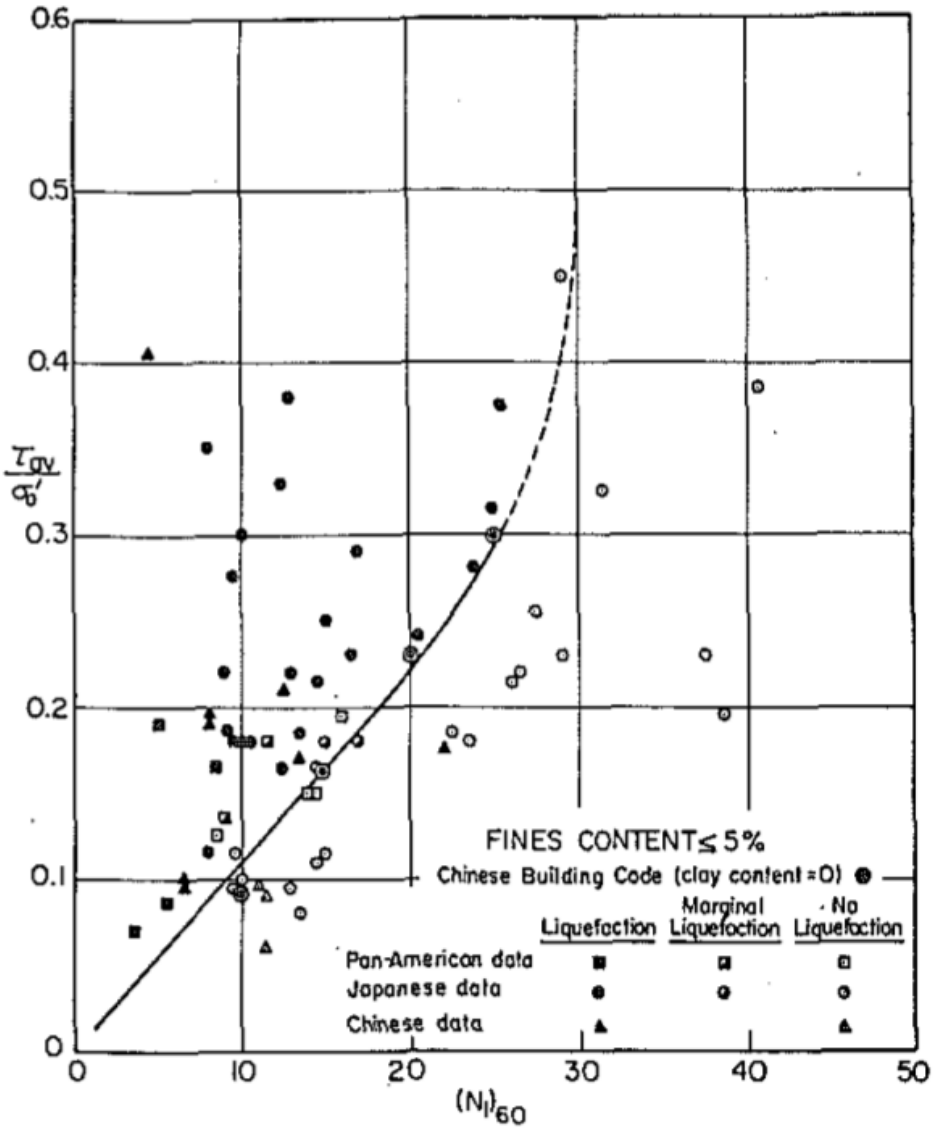


Figure 2.5: Cyclic stress ratio versus corrected SPT blow counts for liquefaction and non-liquefaction case histories for $M7.5$ earthquakes. (Seed et al. 1985) [Fair use.]

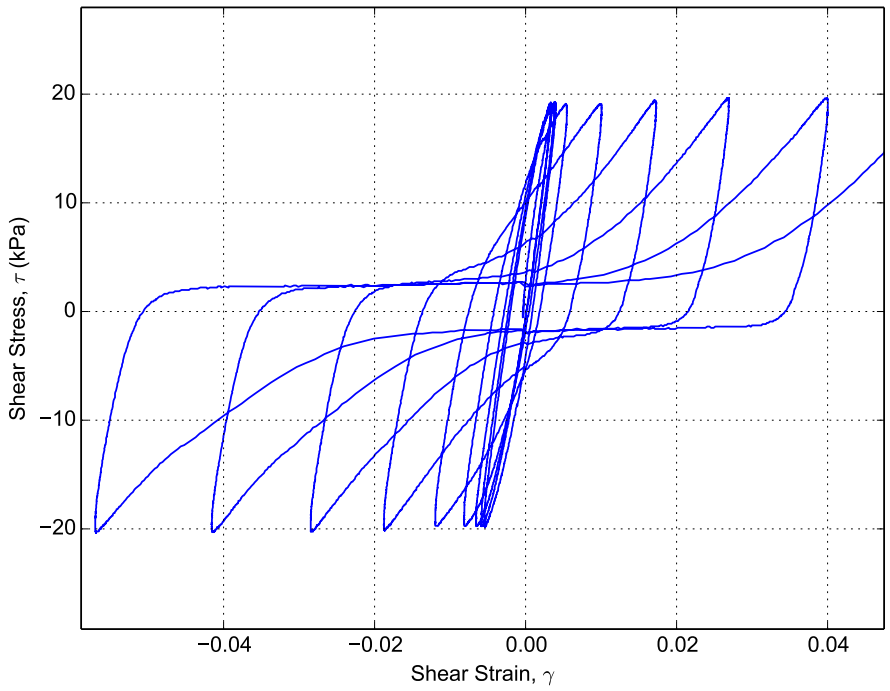


Figure 2.6: Stress-strain hysteresis loops of a stress-controlled cyclic simple shear test.

Chapter 3

Manuscript #1: An Approach for Estimating Seismic Compression Using Site Response Analyses

The following manuscript has been submitted to ASCE's *Journal of Geotechnical and Geoenvironmental Engineering*.

Samuel Lasley made the following contributions:

- With Chen, established a testing procedure for the GCTS equipment and trained Chen on its use
- Performed the index testing of the sand with Chen (maximum and minimum void ratio tests, sieve analyses)
- Reduced all laboratory testing data
- Examined various functional forms for r, the exponential of the Richart-Newmark method, as well as alternate volumetric strain functional forms.
- Selected the best functional forms for r and N; performed the regressions for both
- Used the Richart-Newmark and Byrne methods to estimate the volumetric strain under irregular strain loadings
- Performed all statistical analyses and prepared all figures and tables

- Wrote the first draft of the manuscript and incorporated subsequent edits

Dr. Green made the following contributions to this manuscript:

- Provided the impetus for using the Richart-Newmark method to estimate volumetric strains in sands
- Directed Chen in his laboratory work

Qingsheng Chen contributed in the following ways:

- Performed virtually all the laboratory cyclic simple shear testing and some of the index testing of the sand

Dr. Rodriguez-Marek:

- Provided statistical advice, including using the bootstrapping technique

Dr. Green and Dr. Rodriguez-Marek both:

- Provided valuable feedback throughout the research
- Edited the manuscript

An Approach for Estimating Seismic Compression Using Site Response Analyses

Samuel J. Lasley[1], Russell A. Green[2], Qingsheng Chen[3], Adrian Rodriguez-Marek[4]

[1] Doctoral Candidate, Department of Civil and Environmental Engineering, Virginia Tech, Blacksburg, Virginia, U.S.A.; slasley@vt.edu

[2] Professor, Department of Civil and Environmental Engineering, Virginia Tech, Blacksburg, Virginia, U.S.A.; rugreen@vt.edu

[3] Associate Research Fellow, School of Civil, Mining, and Environmental Engineering, University of Wollongong, New South Wales, Australia; qingsheng_chen@uow.edu.au

[4] Professor, Department of Civil and Environmental Engineering, Virginia Tech, Blacksburg, Virginia, U.S.A.; adrianrm@vt.edu

3.1 Abstract

Seismic compression is the accrual of volumetric strains in unsaturated soils caused by cyclic loading and has caused significant damages to buildings and other structures during earthquakes. To date, the available methods for predicting the severity of seismic compression have mainly been "simplified" procedures, in which a number of equivalent cycles is used to represent the duration of earthquake loading. Often, however, the number of equivalent cycles is computed inconsistently with the underlying mechanics of seismic compression. Proposed herein is a "non-simplified" procedure for predicting the severity of seismic compression. The procedure is based on a modified version of the Richart-Newmark cumulative damage hypothesis, wherein volumetric strain is used as the damage metric. The proposed model was calibrated using data from 460 constant amplitude sinusoidal strain-controlled cyclic simple shear tests performed on clean sand and validated using test data from samples subjected to variable amplitude sinusoidal and earthquake loadings. In addition to predicting the severity of seismic compression, the proposed model can be used to compute number of equivalent shear strain cycles for use in simplified models, consistent with the seismic compression

phenomenon. In comparison with other proposed "non-simplified" models for computing seismic compression, the proposed model gives good agreement with the measured seismic compression.

3.2 Introduction

The objective of the research presented herein is the development of a novel approach for predicting the severity of seismic compression. Seismic compression is the phenomenon in which unsaturated/partially-saturated soils undergo volumetric strain (densification) when subjected to earthquake motions and other types of cyclic loading. This phenomenon has caused significant damage to buildings and other structures, especially those built upon compacted fills (e.g., Slosson 1975; Siddharthan and El-Gamal 1996; Stewart et al., 2002). The proposed methodology uses strain time histories obtained from site response analyses in contrast to existing "simplified" approaches which use a simpler parameterization of the time histories.

A large body of research has been performed to understand seismic compression and to predict its magnitude. Silver and Seed (1971) showed that the magnitude of seismic compression is a function of the amplitude of the applied cyclic shear strain and the relative density of the soil. Based on this work, Seed and Silver (1972) proposed one of the first "simplified" methods to estimate seismic compression. In this context, "simplified" methods are those that do not require numerical site response analyses be performed to estimate the shear stresses or strains induced by earthquake shaking. These methods often represent the earthquake loading as a sinusoidal motion with a duration quantified by number of equivalent cycles (e.g., Seed et al. 1975).

Since the initial procedure proposed by Seed and Silver (1972), several other simplified procedures have been proposed to predict the severity of seismic compression (e.g., Tokimatsu and Seed 1987; Pradel 1998; Stewart and Whang 2003; Duku et al. 2008). Additionally, several laboratory studies have examined the effect of saturation, relative density, fines content, mineralogy, fabric, OCR, plasticity index, overburden stress, and multi-directional shaking on seismic compression (e.g., Pyke et al. 1975; Chu and Vucetic 1992; Whang 2001; Hsu and Vucetic 2004; Stewart et al. 2004; Whang et al. 2004; Sawada et al. 2006; Duku et al. 2006, 2008; Carter et al. 2014).

As a result of this work, there is a good understanding of environmental and compositional factors that influence the magnitude of seismic compression.

In comparison to simplified procedures, very few "non-simplified" procedures have been proposed for evaluating seismic compression, Finn and Byrne (1976) being one of them. In this context, a "non-simplified" procedure is one that requires numerical site response analyses be performed, wherein the earthquake shaking is represented in a form similar to recorded motions (e.g., an acceleration time history). The main advantage of a non-simplified procedure is that it does not rely on number of equivalent cycles correlations to represent the earthquake shaking. The disadvantages are that a more detailed characterization of the site being analyzed is required and additional expertise and effort are required to select appropriate ground motions and to perform numerical site response analyses. However, most of these barriers have been reduced over the years as cost-effective methods for characterizing sites for dynamic analyses are becoming readily available (e.g., SASW/-MASW) and as protocols for selecting input time histories and performing the site response analyses have been developed (e.g., Kottke and Rathje 2008; Bradley 2012; Stewart and Kwok 2008).

The Finn and Byrne (1976) method for evaluating seismic compression is based on the Martin et al. (1975) model for computing volumetric strain in dry sands. This model has since been updated by Byrne (1991) and yields the cumulative volumetric strain at each half cycle of shear strain loading, whether the loading is constant or variable amplitude. As proposed by Finn and Byrne (1976), the shear strain time history used to compute the volumetric strain in a given layer in a soil profile is obtained from a numerical site response analysis, and the magnitude of the seismic compression is equal to summation of the volumetric strains in each layer in the profile. Inherent to the Finn and Byrne approach is the decoupling of the site response and volumetric strain computations, which allows equivalent linear site response analyses to be used (e.g., Schnabel et al. 1972). Alternatively, site response analyses could be performed using software that employ advanced elasto-plastic constitutive models to model the dynamic response of the soil (e.g., SUMDES: Li et al. 1992). Inherent to this approach is the coupling of shear and volumetric strains, yielding the magnitude of seismic compression directly. However, the effort and expertise required to calibrate advanced elasto-plastic constitutive models for use in site response analyses often limits their use to only high-end projects. In contrast, performing equivalent linear

site response analyses has become state-of-practice in many places, making non-simplified procedures similar to the one proposed by Finn and Byrne (1976) a viable option for predicting seismic compression in today's practice.

Proposed herein is a non-simplified approach for predicting seismic compression that builds on that proposed by Finn and Byrne (1976). In the proposed approach volumetric strains are computed using a modified version of the Richart-Newmark (R-N) fatigue model (Richart and Newmark, 1948), wherein volumetric strain is used as the damage metric. An extensive laboratory testing program was performed to develop, calibrate, and validate the modified R-N model for clean sands. The proposed approach allows seismic compression to be predicted more rigorously than using simplified procedures, without the onus of calibrating advanced elasto-plastic constitutive models. Additionally, the modified R-N model proposed herein can be used to compute number of equivalent strain cycles for use in simplified procedures. The advantage of this is that a consistent damage metric (i.e., volumetric strain) is used to equate earthquake motions to an equivalently damaging (i.e., induces the same volumetric strain) sinusoidal motion having a specified number of cycles. This is in contrast to the number of equivalent cycle correlations commonly used with simplified seismic compression approaches, where the equivalent cycle correlations are developed using damage metrics other than volumetric strain (e.g., Green and Terri, 2005).

This paper starts by presenting an overview of fatigue models including: a description of the Palmgren-Miner (P-M) model, which has been extensively used in geotechnical earthquake engineering, the original R-N model, and the Byrne (1991) volumetric strain model. The results of sinusoidal cyclic simple shear tests, which were used to develop and calibrate the modified R-N model, are presented next. The modified R-N model is then validated for computing seismic compression using results from tests in which the samples were subjected to variable amplitude sinusoidal and earthquake loadings. A comparison of measured versus predicted volumetric strains is then made, where the modified R-N, Byrne (1991), and P-M models are used to make the predictions. Finally, a brief case history is presented to illustrate the proposed non-simplified approach for evaluating seismic compression.

3.3 Fatigue Models

Macro cumulative damage fatigue models were the focus of considerable research from the 1940s through the 1960s. Common to all macro cumulative damage fatigue models is the assumption that when a cyclic load above a certain magnitude is applied to a specimen, internal "damage" occurs in the specimen, such that its service life is shortened. As the loading continues, the damage accumulates until the specimen fails. This concept is illustrated in Figure 3.1, wherein the horizontal axis is the number of applied load cycles (n) and the vertical axis is degree of damage (D). As shown in this figure, failure occurs when the degree of damage reaches 100% (i.e., $D = 1$), with the corresponding number of load cycles designated as N.

Features that distinguish one cumulative damage fatigue model from another are: the assumed nature of damage accumulation as a function of the number of cycles of loading (e.g., linear accumulation, power function accumulation); whether the nature of damage accumulation is assumed to be a function of the amplitude of the load (i.e., load-dependent versus load-independent models); and whether the nature of damage accumulation is assumed to be influenced by previous loading (i.e., interaction versus interaction-free models) (e.g., Kaechele, 1963). Figure 3.2 conceptually shows the accumulation of damage in similar specimens subjected to cyclic loading of different amplitudes (S_x). As intuitively expected, the specimen subjected to the largest amplitude load fails in the fewest number of cycles, the specimen subjected to the second largest amplitude load fails in the second fewest number of cycles, and so on. The distinction between load-dependent and load-independent models is highlighted when the number of load cycles is normalized by the respective number of cycles to cause failure in the specimens (i.e., cycle ratio, R = n/N). If the cumulative damage curves for all load amplitudes collapse into one curve, such as shown in Figure 3.3, the fatigue model is referred to as *load-independent*. On the contrary, if cumulative damage curves do not collapse into one curve (e.g., Figure 3.3), the fatigue model is referred to as *load-dependent*.

If the relationship between cumulative damage and number of cycles corresponding to a specified load amplitude is not influenced by previous load cycles, the fatigue model is referred to as *interaction-free*. On the contrary, a model that assumes that the nature of the accumulation of damage is altered by prior loading is referred to as an *interaction* model (e.g., Kaechele,

1963; Stallmeyer and Walker, 1968). Figure 3.3 is an example of a load-dependent, interaction-free model. In contrast, a load-dependent, interaction model would have a family of damage curves for each load amplitude as opposed to a single curve; the damage accumulated would follow a given curve based on the amplitude of the current applied load and the previous load history.

3.3.1 Palmgren-Miner (P-M) Model

The Palmgren-Miner (P-M) fatigue model (Palmgren 1924; Miner 1945) is a load independent, interaction free model that assumes that damage accumulates linearly, as shown in Figure 3.4. Thus, damage accumulates according to the form:

$$D = \sum_{i=1}^{m} \frac{n_i}{N_i} \tag{3.1}$$

where m is the number of load sets (i.e., the number of different amplitudes of the peaks in the load time history), n_i is the number of cycles in load set i, and N_i is the number of cycles to 100% damage for the amplitude of load set i. Inherent to this model is the assumption that an increment in damage from a single cycle is directly proportional to the amplitude of the cycle (Seed et al. 1975; Annaki and Lee 1977; Liu et al. 2001). Also, the sequence of various amplitudes of cyclic loading does not affect the cumulative damage estimation at the end of loading, as illustrated in the right two panes of Figure 3.4. This is in contrast to load-dependent (interaction or interaction-free) cumulative damage models in which the sequence of load amplitudes affects the shape of the damage path and the accumulated damage at the end of loading (e.g., Green and Lee 2006; Carter et al. 2014). Consistent with the assumptions inherent to the P-M model, the model is also classified as a high-cycle fatigue model (Green and Terri 2005). This is because dissipated energy is used as the damage metric and the direct proportionality of the load amplitude and increment in damage implies the material remains in the linear range and has a constant damping ratio.

The P-M model is encapsulated in the equivalent-number-of-cycles concept which was introduced to geotechnical earthquake engineering in the late 1960s

and early 1970s (Lee and Chan 1972; Seed et al. 1975; Annaki and Lee 1977). The equivalent-number-of-cycles concept is widely used in geotechnical earthquake engineering and, in particular, for the analysis of liquefiable soils. Additionally, the equivalent-number-of-cycles concept has been adapted for use in seismic compression estimations (e.g. Tokimatsu and Seed 1975; Duku et al. 2008). However, unlike many fatigue-related phenomena in which the accumulation of "damage" is neither directly observable nor measurable at the macroscopic level, the accumulation of volumetric strain in dry and partially saturated sandy soils subjected to cyclic loading is both directly observable and measurable. Detailed laboratory tests (e.g., Silver and Seed 1971; Stewart et al. 2004; Whang et al. 2004; Duku et al. 2006, 2008) have shown that the rate of volumetric strain accumulation decreases with the increasing number of cycles. Additionally, the accumulation of volumetric strain when plotted as a function of R varies as a function of both load amplitude and load history (i.e., seismic compression is a load-dependent, interaction fatigue phenomenon). Based on this observed soil behavior, the validity of using the P-M model to compute number of equivalent cycles for use in evaluating seismic compression is questionable.

3.3.2 Richart-Newmark (R-N) Model

The Richart-Newmark (R-N) model, as originally proposed by Richart and Newmark (1948), is a load-dependent, interaction-free fatigue model and was proposed for use in seismic compression analyses by Green and Lee (2006). The R-N model has the form:

$$D = R^r \tag{3.2}$$

where r is a curve fitting parameter that varies as a function of the amplitude of loading. The model is plotted in Figure 3.5, and as shown in the left pane of this figure, the damage accumulation path for constant-amplitude cyclic loadings varies as a function of the amplitude of the applied load (i.e., the damage accumulation curve for each load amplitude has a different associated value of r). For variable-amplitude loading conditions, Eqn. 3.2 expands to

(Green and Lee 2006):

$$D_i = \left[(D_{i-1})^{1/r_i} + \left(\frac{n_i}{N_i} \right) \right]^{r_i} \text{ for } i = 1 \text{ to } m \tag{3.3}$$

where D_i is the total damage after the i^{th} set of constant-amplitude cycles of loading, r_i is the r value corresponding to the amplitude of the i^{th} set of loading cycles, n_i is the number of cycles in the i^{th} set of loading cycles, N_i is the value of N corresponding to conditions of the i^{th} set of loading cycles, and m is the number of loading sets. The damage accumulation path for a variable-amplitude loading "jumps" from one curve for a given load amplitude to another as the cyclic amplitude changes. For each "jump", the accumulated damage stays the same, but the cycle ratio will change. The center and right panes of Figure 3.5 illustrate how the damage accumulation paths jump from one constant-amplitude curve to another. Also illustrated in this figure is how the accumulated damage at the end of loading differs depending on the sequencing of the loading cycles, an inherent characteristic of a load-dependent fatigue behavior.

3.3.3 Byrne (1991) Model

Simplifying a previous model by Martin et al. (1975), Byrne (1991) proposed the following model to estimate volumetric strains in dry sands:

$$\varepsilon_v = \sum_i \left(\Delta\varepsilon_{v,1/2} \right)_i \tag{3.4}$$

where ε_v is the accumulated volumetric strain at the end of loading and $\left(\Delta\varepsilon_{v,1/2} \right)_i$ is the increment in volumetric strain for the i^{th} half shear strain cycle having an amplitude γ_i:

$$\left(\Delta\varepsilon_{v,1/2} \right)_i = 0.5\gamma_i C_1 \exp\left[-C_2 \frac{\varepsilon_{vi}}{\gamma_i} \right] \tag{3.5}$$

The variables C_1 and C_2 are material parameters, and ε_{vi} is the volumetric strain at the beginning of the i^{th} load increment. Byrne (1991) provided

equations to estimate C_1 and C_2:

$$C_1 = 7600 D_r^{-2.5} \tag{3.6}$$

$$C_2 = 0.4/C_1 \tag{3.7}$$

where D_r is the relative density of the sand in percent.

Although Martin et al. (1975) or Byrne (1991) never make reference to fatigue theories, their models are inherently load-dependent, interaction macro fatigue models in which volumetric strain is used as the damage metric. All-in-all, this is not surprising given that the detailed laboratory studies mentioned previously showed that the accumulation volumetric strains in soils subjected to cyclic loading is a load-dependent, interaction phenomenon.

3.4 Laboratory Testing Procedure

Strain-controlled cyclic simple shear tests were performed using a modified GCTS simple shear testing apparatus (SSH-100) (see Figure 3.6). The testing apparatus is capable of performing stress- or strain-controlled cyclic simple shear or cyclic triaxial tests with sinusoidal or user-defined (e.g., earthquake) loadings. Prior to these tests, the apparatus was slightly modified by the authors; external bracing was added to limit compliance during cyclic simple shear testing.

The cyclic simple shear tests were performed on poorly-graded quartz sand with an assumed specific gravity of 2.65. Table 3.1 summarizes the index properties of the sand, and Figure 3.7 is a plot of particle size gradation of the sand. The test specimens had dimensions of 101-mm diameter by approximately 24-mm high and were confined by stacked rings.

Sample specimens were prepared by dry pluviating the sand into the confining rings lined with a taut latex membrane. The left side of Figure 3.8 shows the confining rings with the freshly-pluviated sand. The top layer of the specimen was gently smoothed, the top platen was placed on the sand, and the membrane was affixed to the platens with o-rings (see the right photo in

Figure 3.8). The target density of each specimen was achieved by vibrating the specimen with the top platen in place.

After assembling the testing equipment around the specimen, an overburden pressure was applied to the specimen and further densification was allowed to occur. After the overburden-induced volumetric strains ceased to accumulate, the cyclic loading phase was initiated. One of three types of strain-controlled loading was applied to each specimen: (1) Constant-amplitude sinusoidal, (2) Variable-amplitude sinusoidal, or (3) Earthquake-type loading obtained from equivalent-linear site response analyses. Figure 3.9 shows examples of the three types of loading.

3.5 Modification and Calibration of the R-N Fatigue Model

The results of 460 constant-amplitude sinusoidal strain-controlled cyclic simple shear tests were used to modify and calibrate the R-N model. In order to characterize the influencing factors on seismic compression, tests were performed at three values of overburden pressure (i.e., 50, 100, and 250 kPa) and at four different nominal relative densities (i.e., 30, 45, 65 and 80%). Table 3.2 shows the number of tests performed at each combination of relative density and overburden pressure. For each combination, the specimens were subjected to cyclic shear strains ranging from 0.2 to 1% (single-amplitude). Figure 3.10 shows a typical result of the constant-amplitude sinusoidal strain-controlled cyclic simple shear tests.

3.5.1 Model Modification

As mentioned previously, the original R-N model was modified to bring the model in accord with soil response observed in laboratory tests (the other two models did not lend themselves to modification). The modifications entailed changing the R-N model from a load-dependent, interaction-free model to a load-dependent, interaction model. This was achieved by making r a function of R, as well as a function of the load amplitude and soil state. Several different functional forms of r were assessed, but the one that gave

the best results, yet maintained a relatively simple form is:

$$r = 0.1/R^{r_0} + r_0;\ 0 < r_0 < 1 \tag{3.8}$$

where r_0 is a calibration parameter that is a function of soil state and load amplitude.

3.5.2 Model Calibration

To use the modified R-N model for forward predictions of ε_v, two calibration parameters are needed, r_0 and N (recall: $R = n/N$, where N is a function of the load amplitude and soil state and n is from the applied loading function). Using the results from the 460 constant-amplitude sinusoidal strain-controlled cyclic simple shear tests, expressions for both r_0 and N were developed:

$$r_0 = 0.238 - 0.002D_r + 35.23\gamma - 0.0603\frac{\sigma'_v}{P_a} + 7.093\gamma \cdot \frac{\sigma'_v}{P_a} + \varepsilon_{r_0} \tag{3.9}$$

$$\ln(N) = -13.5 + -2.91\ln(\gamma) + 0.217\frac{\sigma_v}{P_a} + 0.0244D_r + \varepsilon_N \tag{3.10}$$

where: D_r is the relative density (in percent) of the specimen at the start of the cyclic phase of testing, γ is the single-amplitude shear strain of the loading cycles (as a decimal), σ'_v is the effective overburden stress, P_a is atmospheric pressure in the same units as σ'_v, and ε_{r_0} and ε_N are normally-distributed error terms with means of zero and standard deviations of 0.048 and 0.62, respectively. The distributions of the error terms were obtained via bootstrapping (Efron and Tibishirani 1994). Inherent to developing the expression for N, ε_v corresponding to "failure" (i.e., D = 1) needed to be specified. This value is somewhat arbitrary because the modified R-N model can predict damage states greater than 100% without issue. However, selecting a value of ε_v corresponding to D = 1 that is within the range of seismic compression expected in-situ during earthquakes ensures the modified R-N model is calibrated over the appropriate range of volumetric strains. Accordingly, for convenience, $\varepsilon_v = 1\%$ was selected to correspond to "failure." Note that this model should be used with caution outside the conditions

for which it was calibrated (i.e., $30\% \leq D_r \leq 100\%$; $50 \leq \sigma'_v \leq 250$ kPa; $0.2 \leq \gamma \leq 1.0\%$).

3.6 Accuracy of the Fatigue Models

3.6.1 Variable-Amplitude Sinusoidal Loading

To assess the accuracy of the three fatigue models in predicting the magnitude of seismic compression during earthquakes, the models were used to predict the volumetric strains in laboratory samples subjected to variable-amplitude sinusoidal and earthquake loadings. In total, twenty three variable-amplitude sinusoidal load tests were performed on samples having a relative density of 55% and an overburden pressure of 100 kPa. The amplitude of the sinusoids was increased or decreased twice (i.e., the load functions consisted of three sets of constant-amplitude sinusoids). Figure 3.11 shows one of the shear strain load functions used. About half of the load functions used started with high amplitude cycles and had step decreases in the load amplitudes with time and the other half started with low amplitude cycles and had step increases in load amplitudes with time.

For each of the 23 variable-amplitude laboratory tests, the seismic compression was estimated using the P-M, Byrne (1991), and the modified R-N models. For the Byrne (1991) model, the calibration parameters C_1 and C_2 were computed using Eqns. 3.6 and 3.7. For the P-M and modified R-N models, N and r_0 were computed using Eqns. 3.9 and 3.10. Note that N is the only calibration parameter for the P-M model. However, unlike the modified R-N model where the volumetric strain selected to correspond to $D = 1$ is arbitrary and inconsequential in computing seismic compression, this is not the case for the P-M model. As a result, to make a fair assessment of the P-M model, the volumetric strain corresponding to $D = 1$ was selected based on the slope of a plot of $\log(\gamma)$ vs. $\log(N)$ curve from the constant amplitude sinusoidal laboratory test data. The slope of the $\log(\gamma)$ vs. $\log(N)$ varied from 2.63 for $\varepsilon_v = 0.1\%$ to 0.34 for $\varepsilon_v = 1.0\%$. In comparison, the slope of the normalized $\log(CSR)$ vs. $\log(N)$ curve used by Liu et al. (2001) to compute number of equivalent stress cycles using the P-M model was approximately 0.35. Accordingly, using N corresponding to $\varepsilon_v = 1\%$ for the P-M model is consistent with how the P-M model has been used in past

applications in geotechnical earthquake engineering. Using the models thusly calibrated, Figure 3.12 shows both the predicted and measured volumetric strains for the variable-amplitude loading function shown in Figure 3.11.

As expected, the modified R-N and Byrne models do much better at predicting the cumulative volumetric strain with time than does the P-M model. This is because both modified R-N and Byrne models are load-dependent, interaction models, while the P-M model is load-dependent, interaction free. Table 3.3 shows the median and interquartile range in the prediction errors for each model. (Interquartile range is the difference between the third and first quartile of the ordered data; the middle fifty percent of the data is encompassed within this range.)

Figure 3.13 shows boxplots of the percent error in prediction for each method. The top and bottom of each box show the location of the 3rd and 1st quartile of the data, and the horizontal line between these two is the median. Any points outside the whiskers are considered outliers. It can be seen that both the R-N and Byrne models generally over-predict volumetric strains, and the P-M model under-predicts it. The measured volumetric strains ranged from 0.5% to 0.97% with a median of 0.7%.

3.6.2 Earthquake Loading

A total of 16 strain-controlled cyclic simple shear laboratory tests were performed using earthquake motions as the loading function. Three different motions were used; these were shear strain time histories computed at the center of a layer in a soil profile using an equivalent-linear site response code. Per Ishihara and Yasuda (1973), one of these motions was a "vibration-type" motion having many high amplitude cycles, and the other two motions were "shock-type" motions having few large-amplitude cycles (Figure 3.14). All three motions were scaled to a maximum shear strain amplitude of 0.35%.

To estimate the volumetric strains for earthquake-type loadings using the fatigue models, it was necessary to define a "cycle" or, more appropriately, a "half cycle" of loading. The definition used in this study is a version of the mean crossing peak counting (also known as zero-crossing) method as described in ASTM E1049 (2011). In this method, zero-crossings define the half cycles and the amplitude of the half cycles was taken as the maximum load between the zero-crossings. Other methods of defining half cycles of

loading were also assessed, namely the rainflow and range counting methods (ASTM E1049 2011). However, these methods resulted in similar or larger errors in the predictions of all the models. Accordingly, given the simplicity (and precedence: Seed et al. 1975; Byrne 1991; Liu et al. 2001) of using the mean crossing peak counting definition, it was adopted by the authors. Also, in predicting the volumetric strains for the earthquake loadings using the three models, it was assumed that no volumetric strains resulted from shear strains having an amplitude less than the threshold shear strain of 0.02% (Dobry et al. 1982; Hsu and Vucetic 2004). This is consistent with how Byrne (1991) implemented his model for predicting pore pressure generation in saturated, undrained sandy soils.

Measured volumetric strains for the earthquake loadings ranged from 0.22 to 0.42% with a median value of 0.28%. Figure 3.15 shows the measured and predicted volumetric strains for the motion shown at the bottom of Figure 3.14. For these tests, all three models, on average, under-predict the volumetric strain. The Byrne model has the lowest median percent error. The P-M model gave the worst median prediction, but had the smallest interquartile range. The modified R-N model's median prediction was slightly worst than Byrne's, but the interquartile range was slightly better. Table 3.4 shows the median and interquartile range of the percent error in prediction for each model. Figure 3.16 shows the range and median of the predictions for all the earthquake loading tests.

3.6.3 Overall Prediction Results

Table 3.5 lists the combined results of all the variable-amplitude sinusoidal and earthquake loadings. As may be observed from the results shown in this table, the Byrne model gave slightly more accurate predictions than the modified R-N model for the irregular loading tests. The Palmgren-Miner method yielded the least accurate predictions. It needs to be mentioned that the results shown in Table 3.5 only correspond to samples having a relative density of 55% and an effective overburden stress of 100 kPa. As discussed in the case history example in the next section, for conditions other than $D_r = 30\%$ and $\sigma'_v = 100$ kPa, the modified R-N model continues to yield accurate results, while the accuracy of the Byrne model decreases substantially, and the P-M model remains inaccurate. Finally, the same relative trends in the prediction errors of these three models for volumetric

strains would be expected in the computed number of equivalent cycles for the variable-amplitude sinusoidal or earthquake loads.

3.7 Case History

To illustrate the implementation of the proposed model for predicting seismic compression in a field setting, a case history presented in Stewart et al. (2002) is analyzed. The case history is of seismic compression in a deep canyon fill caused by the 1994 M_w6.7 Northridge Earthquake. 'Site A' is located approximately 12.2 km from the fault rupture plane and experienced settlements ranging from 0 to 18 cm.

Table 3.6 gives the soil layer properties used for the analysis. This layering represents the stratigraphy near the site where 15.2 cm of settlement was recorded. The relative compaction values (RC) were obtained by randomly sampling from a distribution given by Stewart et al. (2002). The relative densities were estimated using a correlation with relative compaction by Lee and Singh (1971). Also, six pairs of horizontal earthquake recordings were taken from the PEER NGA ground motion database (Chiou et al. 2008). These motions were from earthquakes with similar magnitudes and were sited at similar distances as this case history. Details of these motions are given in Table 3.7; the site to source distance listed in the table is the closest distance to the fault rupture.

Equivalent-linear site response analyses (Schnabel et al. 1972; Lasley et al. 2014) were performed for the site and shear strain time histories were computed at the center of each layer. From these, the volumetric strains were computed using the modified R-N model. The magnitude of seismic compression for the profile was computed by multiplying the computed volumetric strains for each layer by the layer thickness and summing the results for all the layers. Table 3.8 lists the predicted settlements for the profile for each ground motion. To obtain the total settlement, settlements from each motion of each pair were summed. The average settlement predicted for the six pairs of motions was 11.4 cm, slightly less than the measured 15.2 cm of settlement.

Volumetric strains were also computed using the Byrne and P-M models, but their predicted settlements averaged 2.7 cm for the Byrne model and 1.9 cm

for the P-M model, for this profile. The analyses were repeated for a profile wherein all the layers were assumed to have relative densities of 30%. Similar to the predictions of the laboratory results discussed previously for the variable amplitude sinusoidal and earthquake loadings, the modified R-N and the Byrne model predicted very similar magnitudes of seismic compression on average (R-N: 7.5 cm, Byrne: 10.6 cm), while the P-M model predicted an average settlement of 2.9 cm. Accordingly, it appears that the Byrne model is most accurate at low relative densities.

3.8 Summary and Conclusions

The authors proposed a "non-simplified" procedure for predicting the severity of seismic compression. It is based on a modified version of the R-N fatigue model in which volumetric strain is used as the damage metric. In contrast to "simplified" approaches for predicting seismic compression, the proposed model evaluates volumetric strains directly from shear strain time histories computed from equivalent linear site response analyses, thus negating the need for number of equivalent cycle correlations. Modifications were made to the R-N fatigue model to make it a load dependent, interaction model, in accord with the seismic compression phenomenon. To calibrate the modified R-N model, several hundred strain-controlled sinusoidal cyclic simple shear tests were performed on dry samples composed of quartz sand. The induced volumetric strain accumulation in the sand was found to be a function of the amplitude of shear straining, the relative density of the sand, the applied overburden pressure, and the cycle ratio.

In addition to the sinusoidal loadings, 23 variable-amplitude sinusoidal and earthquake-type loading cyclic simple shear tests were also performed on specimens having a relative density of 55% and subjected to an overburden pressure of 100 kPa. The modified R-N, Byrne, and P-M models were used to predict the volumetric strains for these laboratory tests. Of the three models, the P-M model gave the least accurate predictions (significantly less accurate than the other models), and the modified R-N and Byrne models provided comparatively accurate predictions. However, the accuracy of the Byrne model decreases for conditions of higher relative density, as was illustrated in the analysis of a case history, while the modified R-N model continued to yield reasonable results.

In addition to being used to compute seismic compression in a non-simplified approach, the modified R-N model can also be used to compute number of equivalent strain cycles for use in simplified procedures. The advantage of this is that a consistent damage metric (i.e., volumetric strain) is used to equate earthquake motions to an equivalently damaging (i.e., induces the same volumetric strain) sinusoidal motion having a specified number of cycles. This is in contrast to the number of equivalent cycle correlations commonly used with simplified seismic compression approaches, where the equivalent cycle correlations are developed using damage metrics other than volumetric strain.

3.9 Acknowledgements

This study is based on work supported by the U.S. National Science Foundation (NSF) grants CMMI-1030564 and CMMI-1435494, and US Army Engineer Research and Development Center (ERDC) grant W912HZ-13-C-0035. The authors gratefully acknowledge this funding. However, any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of NSF or ERDC.

3.10 References

Annaki, M., and Lee, K. L. (1977). "Equivalent Uniform Cycle Concept for Soil Dynamics." *Journal of the Geotechnical Engineering Division*, 103(6), 549-564.

ASTM Standard E1049, 1985 (2011). "Practices for Cycle Counting in Fatigue Analysis." *ASTM International*, West Conshohocken, PA, www.astm.org

Bradley B.A. (2012). "A ground motion selection algorithm based on the generalized conditional intensity measure." *Soil Dynamics and Earthquake Engineering*, 40, 48-61.

Byrne, P. (1991). "A cyclic shear-volume coupling and pore pressure model for sand." *Proceedings of the Second International Conference on Recent Advances in Geotechnical Earthquake Engineering and Soil Dynamics*, St. Louis, MO, March, Paper, 47-55.

Carter, W.L., Green, R.A., Bradley, B.A., and Cubrinovski, M. (2014). "The Influence of Near-Fault Motions on Liquefaction Triggering during the Canterbury Earthquake Sequence", *Soil Liquefaction during Recent Large-Scale Earthquakes* (R.P. Orense, I. Towhata, N. Chouw, eds.), CRC Press, Leiden, The Netherlands, 57-68.

Chiou, B., Darragh, R., Gregor, N., and Silva, W. (2008). "NGA Project Strong-Motion Database." *Earthquake Spectra*, 24(1), 23:44.

Chu, H-H, and M. Vucetic. (1992). "Settlement of Compacted Clay in a Cyclic Direct Simple Shear Device." *ASTM Geotechnical Testing Journal*, 15(4), 371-79.

Dobry, R., Ladd, R. S., Yokel, F. Y., Chung, R. M., and Powell, D. (1982). "Prediction of pore water pressure buildup and liquefaction of sands during earthquakes by the cyclic strain method." NBS Build. Sci. Ser. 138, National Bureau of Standards, Gaithersburg, MD.

Duku, P. M., Stewart, J. P., and Whang, D. H. (2006). "Effect of post-compaction ageing on seismic compression of fine-grained soils." *Geotechnical Special Publication*, ASCE, 152, 411.

Duku, P. M., Stewart, J. P., Whang, D. H., and Yee, E. (2008). "Volumetric strains of clean sands subject to cyclic loads." *Journal of geotechnical and geoenvironmental engineering*, 134(8), 1073-1085.

Efron, B., & Tibshirani, R. J. (1994). *An introduction to the bootstrap (Vol. 57)*. CRC Press Chicago.

Finn, W. D. L., and Byrne, P. M. (1976). "Estimating settlements in dry sands during earthquakes." *Canadian Geotechnical Journal*, 13(4), 355-363.

Green, R. A., and Lee, J. (2006). "Computation of number of equivalent strain cycles: a theoretical framework." *Geomechanics II*, American Society of Civil Engineers, 471-487.

Green, R. A., and Terri, G. A. (2005). "Number of equivalent cycles concept for liquefaction evaluations-Revisited." *Journal of Geotechnical and Geoenvironmental Engineering*, 131(4), 477-488.

Hsu, C.-C., and Vucetic, M. (2004). "Volumetric threshold shear strain for cyclic settlement." *Journal of Geotechnical and Geoenvironmental Engineering*, ASCE, 130(1), 58-70.

Ishihara, K., and Yasuda, S. (1973). "Sand liquefaction under random earthquake loading condition." *Proceedings of the 5th World Conference on Earthquake Engineering*. Rome: ASCE, 329-338.

Kaechele, L. (1963). *Review and analysis of cumulative-fatigue-damage theories*. DTIC Document.

Kottke, A. and Rathje, E.M. (2008). "A semi-automated procedure for selecting and scaling recorded earthquake motions for dynamic analysis." *Earthquake Spectra*, 24 (4), 911-932.

Lasley, S. J., Green, R. A., and Rodriguez-Marek, A. (2014). "Comparison of equivalent-linear site response analysis software." *Proceedings of the 10th U.S. National Conference on Earthquake Engineering*, Earthquake Engineering Research Institute, Anchorage, AK.

Lee, K.L. and Chan, K. (1972). "Number of equivalent significant cycles in strong motion earthquakes." *Proc. Int. Conf. on Microzonation for Safer Construction Research and Application*, University of Washington, Vol. 2, 609-627.

Lee, K. L., and Singh, A. (1971). "Relative density and relative compaction." *Journal of the Soil Mechanics and Foundations Division*, 97(7), 1049-1052.

Li, X. S., Z. L. Wang, and C. K. Shen (1992). *SUMDES: a nonlinear procedure for response analysis of horizontally-layered sites subjected to multidirectional earthquake loading*, Dept. of Civil Engineering, Univ. of California, Davis.

Liu, A. H., Stewart, J. P., Abrahamson, N. A., and Moriwaki, Y. (2001). "Equivalent number of uniform stress cycles for soil liquefaction analysis." *Journal of Geotechnical and Geoenvironmental Engineering*, 127(12), 1017-1026.

Martin, G.R., Finn, W.D.L., and H.B. Seed (1975). "Fundamentals of liquefaction under cyclic loading." *Journal of the Geotechnical Engineering Division*, ASCE, 101(GT5): 423-438.

Miner, M. A. (1945). "Cumulative damage in fatigue." *Journal of Applied Mechanics*, 12(3), A159-A164.

Palmgren, A. (1924). "Die lebensdauer von kugellagern (Life length of roller bearings, in German)." *Zeitschrift des Vereins Deutscher Ingenieure*, 68(14), 339-341.

Pradel, D. (1998). "Procedure to evaluate earthquake-induced settlements in dry sandy soils." *Journal of Geotechnical and Geoenvironmental Engineering*, 124(4), 364-368.

Pyke, R. M., Chan, C. K., and Seed, H. B. (1975). "Settlement of sands under multidirectional shaking." *Journal of the Geotechnical Engineering Division*, ASCE, 101(4), 379-398.

Richart, F.E. and Newmark, N.M. (1948). "An hypothesis for determination of cumulative damage in fatigue," *ASTM Proceedings*, 48: 767-800.

Sawada, S., Tsukamoto, Y., and Ishihara, K. (2006). "Residual deformation characteristics of partially saturated sandy soils subjected to seismic excitation." *Soil Dynamics and Earthquake Engineering*, 26(2-4), 175-182.

Schnabel, P., Seed, H. B., and Lysmer, J. (1972). "Modification of seismograph records for effects of local soil conditions." *Bulletin of the Seismological Society of America*, 62(6), 1649-1664.

Seed, H. B., and Silver, M. L. (1972). "Settlement of dry sands during earthquakes." *Journal of Soil Mechanics & Foundations Div.*, ASCE, 98(SM4), 381-397.

Seed, H. B., Idriss, I. M., Makdisi, F., and Banerjee, N. (1975). *Representation of Irregular Stress Time Histories by Equivalent Uniform Stress Series in Liquefaction Analyses*, EERC 75-29. Earthquake Engineering Research Center, University of California, Berkeley.

Siddharthan, R. V., and El-Gamal, M. (1996). "Earthquake induced ground settlements of bridge abutment fills." *Analysis and Design of Retaining Structures*, Geotechnical Special Publication No. 60, ASCE, New York, 100-123.

Silver, M. L., and Seed, H. B. (1971). "Volume changes in sands during cyclic loading." *Journal of Soil Mechanics & Foundations Div.*, ASCE, 97(SM9), 1171-1182.

Slosson, J.E. (1975). "Chapter 19: Effects of the earthquake on residential areas." *San Fernando, California, Earthquake of 9 February 1971,* Bulletin 196, California Division of Mines and Geology, Sacramento Calif.

Stallmeyer, J.E. and Walker, W.H. (1968). "Cumulative damage theories and applications." *Journal of the Structural Division*, ASCE, 94(ST12):2739-2750.

Stewart, J. P., and Kwok, A. O. (2008). "Nonlinear seismic ground response analysis: Code usage protocols and verification against vertical array data." *Geotechnical Earthquake Engineering and Soil Dynamics IV* (D. Zeng, M. T. Manzari, and D. R. Hiltunen, eds.), ASCE, Reston, VA, 1-24.

Stewart, J. P., Smith, P. M., Whang, D. H., and Bray, J. D. (2002). *Documentation and Analysis of Field Case Histories of Seismic Compression During the 1994 Northridge, California Earthquake*. Pacific Earthquake Engineering Research Center, University of California, Berkeley.

Stewart, J. P., and D. H. Whang. (2003). "Simplified procedure to estimate ground settlement from seismic compression in compacted soils." *Proceedings 2003 Pacific Conference on Earthquake Engineering*, Christchurch, New Zealand, Vol. 46.

Stewart, J. P., Whang, D. H., Moyneur, M., and Duku, P. (2004). "Seismic compression of as-compacted fill soils with variable levels of fines content and fines plasticity." Consortium of Universities for Research in Earthquake Engineering, Richmond, CA.

Tokimatsu, K., and Seed, H. B. (1987). "Evaluation of settlements in sands due to earthquake shaking." *Journal of Geotechnical Engineering*, ASCE, 113(8), 861-878.

Whang, D. (2001). "Seismic compression of compacted fills." Ph.D. Thesis, University of California, Los Angeles, Civil and Environmental Engineering Department.

Whang, D.H., Stewart, J.P., and Bray, J.D. (2004). "Effect of compaction conditions on the seismic compression of compacted fill soils," *Geotechnical Testing Journal*, ASTM, 27 (4), 371-379.

3.11 Tables

Table 3.1: Index properties of the quartz sand used in this study.

e_{min}	0.68
e_{max}	0.90
G_s	2.65 (assumed)
D_{50}	0.58 mm
C_u	1.9
$\gamma_{d,min}$	1.39 g/cm^3
$\gamma_{d,max}$	1.58 g/cm^3

Table 3.2: Summary of Sinusoidal Tests

σ_v	$D_{r,nom.}$	No. Tests
50	30	42
50	45	37
50	65	32
50	80	28
100	30	82
100	45	66
100	65	49
100	80	29
250	30	30
250	45	25
250	65	19
250	80	21
-	Total:	460

Table 3.3: Median and Interquartile Range of the Percent Error Prediction for Variable-Amplitude Sinusoidal Loadings

Method	Median % Error	25th Percentile	75th Percentile	IQR of % Error
R-N	-6.2	-17.2	6.2	23.4
Byrne	-3.3	-21.7	4.7	26.5
P-M	31.6	11.5	37.6	26.1

Table 3.4: Median and Interquartile Range of the Percent Error Prediction for Earthquake Motions

Method	Median % Error	25th Percentile	75th Percentile	IQR of % Error
R-N	12.7	5.7	23.5	17.8
Byrne	5.3	-1.2	17.0	18.2
P-M	90.7	87.5	94.4	6.9

Table 3.5: Median and Interquartile Range of the Percent Error Prediction for All Irregular Motions

Method	Median % Error	25th Percentile	75th Percentile	IQR of % Error
R-N	5.3	-7.5	11.7	19.3
Byrne	1.9	-11.0	8.0	19.0
P-M	47.5	29.5	90.5	61.0

Table 3.6: Soil Profile at Location B-2 of Site A (after Stewart et al. 2002).

Layer No.	Soil Type	Thickness (m)	V_s (m/s)	RC (%)	D_r (%)
1	Fill	1.0	200.0	96.0	81.0
2	Fill	1.3	214.0	97.0	83.0
3	Fill	1.4	190.0	95.0	77.0
4	Fill	1.7	239.0	96.0	80.0
5	Fill	1.4	199.0	94.0	71.0
6	Fill	1.6	143.0	89.0	44.0
7	Fill	1.4	272.0	93.0	65.0
8	Fill	1.7	299.0	95.0	77.0
9	Fill	1.4	268.0	95.0	74.0
10	Fill	1.5	354.0	89.0	45.0
11	Fill	1.6	253.0	92.0	62.0
12	Claystone	2.1	455.0		
13	Claystone	2.5	537.0		
Half-space	Claystone	-	886.0		

Table 3.7: Earthquake Recordings Used in This Case History Study

NGA No.	Earthquake	Station	M_w	Distance (km)
164	Imperial Valley-06	Cerro Prieto	6.53	15.19
265	Victoria, Mexico	Cerro Prieto	6.33	14.37
285	Irpinia, Italy-01	Bagnoli Irpinio	6.9	8.18
763	Loma Prieta	Gilroy - Gavilan Coll.	6.93	9.96
828	Cape Mendocino	Petrolia	7.01	8.18
1787	Hector Mine	Hector	7.13	11.66

Table 3.8: Estimated Settlements (cm) Across the Entire Profile for Each Input Ground Motion

Motion	From Motion 1	From Motion 2	Total Settlement
164	4.6	4.2	8.8
828	15.4	9.0	24.4
265	5.9	4.6	10.5
1787	4.0	5.9	9.9
763	4.7	4.5	9.1
285	3.5	2.4	5.9

3.12 Figures

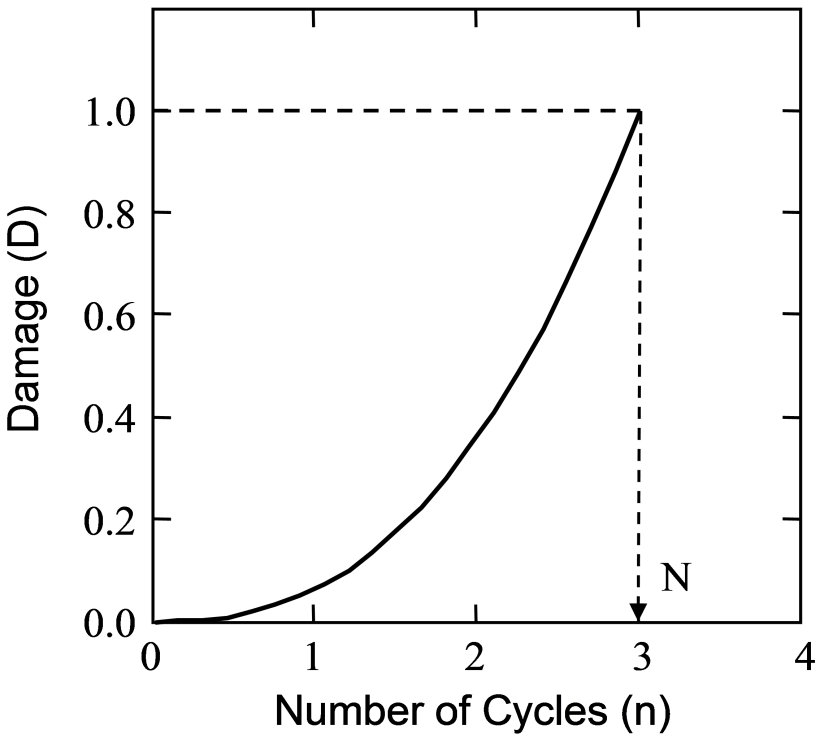


Figure 3.1: Relationship between cumulative damage (D) and the number of applied load cycles (n) having amplitude S. Failure occurs when D=1, with the corresponding number of applied cycles designated as N. (Adapted from Stallmayer and Walker, 1968) [Fair use.]

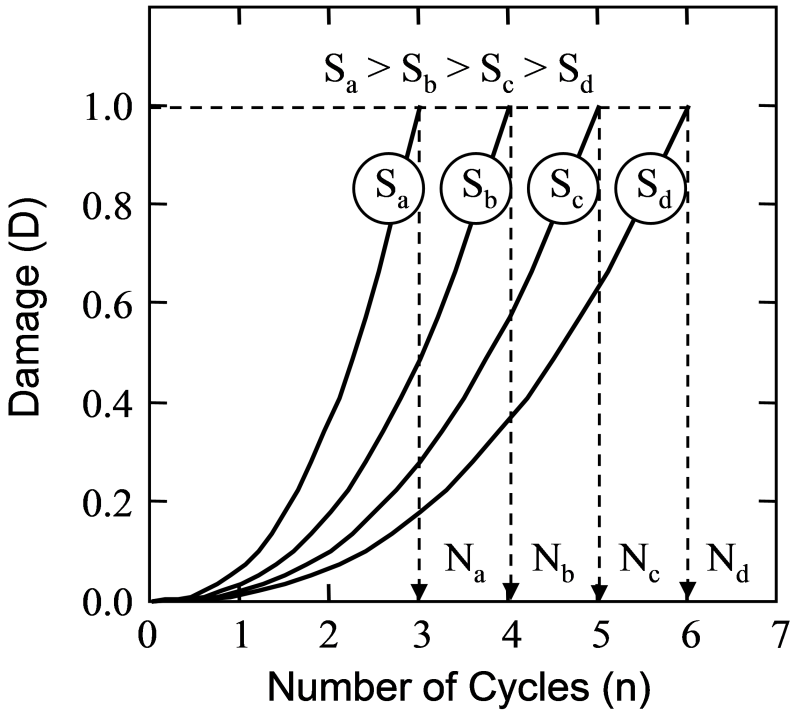


Figure 3.2: Damage-cycle relationship for various load amplitudes. (Adapted from Stallmeyer and Walker, 1968) [Fair use.]

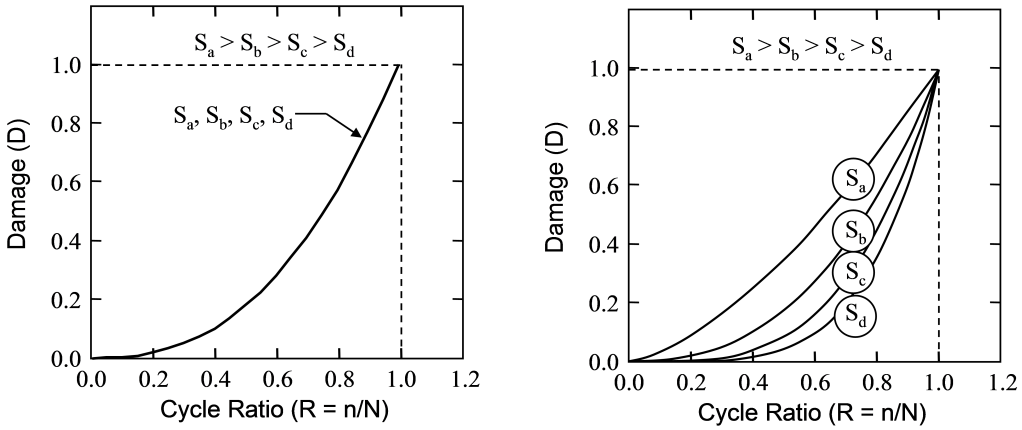


Figure 3.3: Damage-cycle ratio relationships for a) load-independent material, and b) load-dependent material. (Adapted from Stallmeyer and Walker, 1969) [Fair use.]

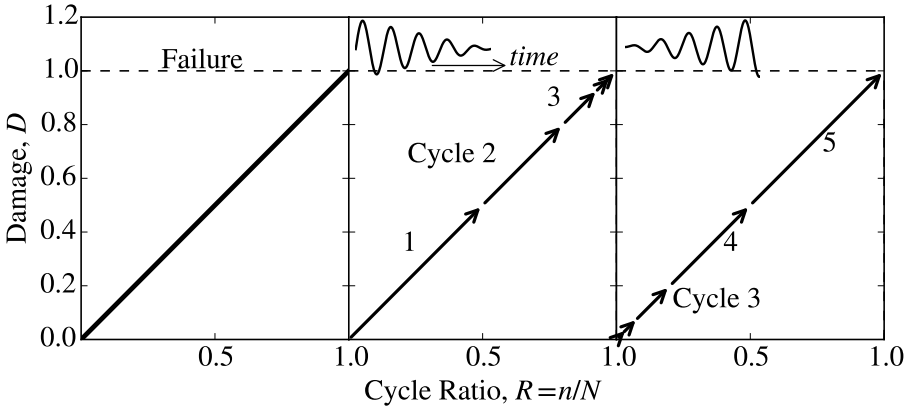


Figure 3.4: The left plot shows the damage accumulation path according to the Palmgren-Miner hypothesis. Two different cyclic loading time histories will follow the same damage accumulation path, as shown in the center and right plots. This is because the P-M hypothesis is load-independent.

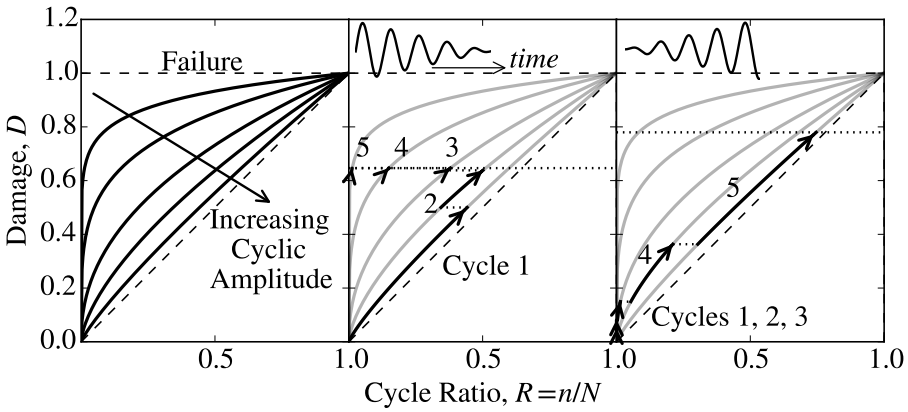


Figure 3.5: At left, various load accumulation paths predicted by the Richart-Newmark hypothesis for a range of r values. Right and center, the R-N hypothesis is load-dependent, meaning that the ordering of load cycles affects the damage accumulation path and the amount of predicted damage.

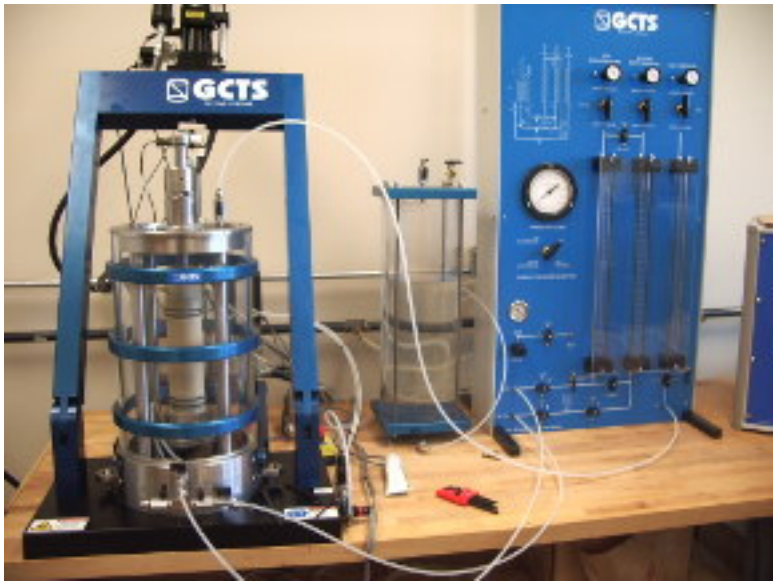


Figure 3.6: The GCTS testing apparatus used for this study with triaxial specimen. (Modifications not shown.)

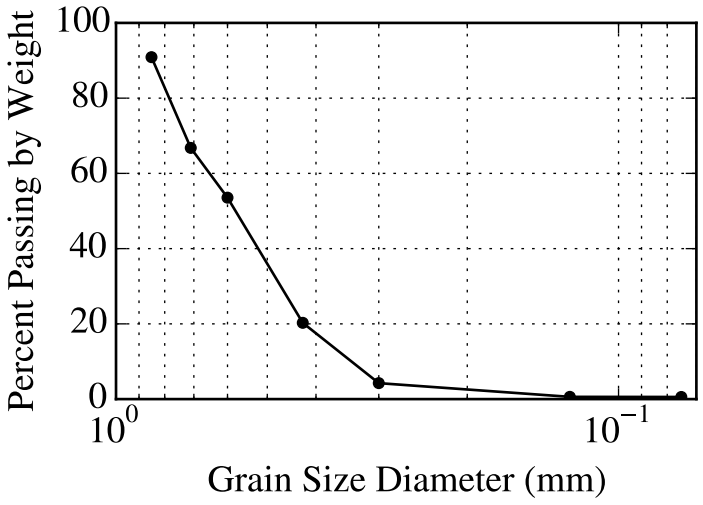


Figure 3.7: The particle size gradation of the quartz sand used in this study.

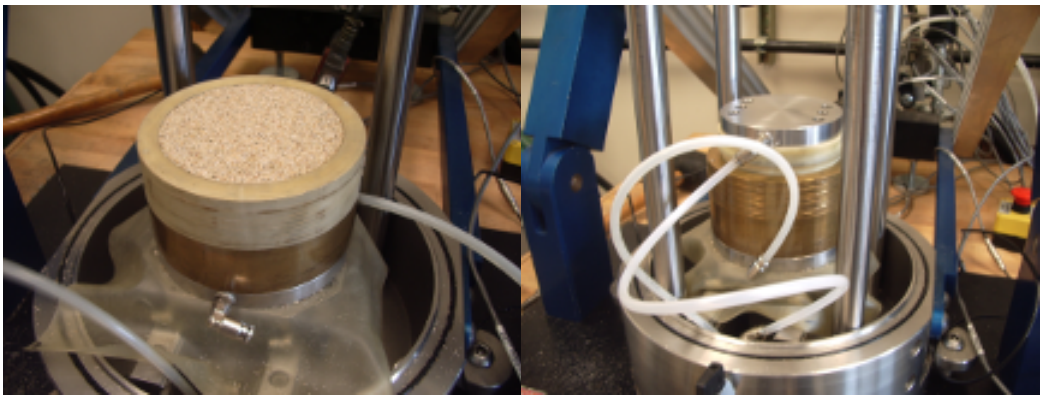


Figure 3.8: On the left, confining rings with pluviated sand. Right, specimen with top platen in place and under vacuum. (Note: the pictured sand was not used in this study.)

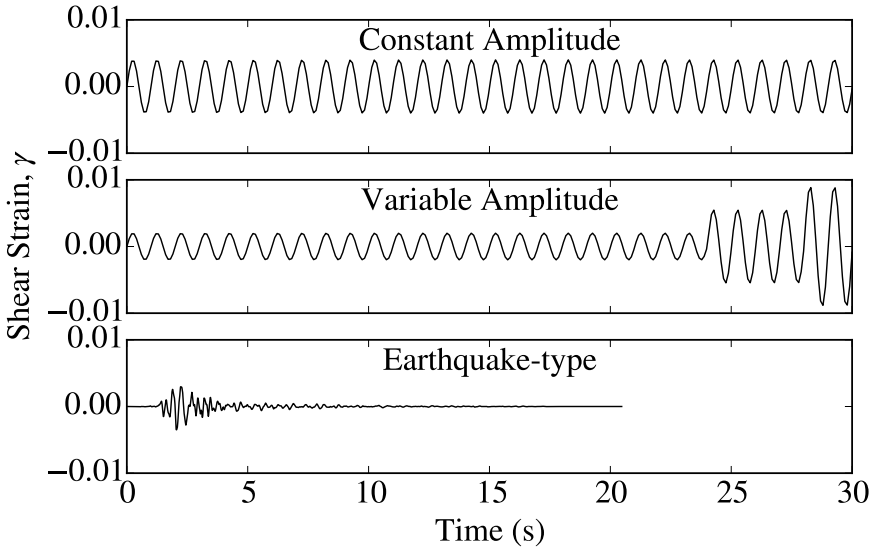


Figure 3.9: Examples of the constant-amplitude sinusoidal, variable-amplitude sinusoidal, and earthquake-type loadings used for the cyclic simple shear testing.

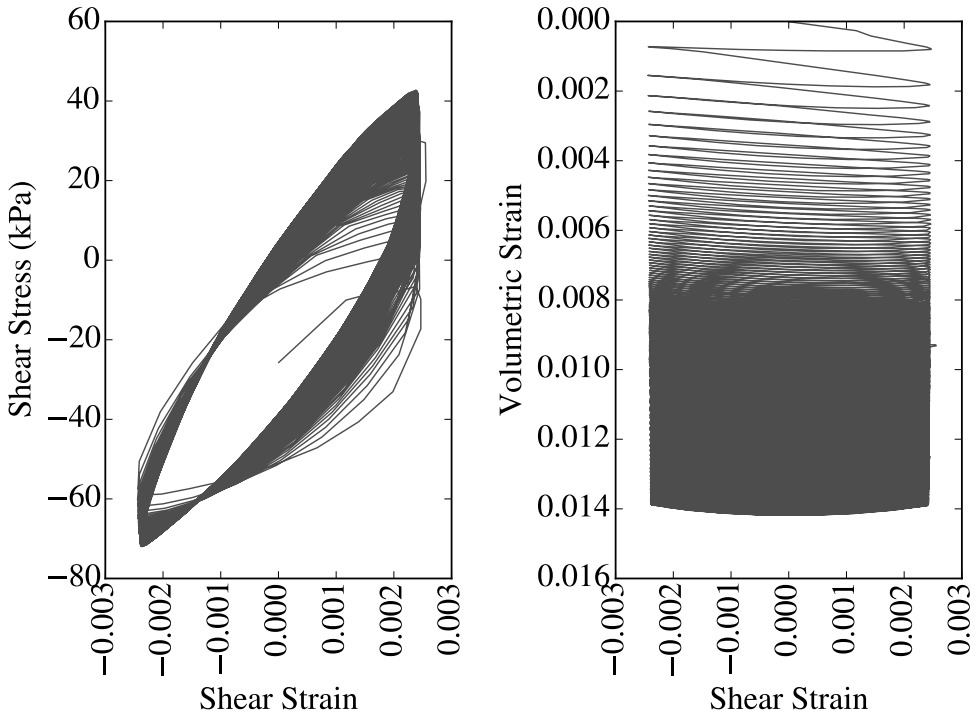


Figure 3.10: Typical results from a constant-amplitude strain-controlled cyclic simple shear test on dry sand.

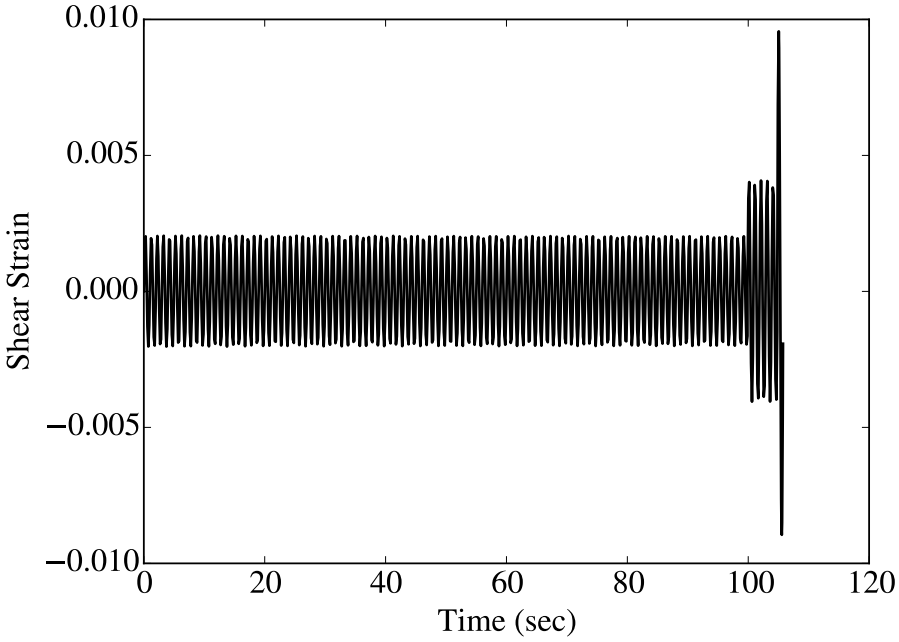


Figure 3.11: Shear strain time history of a variable-amplitude sinusoidal test.

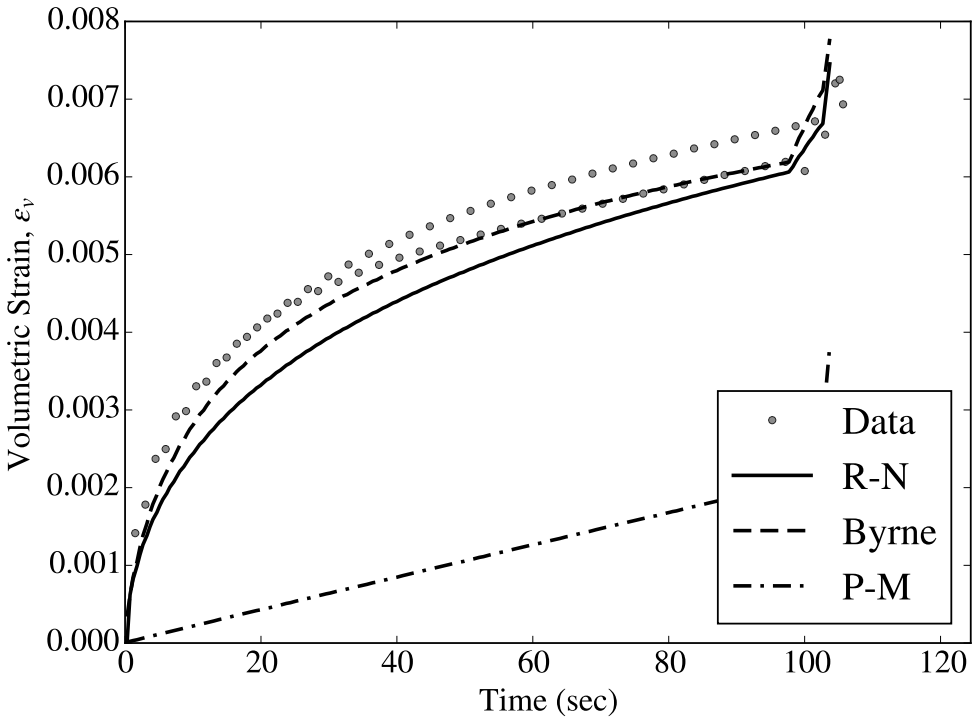


Figure 3.12: Predicted and actual damage paths for a variable-amplitude sinusoidal loading function.

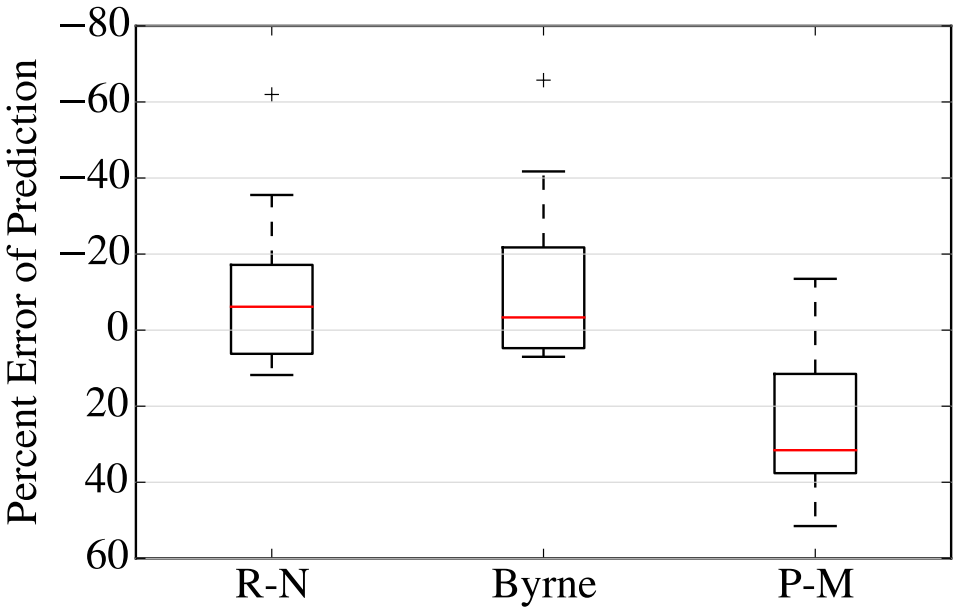


Figure 3.13: Boxplots of the percent error in prediction of volumetric strains for irregular-sinusoid loadings. Negative values signify overprediction.

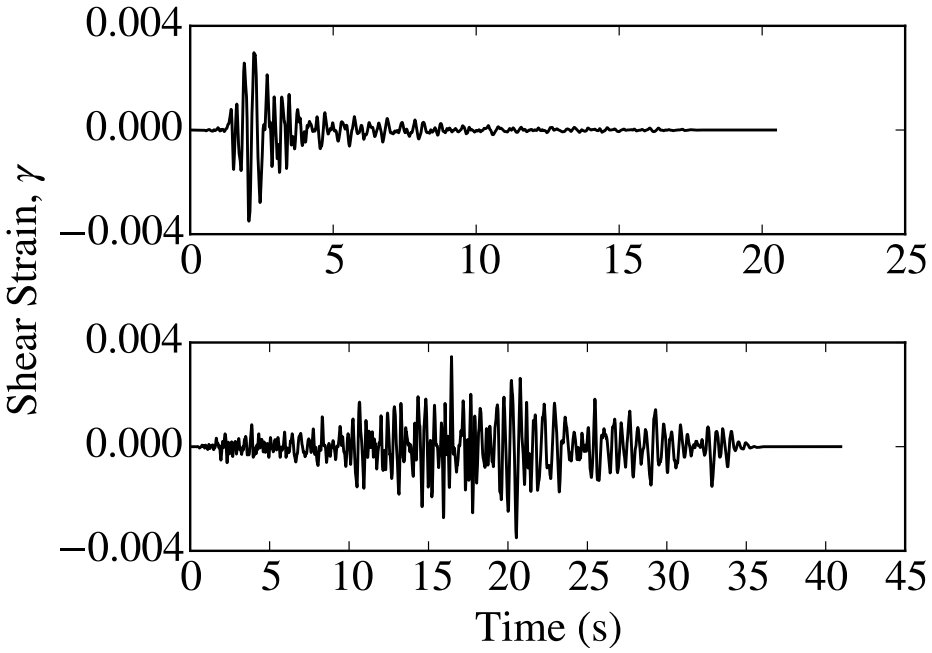


Figure 3.14: At top, a *shock-type*, and, at bottom, a *vibration-type* motion used as the loading function.

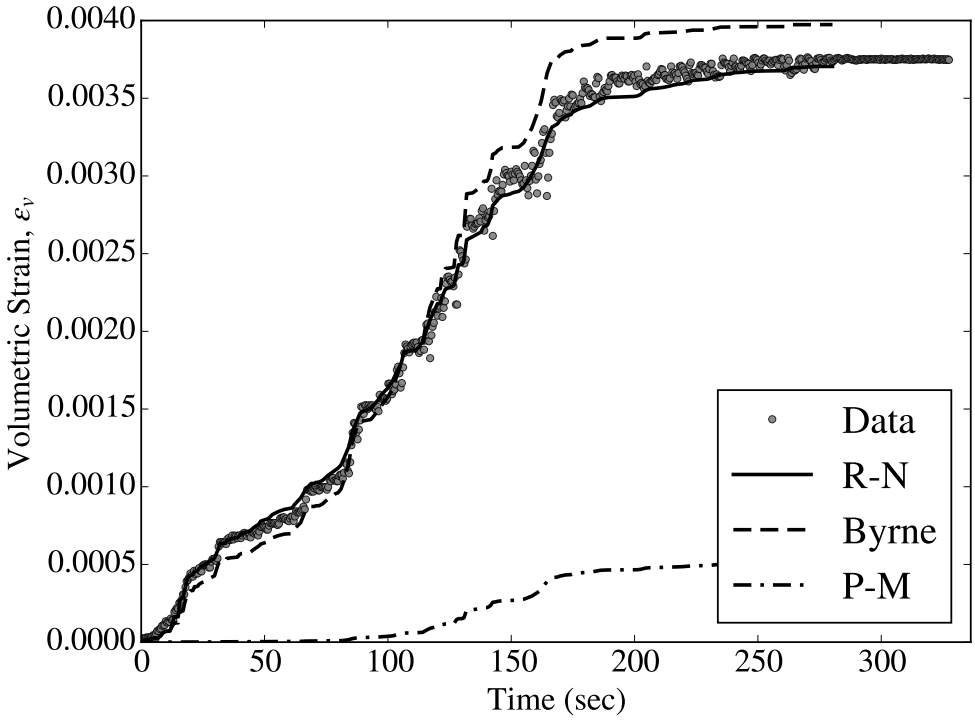


Figure 3.15: Actual and predicted volumetric strains for a *vibration-type* motion.

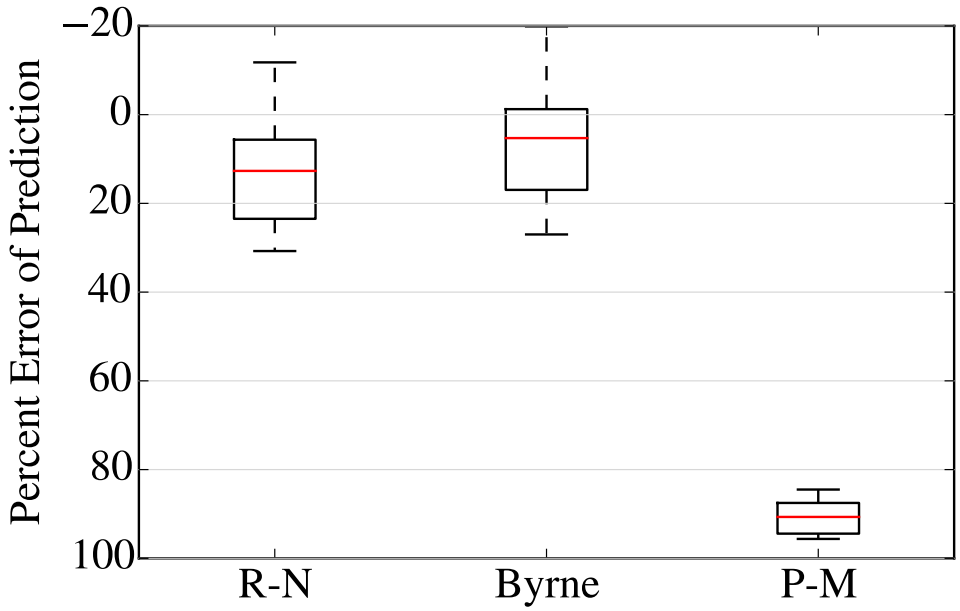


Figure 3.16: Boxplots of the percent error in prediction of volumetric strains for earthquake motion loadings.

Chapter 4

Manuscript #2: Number of Equivalent Cycles for Liquefaction Evaluations in Active Tectonic and Stable Continental Regimes

The authors of the following manuscript intend to submit it to ASCE's *Journal of Geotechnical and Geoenvironmental Engineering*.

Samuel Lasley made the following contributions to this manuscript:

- wrote the entire first draft of the manuscript under the general direction of the other two authors.
- digitized and otherwise prepared the soil profiles from the figures given by Cetin (2000)
- acquired the WUS earthquake ground motions from the PEER database and the CEUS motions from Jongwon Lee.
- wrote ShakeVT2, the equivalent-linear site response program mentioned in the paper
- performed all the equivalent-linear site response analyses
- examined the resulting data and developed functional forms for each regression

- performed all regressions using R and python and selected the best models
- compared the regressed functional forms with those from other researchers
- prepared all original tables and figures

Dr. Green made the following contributions to this manuscript:

- provided the framework for the proposed energy-based liquefaction evaluation procedure and associated equivalent number of cycles correlation (see Green 2000)
- suggested the use of the Cetin (2000) soil profiles and the earthquake motions

Dr. Rodriguez-Marek made the following contributions:

- suggested some functional forms for the r_d models and provided the technical know-how to regress using maximum likelihood
- gave valuable feedback about the soil profiles, site response analyses, regressions

Dr. Green and Dr. Rodriguez-Marek both:

- provided feedback during all stages of the research
- edited the manuscript

Number of Equivalent Cycles for Liquefaction Evaluations in Active Tectonic and Stable Continental Regimes

Samuel J. Lasley[1], Russell A. Green[2], Adrian Rodriguez-Marek[2]

[1] Doctoral Candidate, Department of Civil and Environmental Engineering, Virginia Tech, Blacksburg, Virginia, U.S.A.; slasley@vt.edu

[2] Professor, Department of Civil and Environmental Engineering, Virginia Tech, Blacksburg, Virginia, U.S.A.; rugreen@vt.edu; adrianrm@vt.edu

4.1 Abstract

The number of equivalent cycles (n_{eq}) concept plays an important role in geotechnical earthquake engineering and underlies the accounting for ground motion duration in simplified liquefaction evaluation procedures, whether explicitly or implicitly. In this regard, several n_{eq} correlations have been proposed over the years that were developed using a similar variant of the Palmgren-Miner (P-M) fatigue theory. The correlations presented herein were developed using an alternative implementation of the P-M theory that better accounts for the non-linear response of the soil and for multidirectional shaking. The resulting n_{eq} values computed using this alternative implementation of the P-M theory show a stronger correlation to significant duration of strong motion than values computed using the more commonly used variant of the P-M theory. n_{eq} correlations are proposed herein for shallow crustal earthquakes in both active tectonic and stable continental regimes. Additionally, two forms of the correlations are presented, one being expressed as a function of peak ground acceleration (a_{max}). This relation shows a strong negative correlation between a_{max} and n_{eq}, implying that motions with high amplitudes have short durations and vice versa. This negative correlation is not accounted for in most previously proposed n_{eq} correlations, which could result in the erroneous weighting of the unlikely scenarios of high amplitude-longer duration and low amplitude-short duration motions in liquefaction hazard studies.

4.2 Introduction

Number of equivalent cycles (n_{eq}) correlations for use in liquefaction evaluations are proposed herein that were developed using the alternative implementation of the Palmgren-Miner (P-M) fatigue theory that was proposed by Green and Terri (2005). The n_{eq} concept has long been used to quantify the damaging effects of earthquake motions on soil (e.g., Lee and Chan 1972; Seed et al. 1975; Annaki and Lee 1977). In early liquefaction studies, researchers used the results of cyclic laboratory tests (e.g., cyclic triaxial, cyclic simple shear, shake table) to quantify liquefaction susceptibility. In order to link liquefaction potential to specific earthquake ground motions, it was necessary to either develop laboratory equipment capable of imposing earthquake-type motions (costly) or to develop procedures to convert an earthquake load to an "equivalently-damaging" number of uniform (generally sinusoidal) cycles. This latter approach has the advantages of allowing comparisons between many different earthquake motions and of being relatively easy and cost-effective to implement. While recent liquefaction potential evaluation methods have moved away from the direct use of laboratory testing and towards field-based correlations (e.g., Peck 1979), the n_{eq} concept still underlies the accounting for strong motion duration via magnitude scaling factors (MSF), whether explicitly (e.g., Boulanger and Idriss 2015) or implicitly (e.g., Cetin et al. 2004).

Over the years, several n_{eq} correlations for use in liquefaction evaluations have been proposed (e.g., Seed et al. 1975; Haldar and Tang 1981; Liu et al. 2001; Biondi et al. 2004). With few exceptions (e.g., Green 2001 and Lee 2009), these correlations were developed using the variant of the P-M fatigue theory proposed by Seed et al. (1975), or a slight modification thereof. As detailed in Green and Terri (2005) this approach has shortcomings in accounting for non-linear soil response, as well as not being readily adaptable to account for multidirectional shaking. To overcome these limitations, Green and Terri (2005) proposed an alternative implementation of the P-M theory that is based on the direct equating of dissipated energy in a unit volume of soil subjected to an earthquake motion and to a sinusoidal motion having specified amplitude and equivalent number of cycles. Lee (2009) used this alternative procedure to develop n_{eq} correlations for several simple soil profiles subjected to shallow crustal earthquake motions in both active tectonic and stable continental regimes.

Extending the work of Lee (2009), herein n_{eq} correlations are proposed that were developed using soil profiles representative of those in the liquefaction case history database. Fifty soil profiles characterized in post-earthquake site investigations in California (Cetin 2000) were used in equivalent linear site response analyses to develop the proposed n_{eq} correlations for active shallow crustal tectonic regimes (e.g., western US: WUS). These profiles were subjected to ground motions obtained from the PEER NGA database (Chiou et al. 2008). To develop n_{eq} correlations for stable continental tectonic regimes (e.g., central-eastern US: CEUS), the small strain shear wave velocities (V_S) of the bed rock of the aforementioned profiles were increased and subjected to motions from the McGuire et al. (2001) CEUS database. Additionally, three approaches of increasing rigor were used to account for multidirectional shaking, the simplest of which was used by Lee (2009). Mixed-effects regression analyses were used to develop the proposed correlations, wherein both the similarity in the soil profiles and motions from the same earthquake were considered as random events.

After a brief review of the P-M fatigue theory, overviews of select previous n_{eq} studies are presented; this overview is not all inclusive and is not intended to be. This is followed by a summary of the approach used in this study to develop the proposed n_{eq} correlations. Finally, a discussion and comparison of the approaches used and correlations proposed herein versus those of select previous studies is presented.

4.3 The Palmgren-Miner (P-M) Fatigue Theory

The P-M fatigue theory (Palmgren 1924; Miner 1945), often referred to as "Miner's rule" in the US, is one of several macro-level cumulative damage hypotheses that has been proposed to assess the service life of a material (e.g., a piece of metal) due to fatigue. Detailed overviews of the P-M fatigue theory are presented in numerous papers and books (e.g., Kaechele 1963; Dowling 1972; Collins 1981; Pook 2007), and thus a complete review is not warranted herein. However, a brief summary is given to put into perspective the implementation of the P-M theory used in this study versus that used by others.

In its basic form, the P-M fatigue theory for uniform loading is expressed as (Miner, 1945):

$$\frac{\omega_1}{W} = \frac{n_1}{N_1} \tag{4.1}$$

where: N_1 = number of stress cycles to failure at stress level S_1; n_1 = number of cycles having peak stress amplitude S_1; ω_1 = absorbed work after n_1 cycles; W = absorbed work at failure; and S_1 = peak amplitude of applied cyclic stress. For an applied cyclic stress having varying peak amplitudes (as opposed to a uniform cyclic load), the P-M hypothesis expands to:

$$\frac{1}{W}\sum_{i=1}^{m}\omega_i = \sum_{i=1}^{m}\frac{n_i}{N_i} \tag{4.2}$$

where the load history is comprised of m different peak stress amplitudes (i.e., S_1, S_2,..., S_m); n_i is the number of cycles in the load history having a given peak stress amplitude S_i; ω_i is the absorbed work after n_i cycles; and N_i is number of stress cycles to failure at stress level S_i.

Miner (1945) defined cumulative component damage (D) as the ratio of the absorbed work (ω_i) after n_i cycles and the absorbed work at failure (W):

$$D = \frac{1}{W}\sum_{i=1}^{m}\omega_i \tag{4.3}$$

Thus, per Eqn. 4.2 cumulative damage can also be written as:

$$D = \sum_{i=1}^{m}\frac{n_i}{N_i} \tag{4.4}$$

In the above expressions, $0 \leq D \leq 1$: $D = 0$ corresponds to the damage state of a component that has not been damaged (e.g., never been subjected to cyclic loading); $D = 1$ corresponds to the damage state of a failed component; and $0 < D < 1$ corresponds to the damage state of a component that has been subjected to loading but has not yet failed. Inherent to the above expressions is the assumption that the absorbed work (ω) and the absorbed work at failure (W) are independent of the frequency of the applied load

and, additionally, that W is independent of the amplitude of the applied load. Miner (1945) presents experimental data that reasonably validates these assumptions for aluminum alloy components subjected to thousands of cycles of loading, with failure (i.e., $D = 1$) defined as the initiation of a crack.

A mathematical expression to compute a sinusoidal load having a specified amplitude (S_{ref}) and n_{eq} duration that induces the same amount of damage to a component as a variable amplitude load can be easily derived from Eqn. 4.4. This is done by equating the component damage caused by the variable amplitude load to that induced by a uniform cyclic load that induces failure in N_{ref} cycles:

$$n_{eq} = \sum_i \frac{N_{ref}}{N_i} n_i \tag{4.5}$$

4.4 Previous n_{eq} Studies for Liquefaction Evaluations

4.4.1 Seed et al. (1975)

Seed et al. (1975) noted the analogy between metal fatigue and soil liquefaction and modified the P-M theory for computing n_{eq} for earthquake motions for evaluating liquefaction potential. Per Eqn. 4.5, use of the P-M theory requires a correlation relating the amplitude of a sinusoidal load to the number of cycles required to cause failure in the object being loaded. For metal fatigue and liquefaction evaluations these correlations are experimentally determined and are referred to as S-N and CSR-N_{liq} curves, respectively. CSR (i.e., cyclic stress ratio) is the peak amplitude of the applied cyclic shear stress (τ) divided by the initial vertical effective confining stress (σ'_{v0}), and N_{liq} is the corresponding number of applied cycles required to induce initial liquefaction, where initial liquefaction is commonly defined as either the excess pore pressure becoming equal to σ'_{v0}, or as the induced cyclic strain exceeding a limiting value.

The shape and position of the CSR-N_{liq} curve is highly variable and depends on the soil fabric, soil density, effective confining stress, etc. To alleviate

the need to obtain soil/state specific CSR-N_{liq} curves, Seed et al. (1975) sought to normalize the CSR-N_{liq} curve such that the resulting shape was representative of most sands. The normalization process developed by Seed et al. (1975) is illustrated in Figure 4.1, wherein CSR_1 is the cyclic stress ratio that causes liquefaction in one cycle, FS is a "safety factor", and α is the normalized peak amplitude of the resulting uniform cyclic load. The basis for applying the safety factor (FS) was to account for ground motions that are not severe enough to induce liquefaction. In essence, the use of the FS was an attempt to develop a contour of constant D that is less than 1. The normalized peak amplitude (α) of the resulting uniform cyclic load was selected as 0.65 by Seed et al. (1975). The basis for $\alpha = 0.65$ is likely a remnant of earlier studies where judgment was used to determine the CSR-n_{eq} combination for an earthquake motion (Seed and Idriss 1971; Whitman 1971). As noted in Seed et al. (1975), there is nothing rigorous about setting $\alpha = 0.65$. The normalized curve developed by Seed et al. (1975) using $FS = 1.5$ and $\alpha = 0.65$ is shown in Figure 4.2.

Assuming that ground surface accelerations (a) and corresponding shear stresses (τ) in the upper $\sim$ 6 m (20 ft) of a soil profile are proportional, Seed et al. (1975) used the normalized curve shown in Figure 4.2, in conjunction with Eqn. 4.5, to convert random surface acceleration time histories into equivalently damaging uniform cycles. Inherent to the Seed et al. procedure, the ratio of acceleration and peak ground acceleration (a/a_{max}) is assumed equal to $FS \cdot CSR/CSR_1$. Alternatively stated, a/a_{max} is assumed equal to τ/τ_{max}, where $\tau_{max} = \tau_1/FS$ and τ_1 is the cyclic shear stress required to induce liquefaction in one cycle. However, it can be shown mathematically that for CSR-N_{liq} curves that plot as straight lines on log-log paper, which is the case for most soils, the assumed value of FS does not have any effect on the computed n_{eq} values. The Seed et al. procedure for computing n_{eq} consists of taking a weighted average of the peaks in an acceleration time history, wherein the normalized CSR-N_{liq} curve defines the weighting function. As a result, a normalized CSR-N_{liq} curve, such as that shown in Figure 4.2, is commonly referred to as a weighting factor (WF) curve (e.g., Liu et al. 2001 and Biondi et al. 2004).

There are several different "peak counting" methods (ASTM 2011) that are used in fatigue analyses to obtain the number and amplitudes of peaks in an irregular time history (e.g., an earthquake motions). Seed et al. (1975) used a version of the mean crossing peak counting (also known as zero-crossing)

method wherein the amplitude of one cycle is taken as the maximum value of the time history between successive zero crossings. Because the acceleration pulses or "peaks" in earthquake time histories are rarely symmetrical about the $0g$ axis, a peak above or below the $0g$ axis only represents a half cycle of loading, not a full cycle. To account for this Seed et al. computed the number of equivalent cycles separately for the positive and negative peaks in the time history, with the average of the two being n_{eq} for the record. Also, Seed et al. excluded peaks having amplitudes less than approximately $0.3 \cdot FS \cdot CSR/CSR_1$ (or equivalently $0.3 \cdot \tau/\tau_{max}$) in computing n_{eq} because their contribution was considered to be negligible. To calculate the equivalent number of cycles from two components of motion, Seed et al. suggested two ways without giving a clear preference: (1) Normalize each component of motion by its own a_{max}; and (2) Normalize each motion by the larger a_{max} of the two motions. Seed et al. used both methods to develop n_{eq} correlations from a suite of ~ 60 motions from earthquakes in the WUS and South America. The mean curve obtained by treating each component of motion separately can be approximated as:

$$\ln(n_{eq}) = -1.405 + 0.547M \tag{4.6}$$

where M is the magnitude of the earthquake. It should be noted that this procedure assumes that the ratio a/a_{max} at the surface is equal to CSR/CSR_{max} at depth in the soil profile, which is tenuous as depth increases.

4.4.2 Liu et al. (2001)

Liu et al. (2001) adopted the Seed et al. (1975) variant of the P-M theory to develop an n_{eq} correlation from a ground motion database of 1,528 recordings from 107 earthquake events. All earthquake events were from active shallow crustal regimes; subduction and intraplate events were excluded. However to account for multidirectional shaking, Liu et al. (2001) used the normalized vector sum of horizontal pairs of recordings to calculate n_{eq}. All acceleration values are positive in the normalized vector sum time history and the peak amplitude is equal to 1. Liu et al. (2001) do not explicitly state what peak counting method they used, but it is implied that each reversal in the normalized vector sum time history was treated as a "peak," with an amplitude measured from the zero axis. This approach differs from all

established peak counting methods used in fatigue analyses (ASTM 2011). Liu et al. used two WF curves to compute n_{eq}, referred to as "laboratory-based" and "field-based." Assigning weights of 2/3 and 1/3 to the n_{eq} values computed using these respective curves, Liu et al. developed the following n_{eq} predictive equation:

$$\ln(n_{eq}) = \ln\left[\frac{\left(\frac{\exp(1.53+1.51(M-5.8))}{10^{1.5M+16.05}}\right)^{-\frac{1}{3}}}{4.9 \cdot 10^6 \beta} + 0.75S + 0.095R\right] + \varepsilon \qquad (4.7)$$

where: M is magnitude of the earthquake (moment magnitude preferred), β is the shear wave velocity at the source (generally taken by Liu et al. as 3.2 km/s), S is 0 for rock sites and 1 for soil sites, R is site-to-source distance in kilometers, and ε is a normally distributed error term with a mean of zero and a standard deviation of 0.56.

4.4.3 Biondi et al. (2004)

Biondi et al. (2004) used the Seed et al. (1975) variant of the P-M theory to develop n_{eq} correlations from 1962 motions. They treated each component separately and used the same WF curve (Figure 4.2) used by Seed et al. (1975). Biondi et al. (2004) proposed several regression models; their simplest functional form is:

$$\ln(n_{eq}) = -4.995 - 0.4536 \ln(a_{max}) + 3.204 \ln(M) + \varepsilon \qquad (4.8)$$

where M is the earthquake magnitude (moment magnitude preferred) and ε is the error term with a mean of zero and a standard deviation of 0.68.

4.4.4 Green and Terri (2005)

Thus far, the n_{eq} correlations that have been presented have been developed based on Eqn. 4.3 (via Eqn. 4.5). However, Green and Terri (2005) showed

that an equally valid interpretation of n_{eq} can be obtained from Eqn. 4.4, resulting in:

$$n_{eq} = \frac{\sum \omega_i}{\omega_{ref}} \tag{4.9}$$

where $\sum \omega_i$ is the total absorbed work (or dissipated energy) in a unit volume of soil subjected to an earthquake motion, and ω_{ref} is the energy dissipated in a unit volume of soil subjected to one equivalent cycle of loading having a reference amplitude τ_{ref}. The total dissipated energy is simply the cumulative area bound by stress-strain hysteresis loops. For vertically propagating shear waves in a soil profile, $\sum \omega_i$ can be calculated from shear stress and strain time histories at depth in the profile using the trapezoidal rule:

$$\sum \omega_i = \frac{1}{2} \sum_{j=1}^{k-1} (\tau_{j+1} + \tau_j)(\gamma_{j+1} - \gamma_j) \tag{4.10}$$

where k is the total number of time steps in the time histories, and τ_j and γ_j are the shear stress and shear strain at time step j, respectively.

The denominator of Eqn. 4.9, ω_{ref}, can be related to the soil's equivalent viscous damping ratio (Jacobsen 1960; Green and Terri 2005):

$$\omega_{ref} = \frac{2\pi D_\gamma \tau_{ref}^2}{G_\gamma} \tag{4.11}$$

where G_γ and D_γ are the degraded shear modulus and damping ratio at a shear strain corresponding to τ_{ref}, defined consistently with CSR from the simplified liquefaction evaluation procedure as $0.65 \cdot \tau_{max}$. Substituting Eqn. 4.11 into Eqn. 4.9 yields:

$$n_{eq} = \frac{\sum \omega_i \, G_\gamma}{2\pi \, D_\gamma \, \tau_{ref}^2} \tag{4.12}$$

As discussed in Boulanger and Idriss (2015), n_{eq} is a function of both the earthquake loading and the soil response, with the latter not fully accounted for in previous studies. However by computing n_{eq} from both the applied τ and the induced γ, both the loading and soil response are fully and rigorously

taken into account in Eqn. 4.12. Additionally, both the WF curve and the cycle counting method are inherent to this alternative implementation of the P-M theory.

Green and Terri (2005) performed a parametric study of n_{eq} using 326 ground motion recordings from 29 different earthquakes. These motions were applied to a soil profile consisting of loose sand using the site response code SUMDES (Li et al. 1992). The results of paired horizontal components of motion were combined by summing the dissipated energies from each component. Also, ω_{ref} was computed for an equivalent cycle having an amplitude corresponding to 0.65 times the geometric mean of the maximum amplitudes of the pair of horizontal components of motion:

$$\tau_{\text{ref}} = 0.65 \cdot \sqrt{\max(|\tau_1|) \cdot \max(|\tau_2|)} \tag{4.13}$$

where $\max(|\tau_1|)$ and $\max(|\tau_2|)$ are the peak or maximum absolute values of the two components of shear stress induced in the soil profile at a given depth. Although they did not propose a functional form for n_{eq}, they showed that n_{eq} could vary as a function of depth in a soil profile and increases with site-to-source distance as well as magnitude, supporting the trend shown by Liu et al. (2001).

4.4.5 Lee (2009)

Using the alternative approach to implement the P-M theory proposed by Green and Terri (2005), Lee (2009) performed equivalent linear site response analyses on 6 simple soil profiles to develop n_{eq} correlations. All soil profiles were composed of 12 clean sand layers with the bedrock located at a depth of 30 m. The distinguishing feature between the soil profiles was the variation of small strain shear wave velocity (V_S) with depth. They ranged from a profile with a constant velocity of 139 m/s for all depths to a profile with shear wave velocities ranging from 75 m/s in the surface layer to 225 m/s in the deepest soil layer. The earthquake ground motions used in the site response analyses were obtained from two databases compiled by McGuire et al. (2001): one for the WUS (296 motions) and another for the CEUS (270 motions). The same soil profiles were used in the site response analyses for both the WUS and CEUS motions, but Lee (2009) increased the V_S of

the bedrock by a factor of ~ 3.8 for the CEUS analyses. Each motion was treated separately and used to compute n_{eq} at several depths in the profiles.

The functional form of Lee's n_{eq} correlation is:

$$\ln(n_{eq}) = \exp(C_1 z) + C_2 R^{C_3} + C_4 M_w + C_5 + \varepsilon \tag{4.14}$$

where z is depth in meters, M_w is moment magnitude, R is site-to-source distance in km, defined as the closest distance to the fault rupture plane, C_1-C_5 are regression coefficients, and ε is an error term. Lee provided regression coefficients separately for each profile and each ground motion database (i.e., WUS and CEUS). Table 1 lists the regression coefficients and the standard deviation of ε for one of the soil profiles considered by Lee (2009).

4.5 Approach Used to Develop Proposed Correlations

This study builds upon those of Green and Terri (2005) and Lee (2009), but uses different ground motion databases, soil profiles that are representative of those in the earthquake liquefaction case history databases, and uses three different approaches of increasing rigor to account for multidirectional shaking.

4.5.1 Earthquake Databases

Equivalent linear site response analyses were performed using SHAKEVT2 (Lasley et al. 2014), a rewrite of SHAKE and SHAKE91 (Schnabel et al. 1971; Idriss and Sun 1992), wherein the Darendeli and Stokoe (2001) shear modulus reduction and damping curves were used to model the non-linear response of the soil. The earthquake motions used in the analyses came from two different earthquake databases. The first database is the PEER NGA database (Chiou et al. 2008), from which 194 pairs of rock components from 47 different earthquakes were obtained. The average small strain shear wave velocities of all the upper 30 m (V_{S30}) of the sites where the motions were recorded are greater than 650 m/s. The moment magnitudes (M_w) of the earthquake

motions range from 5.27 to 7.62, and site-to-source distances (R) range from 1.8 to 152 km. These shallow crustal earthquake motions are similar to those experienced in much of the WUS and are henceforth referred to as "WUS motions."

The second ground motion database used in this study is the McGuire et al. (2001) CEUS database. Because few CEUS earthquake motions have been recorded, the majority of the 149 pairs of motions in the CEUS database are scaled WUS motions. The M_w of these motions range from 4.5 to 7.6 and R range from 0.1 to 199 km. The recorded motions include the 1988 Saguenay (M_w5.9 mainshock and M_w4.5 aftershock), the 1985 Nahanni (M_w6.8), and the 1989 New Madrid, MO (M_w4.7) earthquakes.

McGuire et al. (2001) scaled WUS motions for CEUS conditions using response spectral transfer functions generated from the single-corner frequency point source model in conjunction with random vibration theory (RVT) (e.g., Brune 1970; 1971; Boore 1983; McGuire et al. 2001; Silva and Lee 1987). The transfer functions account for the differences in seismic source, wave propagation path properties, and generic site effects between the WUS and CEUS regions. Many seismological publications have shown successful results of the RVT point source model for generating strong ground motions for both WUS and CEUS (Boore 1983; 1986; Hanks and McGuire 1981; McGuire et al. 1984; Schneider et al. 1993; Silva 1993). In generating the scaled CEUS motions, recorded WUS motions were used as "seed" motions in the spectral scaling process, resulting in scaled motions that have realistic characteristics. In this context, the stochastic point source model is a reliable and reasonable approach for estimating spectral characteristics of strong ground motions for engineering analyses. The scaling method however, should be validated as additional recordings of stable continental motions become available.

Tables 4.10 and 4.11 in the electronic supplement list the WUS and CEUS motions, respectively, used in the analyses. Also, Figure 4.3 shows the distribution of earthquake events as a function of magnitude and distance for both databases.

4.5.2 Soil Profiles

The soil profiles used in this study were compiled by Cetin (2000). The 50 well-characterized profiles are from post-earthquake site investigations in

California. For each profile, Cetin provides the qualitative description of the layers (e.g. clay, fine sand) and the shear wave velocities. Additional soil layer details (unit weight, plasticity index, at-rest lateral earth pressure coefficient, and friction angle) were randomly selected from a distribution of probable values corresponding to the qualitative description. The profiles were analyzed as-is with the WUS motions. For simulated conditions representative of those in the CEUS, the shear wave velocity of the bedrock layers of each profile was increased by a factor 3.5 and analyzed using the CEUS input motions. This factor of 3.5 brought the bedrock shear wave velocities into agreement with higher bedrock shear wave velocities of the CEUS. Several of the profiles had similar characteristics; Figures 4.4 and 4.5 show the grouped profiles for the WUS and CEUS, respectively.

4.5.3 Multidirectional Shaking

As discussed above, several different approaches have been used to account for multidirectional shaking. Specific to the alternative implementation of the P-M theory, Lee (2009) treated each motion separately, implying that multiplying n_{eq} by 2 will account for multidirectional shaking (e.g., Pyke et al., 1975). Henceforth, this approach is referred to as Approach 1. In contrast, Green and Terri (2005) accounted for multidirectional shaking by performing separate site response analyses for each horizontal component in a pair of motions, adding the energy dissipated at the respective depths for each component of motion, and setting the amplitude of the equivalent cycle as the 0.65 times the geometric mean of the maximum shear stresses experienced at a given depth. Henceforth, this approach is referred to as Approach 2. In addition to these two approaches, a third approach was also used in this study (i.e., Approach 3), summarized as follows.

1. Separately, each of the paired components was applied at the base of a given soil profile using the equivalent-linear algorithm.
2. Once both analyses converged, the degraded profiles were compared and a composite profile was constructed by using the softer layer properties of the two profiles.
3. The site responses analyses were performed again, holding the soil properties constant, and the energy dissipated at the respective depths in

the profile subjected to the two components of motion were summed.

4. The amplitude of the equivalent cycle was set as 0.65 times the geometric mean of the maximum shear stresses experienced at a given depth.

5. Eqn. 4.12 was used to compute n_{eq}.

The rationale for this approach is that in the field, it is likely that the strain induced in one direction of motion will influence the soil stiffness in other directions.

4.5.4 Regression Analysis

Regression was performed using the R (2013) package `lme4` (Bates et al. 2014), which implements a mixed effects regression. The average event residual and average profile group residuals are assumed to be random effects. The use of mixed effects regression avoids potential biases from earthquakes or profile groups that have a larger number of data points. For the CEUS dataset, the random effect of earthquake event was not included in the final regressions, rather only the profile group was included, because the earthquake event could not be properly constrained by the data. Earthquake faulting mechanism was also considered as a random effect, but it was found to be insignificant for both datasets and was not used in the final regression analyses. A bootstrapping technique (Efron and Tibshirani 1994) was employed to obtain the mean and standard deviation of all regression coefficients. It was done in the following way:

1. Ten thousand data points were randomly selected (without replacement) from the dataset of interest.

2. The regression coefficients that best fit the 10,000 data points for the functional form of interest were obtained.

3. Steps 1 and 2 were repeated for 1000 iterations and the regression coefficients for each iteration were recorded.

4. The mean and standard deviation of the distribution of each regression coefficient were calculated.

4.5.5 Proposed n_{eq} Correlations

n_{eq} correlations were developed for each of the three approaches discussed above to account for multidirectional shaking. Several functional forms of the regression equations were examined for the correlations, including the form used by Lee (2009). Of these, two functional forms were chosen because of their superior fit and the convenience of their predictive variables. As shown in Figures 4.6 and 4.7, n_{eq} computed using Approach 1 (i.e., $n_{eq,\mathrm{A1}}$) for 0 to 4 m depths is strongly correlated with the peak ground acceleration (a_{max}) at the ground surface; this same trend was identified for the n_{eq} values computed using the other two approaches. Thus, the first functional form of the proposed correlation includes a_{max} as a predictor variable:

$$\ln(n_{eq}) = a_1 + a_2 \ln(a_{max}) + a_3 M_w + \varepsilon_1 \tag{4.15}$$

where a_{max} is in units of g, M_w is the moment magnitude of the earthquake, a_1-a_3 are regression coefficients, and ε_1 is an error term which is assumed to be normally distributed. Table 4.2 lists the mean values of the regression coefficients for both the WUS and CEUS datasets, and Table 4.3 lists the uncertainty for each component as determined by the standard deviations of each coefficient computed using the bootstrapping technique. The low value of the standard deviations indicates that both the M_w and a_{max} scaling are well constrained by the data. Table 4.4 summarizes the uncertainties in the correlation where τ_{event} and τ_{profile} are the standard deviations of the earthquake event and soil profile random effects, respectively, and σ_ε is the standard deviation of the residual. If these standard deviations are assumed to be independent, the total uncertainty is:

$$\sigma_{\mathrm{Total}} = \sqrt{\tau_{\mathrm{event}}^2 + \tau_{\mathrm{profile}}^2 + {\sigma_\varepsilon}^2} \tag{4.16}$$

As shown in Table 4.4, Approach 2 provides the smallest total error for both earthquake databases, and overall, the WUS database has smaller errors than the CEUS database.

The functional form of this correlation lends itself for use with simplified liquefaction procedures because they require both the magnitude (for MSF) and a_{max} as input variables. It was expected that the V_S of the layer would

be a significant predictor variable, but it was found not to be significant when used in conjunction with a_{max}.

The second functional form for the proposed n_{eq} correlation is more in line with that used by others (e.g., Liu et al. 2001 and Lee 2009) in that it expresses n_{eq} as a function of M_w and R. This allows for better comparison with previously proposed n_{eq} correlations and is useful when a_{max} is not known:

$$\ln(n_{eq}) = b_1 + b_2 M_w + b_3 \ln(R) + \varepsilon_2 \tag{4.17}$$

where M_w and R are defined the same as in Eqn. 4.14, b_1-b_5 are regression coefficients, and ε_2 is a normally-distributed error term. Tables 4.5 and 4.6 give the mean and standard deviations of each regression coefficient, respectively. Table 4.7 provides the inter- and intra-event errors as well as the residual and total errors. Compared to Eqn. 4.15, the total errors are slightly less for the WUS dataset and slightly greater for the CEUS dataset. Once again, Approach 2 provides the smallest total errors.

The standard deviations of total error are of a similar magnitude as some of the other studies, namely, Lee (2009), Liu et al. (2001), and Biondi et al. (2004). Figure 4.8 shows a plot of the two functional forms for the n_{eq} correlations (i.e., Eqns. 4.15 and 4.17) for the WUS and CEUS ground motion datasets and for all three approaches used to account for multidirectional shaking. For the WUS dataset, Approaches 1 and 2 predict very similar values of n_{eq}; Approach 3, however, predicts greater values of n_{eq}, especially at medium to large values of a_{max} and most site-to-source distances. For the CEUS, however, the $n_{eq,\mathrm{A3}}$ values have a weaker dependence on M_w and predict larger n_{eq} values at low magnitudes. For all approaches, the CEUS correlations predict higher values of n_{eq} at low a_{max} but lower values of n_{eq} at longer site-to-source distance when compared to the WUS correlations.

4.6 Comparison of n_{eq} Correlations

4.6.1 Green and Terri (2005) vs. Seed et al. (1975) Approaches

As discussed in Green and Terri (2005) and summarized above, the alternative implementation of the P-M theory that they propose and the more commonly used implementation proposed by Seed et al. (1975) are both underlain by the original P-M theory. To illustrate this, n_{eq} values computed using the two implementations are compared in Figures 4.9 and 4.10. In calculating the n_{eq} values per Seed et al. (1975), both the Seed et al. (1975) and the Liu et al. (2001) variants were used. However, for consistency with the Darendeli and Stokoe (2001) shear modulus and damping degradation curves used in the site response analyses performed to compute n_{eq} per Green and Terri (2005), the WF curve used in conjunction with Seed et al. (1975) and Liu et al. (2001) plots as a straight line on log-log paper and had a slope of -0.34. In addition to being consistent with the Darendeli and Stokoe (2001) shear modulus and damping degradation curves for the densities and confining stresses of interest, this assumed curve is consistent with laboratory curves developed from high-quality undisturbed samples obtained by freezing (Yoshimi et al. 1984).

To allow for as consistent of a comparison as possible, n_{eq} values shown in Figures 4.9 and 4.10 computed using Green and Terri (2005) are for a shallow depth in the profiles (0 to 4 m), while the n_{eq} values computed using Seed et al. (1975) and Liu et al. (2001) were computed from the acceleration time histories at the profile's surface output by SHAKEVT2. Also, consistent with how each of the methods accounts for multidirectional shaking, the n_{eq} values computed using Seed et al. (1975) (i.e., $n_{eq,\mathrm{Sea75}}$) are compared with n_{eq} values computed using Approach 1 (i.e., $n_{eq,\mathrm{A1}}$), while the n_{eq} values computed using Liu et al. (2001)(i.e., $n_{eq,\mathrm{Lea01}}$) are compared with n_{eq} values computed using Approaches 2 and 3 (i.e., $n_{eq,\mathrm{A2}}$ and $n_{eq,\mathrm{A3}}$), respectively.

As may be observed from Figures 4.9 and 4.10, there is almost a one to one correlation between $n_{eq,\mathrm{Sea75}}$ and $n_{eq,\mathrm{A1}}$, as indicated by Pearson's r values of 0.95 and 0.92 for WUS and CEUS motions, respectively. Again, this is not surprising because both procedures are underlain by the P-M theory (i.e., Eqn. 4.4 vs Eqn. 4.3). However, this one to one correlation becomes tenuous

at deeper depths in the profiles because of the inherent assumption in Seed et al. (1975) that the ground surface accelerations are directly proportional to shear stresses in the upper $\sim$ 6 m (20 ft) of the soil profiles. Also from Figures 4.9 and 4.10, it may be observed that the $n_{eq,\text{Lea01}}$ values have a strong correlation with the $n_{eq,\text{A2}}$ and $n_{eq,\text{A3}}$ values, as indicated by Pearson's r values of 0.77 and 0.81 for WUS and CEUS motions for $n_{eq,\text{Lea01}}$ versus $n_{eq,\text{A2}}$ and 0.83 and 0.82 for WUS and CEUS motions for $n_{eq,\text{Lea01}}$ versus $n_{eq,\text{A3}}$. However, the $n_{eq,\text{Lea01}}$ values tend to be systematically less than both the $n_{eq,\text{A2}}$ and $n_{eq,\text{A3}}$ values. This is attributed to Liu et al. (2001) using the normalized vector sum approach to account for multidirectional shaking, which is not conducive to using any of the established fatigue peak counting methods.

4.6.2 Correlation with Other Ground Motion Duration Metrics

Although its units are number of cycles as opposed to units of time, n_{eq} is a ground motion duration metric. Accordingly, n_{eq} should correlate with other ground motion duration metrics. Towards this end, Figures 4.11 and 4.12 show plots of $n_{eq,\text{A1}}$ for 0 to 4 m depths versus bracketed duration ($\pm 0.05g$ threshold), 5-95% strong motion duration, and 5-75% strong motion duration. As may be observed from these figures, $n_{eq,\text{A1}}$ has the strongest correlation with 5-75% strong motion duration. This is not surprising because 5-75% strong motion duration has been recognized as the duration metric that can be predicted with the least uncertainty (Abrahamson and Silva 1996). In fact, the functional form of the Liu et al. (2001) n_{eq} prediction equation (Eqn. 4.7) was adopted from the Abrahamson and Silva (1996) prediction equation for 5-75% strong motion duration.

To compare the strengths of the $n_{eq,\text{A1}}$, $n_{eq,\text{A2}}$, and $n_{eq,\text{A3}}$ correlations with other duration metrics to the strengths of correlations among $n_{eq,\text{Sea75}}$ and $n_{eq,\text{Lea01}}$ and the same duration metrics, the Pearson correlation coefficient, r, for each correlation is presented in Tables 4.8 and 4.9 for the WUS and CEUS, respectively. However, direct correlations between $n_{eq,\text{A2}}$, $n_{eq,\text{A3}}$, and $n_{eq,\text{Lea01}}$ and the other duration metrics is not possible. This is because Approaches 2 and 3 and Liu et al. (2001) yield a single, combined n_{eq} value for both components of motion, while the other duration metrics yield duration values

for each component of acceleration at the profile's surface. To allow for a comparison, $n_{eq,\mathrm{A2}}$, $n_{eq,\mathrm{A3}}$, and $n_{eq,\mathrm{Lea01}}$ were compared to the geometric mean of the time durations of the two components of motion. As may be observed from Tables 4.8 and 4.9, $n_{eq,\mathrm{A1}}$, $n_{eq,\mathrm{A2}}$, and $n_{eq,\mathrm{A3}}$ have stronger correlations to the other duration metrics than do $n_{eq,\mathrm{Sea75}}$ and $n_{eq,\mathrm{Lea01}}$, for every case except bracketed duration of WUS motions.

4.6.3 Comparison of n_{eq} Predictive Equations

Figure 4.13 shows a comparison of various n_{eq} correlations for an M_w5.5 earthquake where Eqn. 4.15 was used to compute $n_{eq,\mathrm{A1}}$. In this and the subsequent figure, the curve labeled as Lee (2009) is for WUS. Also, the predicted values of $n_{eq,\mathrm{Lea01}}$ are divided by 2 to allow better comparison with the other correlations. As may be observed in this figure, the correlation for $n_{eq,\mathrm{A1}}$ for the WUS compares well with the $n_{eq,\mathrm{Lea01}}$ correlation at short and long distances, but not at intermediate distances. Lee (2009) and the correlation for $n_{eq,\mathrm{A1}}$ for the WUS give similar results for site-to-source distances less than about 100 km but diverge some at distances greater than 100 km. The correlation for $n_{eq,\mathrm{Sea75}}$ gives similar results as the other WUS relations for short distances, but significantly underpredicts n_{eq} at longer distances. Finally, $n_{eq,\mathrm{A1}}$ for the CEUS is less sensitive to site-to-source distance than the comparable correlation for the WUS; it predicts larger n_{eq} at short distances and fewer n_{eq} at longer distances relative to the WUS relation.

Figure 4.14 is similar to Figure 4.13, but for an M_w7.5 earthquake. As may be observed in this figure, Lee (2009) and the correlation for $n_{eq,\mathrm{A1}}$ for the WUS give similar results for site-to-source distances less than about 100 km but diverge some at distances greater than 100 km, the same trend identified for an M_w5.5 event. The correlation for $n_{eq,\mathrm{Lea01}}$ predicts larger n_{eq} than the correlation proposed herein for the WUS at distances less than $\sim$ 50 km, but the two relations predict very similar n_{eq} values at greater distances. The correlation for $n_{eq,\mathrm{Sea75}}$ predicts a larger n_{eq} than the correlation proposed herein for the WUS at distances less than $\sim$ 75 km, but underpredicts n_{eq} at longer distances. Finally, similar to the trends identified for a M_w5.5 event, $n_{eq,\mathrm{A1}}$ for the CEUS is less sensitive to site-to-source distance than the comparable correlation for the WUS; again it predicts larger n_{eq} at short distances and fewer n_{eq} at longer distances relative to the WUS relation.

Because the Biondi et al. (2004) correlation (Eqn. 4.8) and Eqn. 4.15 both have a_{max} as a predictor variable, they are compared in Figure 4.15. Both correlations show a trend of decreasing n_{eq} with increasing a_{max}, and Biondi et al. (2004) predicts similar n_{eq} values for a M_w7.5 earthquake as Eqn. 4.15 for WUS. However for M_w5.5, Biondi et al. (2004) predicts smaller n_{eq} than Eqn. 4.15 for the WUS. Eqn. 4.15 for the CEUS is less sensitive to M_w than it is for the WUS and predicts larger n_{eq} for a_{max} less than $\sim 0.8g$ for the WUS and CEUS, with the predicted n_{eq} values for both regimes tending to converge for larger values of a_{max}.

4.7 Summary and Conclusions

The use of number of equivalent cycle (n_{eq}) concept in liquefaction evaluations extends from the late 1960s to the present day. It forms an important part of the simplified liquefaction evaluation procedure by accounting for ground motion duration, whether explicitly or implicitly. The majority of the proposed n_{eq} correlations were developed using the Seed et al. (1975) variant of the Palmgren-Miner (P-M) fatigue theory (e.g., Seed et al. 1975; Haldar and Tang 1981; Liu et al. 2001; Biondi et al. 2004). However, Green and Terri (2005) detailed shortcomings of this variant and proposed an alternative implementation of the P-M theory that is based on the direct equating of dissipated energy in a unit volume of soil subjected to an earthquake motion and to a sinusoidal motion having specified amplitude and equivalent number of cycles.

This study builds upon those of Green and Terri (2005) and Lee (2009), but uses different ground motion databases, soil profiles that are representative of those in the earthquake liquefaction case history databases, and uses three different approaches of increasing rigor to account for multidirectional shaking. Mixed effects regression analyses were used to develop the two proposed correlations, wherein both the similarity in the soil profiles and motions from the same earthquake were considered as random events. The first correlation (Eqn. 4.15) has a_{max} and M_w as predictor variables and the second (Eqn. 4.17) has M_w and R as predictor variables. Eqn. 4.15 lends itself for use with simplified liquefaction procedures because both M_w and a_{max} are already required as input variables. Also, similar to the correlation proposed by Biondi et al. (2004), Eqn. 4.15 shows a strong negative correlation be-

tween n_{eq} and a_{max}. This is not accounted for in most previously proposed n_{eq} correlations, which could result in the erroneous weighting of the unlikely scenarios of high amplitude-longer duration and low amplitude-short duration motions in liquefaction hazard studies. The proposed correlations are applicable for site-to-source distances up to 150 km for the WUS and up to 200 km for CEUS and for earthquakes with magnitudes ranging from 5.3 to 7.6 for the WUS and 4.5 to 7.6 for CEUS.

4.8 Acknowledgments

This research is partially funded by National Science Foundation (NSF) grants CMMI-1030564 and CMMI-1306261. This support is gratefully acknowledged. However, any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

4.9 References

Abrahamson, N.A. and Silva, W.J. (1996). "Empirical ground motion models." *Rep.* Brookhaven National Laboratory, Lipton, NY.

Annaki, M., and Lee, K. L. (1977). "Equivalent uniform cycle concept for soil dynamics." *Journal of the Geotechnical Engineering Division*, 103(GT6), 549–564.

ASTM Standard E1049, 1985 (2011). "Practices for Cycle Counting in Fatigue Analysis." *ASTM International*, West Conshohocken, PA, www.astm.org

Bates D., Maechler M., Bolker B. and Walker S. (2014). "lme4: Linear mixed-effects models using Eigen and S4." R package version 1.1-7, .

Biondi, G., Cascone, E., and Maugeri, M. (2004). "Number of uniform stress cycles equivalent to seismic loading." Proceedings of 11th ICSDEE/3rd ICEGE, 705–712.

Boore, D.M. (1983). "Stochastic simulation of high-frequency ground motions based on seismological models of the radiated spectra." *B. Seismol.*

Soc. Am., 73(6A), 1865-1894.

Boore, D.M. (1986). "Short-period P- and S-wave radiation from large earthquakes: implications for spectral scaling relations", *B. Seismol. Soc. Am.*, 76(1), 43-64.

Boulanger, R. W., and Idriss, I. M. (2015). "Magnitude scaling factors in liquefaction triggering procedures." *Soil Dynamics and Earthquake Engineering*, 10.1016/j.soildyn.2015.01.004.

Brune, J.N. (1970). "Tectonic stress and spectra of seismic shear waves from earthquakes", *J. Geophys. Res.*, 75(26), 611-614.

Brune, J.N. (1971). "Correction", *J. Geophys. Res.*, 76(20), 1441-1450.

Cetin, K. O. (2000). "Reliability-based assessment of seismic soil liquefaction initiation hazard." Ph.D., University of California, Berkeley.

Cetin, K. O., Seed, R. B., Kiureghian, A. Der, Tokimatsu, K., Harder Jr, L. F., Kayen, R. E., and Moss, R. E. (2004). "Standard penetration test-based probabilistic and deterministic assessment of seismic soil liquefaction potential." *Journal of Geotechnical and Geoenvironmental Engineering*, 130(12), 1314–1340.

Chiou, B., Darragh, R., Gregor, N., and Silva, W. (2008). "NGA Project Strong-Motion Database." *Earthquake Spectra*, 24(1), 23–44.

Collins, J.A. (1981). *Failure of Materials in Mechanical Design: Analysis, Prediction, Prevention.* John Wiley & Sons, New York.

Darendeli, M. B., and Stokoe, K. H. (2001). "Development of a new family of normalized modulus reduction and material damping curves." Geotechnical Engineering Report GD01-1, University of Texas at Austin.

De Alba, P., Chan, C. K., and Seed, H. B. (1975). "Determination of soil liquefaction characteristics by large-scale laboratory tests, EERC 75-14." Earthquake Engineering Research Center, University of California, Berkeley, CA.

Dowling, N.E. (1972). "Fatigue Failure Predictions for Complicated Stress-Strain Histories." *J. Materials*, 7(1), 71-78.

Efron, B., & Tibshirani, R. J. (1994). *An Introduction to the Bootstrap (Vol. 57).* CRC Press Chicago.

Green, R. A. (2001). "Energy-Based Evaluation and Remediation of Liquefiable Soils." Ph.D., Virginia Polytechnic Institute and State University, Blacksburg, Virginia.

Green, R. A., and Terri, G. A. (2005). "Number of equivalent cycles concept for liquefaction evaluations—Revisited." *Journal of Geotechnical and Geoenvironmental Engineering*, 131(4), 477–488.

Haldar, A., and Tang, W. H. (1981). "Statistical study of uniform cycles in earthquakes." *Journal of the Geotechnical Engineering Division*, 107(5), 577–589.

Hanks, T.C. and McGuire, R.K. (1981). "The character of high-frequency strong ground motion." *B. Seismol. Soc. Am.*, 71(6), 2071-2095.

Idriss, I. M., and Sun, J. I. (1992). "User's manual for SHAKE91: A computer program for conducting equivalent linear seismic response analyses of horizontally layered soil deposits." Center for Geotechnical modeling Department of Civil & Environmental Engineering, University of California.

Jacobsen, L.S. (1960). "Damping in composite structures." *Proceedings of the Second World Conference on Earthquake Engineering*, 2, 1028-1044.

Kaechele, L. (1963). *Review and analysis of cumulative-fatigue-damage theories.* DTIC Document.

Lasley, S. J., Green, R. A., and Rodriguez-Marek, A. (2014). "Comparison of equivalent-linear site response analysis software." *Proceedings of the 10th U.S. National Conference on Earthquake Engineering*, Earthquake Engineering Research Institute, Anchorage, AK.

Lee, K. L., and Chan, K. (1972). "Number of equivalent significant cycles in strong motion earthquakes." *Proceedings of the International Conference on Microzonation For Safer Construction Research and Application*, Seattle, Washington, 609–627.

Lee, J. (2009). "Engineering characterization of earthquake ground motions." PhD Dissertation, University of Michigan, Ann Arbor, MI.

Li, X. S., Z. L. Wang, and C. K. Shen (1992). "SUMDES: a nonlinear procedure for response analysis of horizontally-layered sites subjected to multidirectional earthquake loading," Dept. of Civil Engineering, Univ. of California, Davis.

Liu, A. H., Stewart, J. P., Abrahamson, N. A., and Moriwaki, Y. (2001). "Equivalent number of uniform stress cycles for soil liquefaction analysis." *Journal of Geotechnical and Geoenvironmental Engineering*, 127(12), 1017–1026.

McGuire, R.K., Becker, A.M. and Donovan, N.C. (1984). "Spectral estimates of seismic shear waves", *Bull. Seismol. Soc. Am.*, 74(4), 1427-1440.

McGuire, R. K., Silva, W. J., and Costantino, C. J. (2001). *Technical Basis for Revision of Regulatory Guidance on Design Ground Motions: Hazard- and Risk-consistent Ground Motion Spectra Guidelines.* Division of Engineering Technology, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, Washington, D.C.

Masing, G.(1926). "Eigenspannungen und verfertigung beim messing." *Proc., 2nd Int. Congress on Applied Mechanics*, Zurich, Switzerland.

Miner, M. A. (1945). "Cumulative damage in fatigue." *Journal of Applied Mechanics*, 12(3), A159-A164.

Palmgren, A. (1924). "Die lebensdauer von kugellagern (Life length of roller bearings, in German)." *Zeitschrift des Vereins Deutscher Ingenieure*, 68(14), 339-341.

Peck, R. B. (1979). "Liquefaction Potential: Science Versus Practice." *Journal of the Geotechnical Engineering Division*, 105(3), 393-398.

Pook, L. P. (2007). *Metal fatigue: what it is, why it matters.* Solid mechanics and its applications, Springer, Dordrecht.

Pyke, R. M., Chan, C. K., and Seed, H. B. (1975). "Settlement of sands under multidirectional shaking." *Journal of the Geotechnical Engineering Division*, 101(4), 379-398.

R Core Team (2013). "R: A language and environment for statistical computing." R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Schneider, J.F., Silva, W.J. and Stark, C. (1993). "Ground motion model for the 1989 M 6.9 Loma Prieta earthquake including effects of source, path, and site", *Earthq. Spectra*, 9(2), 251-287.

Seed, H. B., and Idriss, I. M. (1971). "Simplified procedure for evaluating soil liquefaction potential." *Journal of the Soil Mechanics and Foundations Division*, 97(9), 1249–1273.

Seed, H. B., Idriss, I. M., Makdisi, F., and Banerjee, N. (1975). "Representation of irregular stress time histories by equivalent uniform stress series in liquefaction analyses, EERC 75-29." *Earthquake Engineering Research Center*, University of California, Berkeley.

Schnabel, P., Seed, H. B., and Lysmer, J. (1972). "Modification of seismograph records for effects of local soil conditions." *Bulletin of the Seismological Society of America*, 62(6), 1649–1664.

Silva, W. J., and Lee, K. (1987). "WES RASCAL code for synthesizing earthquake ground motions: state-of-the-art for assessing earthquake hazards in the United States." Report 24, US Army Engineering Waterways Experiment Station Vicksburg, MS.

Silva, W.J. (1993). "Factors controlling strong ground motion and their associated uncertainties", Dynamic Analysis and Design Considerations for High-Level Nuclear Waste Repositories, San Francisco, CA, USA, 132-161.

Whitman, R. V. (1971). "Resistance of Soil to Liquefaction and Settlement." *Soils and Foundations*, 11(4), 59–68.

Yoshimi, Y., Tokimatsu, K., Kaneko, O., and Makihara, Y. (1984). "Undrained cyclic shear strength of a dense Niigata sand." *Soils and Foundations*, 24(4), 131–145.

Youd, T. L., Idriss, I. M., Andrus, R. D., Arango, I., Castro, G., Christian, J. T., Dobry, R., Finn, W. D. L., Harder, L. F., Hynes, M. E., Ishihara, K., Koester, J. P., Liao, S. S. C., Marcuson, W. F., Martin, G. R., Mitchell, J. K., Moriwaki, Y., Power, M. S., Robertson, P. K., Seed, R. B., and Stokoe, K. H. (2001). "Liquefaction Resistance of Soils: Summary Report from the 1996 NCEER and 1998 NCEER/NSF Workshops on Evaluation of Liquefaction Resistance of Soils." *Journal of Geotechnical and Geoenvironmental Engineering*, 127(10), 817–833.

4.10 Tables

Table 4.1: Regression Coefficients for Lee's (2009) n_{eq} Correlation

Motion Database	C_1	C_2	C_3	C_4	C_5	σ_ε
WUS	-0.0101	1.664	0.131	0.122	-2.11	0.59
CEUS	-0.0190	1.857	0.136	0.050	-1.77	0.50

Table 4.2: Regression Coefficients for n_{eq} Form #1

Approach	Database	a_1	a_2	a_3
1	WUS	-0.3643	-0.4105	0.2553
1	CEUS	0.4654	-0.5626	0.1423
2	WUS	0.4605	-0.4082	0.2332
2	CEUS	1.147	-0.3772	0.1842
3	WUS	1.19	-0.2782	0.2051
3	CEUS	2.092	-0.5133	0.04894

Table 4.3: Standard Deviations of the Regression Coefficients for n_{eq} Form #1

Approach	Database	σ_{a_1}	σ_{a_2}	σ_{a_3}
1	WUS	0.07523	0.008105	0.01104
1	CEUS	0.05652	0.01291	0.008394
2	WUS	0.08616	0.009325	0.0118
2	CEUS	0.06136	0.03454	0.01345
3	WUS	0.07238	0.008759	0.01007
3	CEUS	0.05649	0.008551	0.007723

Table 4.4: Inter-event, Intra-event and Total Errors from Eqn. 4.15

Approach	Database	τ_{event}	τ_{profile}	σ_{ε_1}	σ_{Total}
1	WUS	0.4124	0.1931	0.4512	0.6411
1	CEUS	-	0.3021	0.5898	0.6627
2	WUS	0.4051	0.1856	0.3851	0.5889
2	CEUS	-	0.2960	0.5319	0.6087
3	WUS	0.3621	0.1491	0.4358	0.5858
3	CEUS	-	0.2620	0.5781	0.6347

Table 4.5: Regression Coefficients for n_{eq} Form #2

Approach	Database	b_1	b_2	b_3
1	WUS	0.09584	0.1597	0.3168
1	CEUS	1.44	0.1102	0.08772
2	WUS	0.7924	0.1396	0.3489
2	CEUS	1.738	0.1688	0.06753
3	WUS	1.578	0.1433	0.1919
3	CEUS	3.474	-0.03238	0.03011

Table 4.6: Standard Deviations of the Regression Coefficients for n_{eq} Form #2

Approach	Database	σ_{b_1}	σ_{b_2}	σ_{b_3}
1	WUS	0.07508	0.01132	0.008109
1	CEUS	0.06797	0.009412	0.005701
2	WUS	0.08633	0.0119	0.007591
2	CEUS	0.09444	0.01367	0.006156
3	WUS	0.07095	0.01034	0.008898
3	CEUS	0.07537	0.0105	0.00579

Table 4.7: Inter-event, Intra-event and Total Errors from Eqn. 4.17

Approach	Database	τ_{event}	τ_{profile}	σ_{ε_2}	σ_{Total}
1	WUS	0.3588	0.1424	0.4749	0.6120
1	CEUS	-	0.2020	0.7087	0.7369
2	WUS	0.3695	0.1386	0.3952	0.5585
2	CEUS	-	0.1978	0.6494	0.6789
3	WUS	0.3775	0.1191	0.4477	0.5976
3	CEUS	-	0.1705	0.6777	0.6988

Table 4.8: Strength of Correlation Between Various Measures of Duration and n_{eq} (WUS Dataset)

Duration	Relationship	$r_{n_{eq,\text{A1}}}$, $r_{n_{eq,\text{Sea75}}}$	$r_{n_{eq,\text{A2}}}$, $r_{n_{eq,\text{Lea01}}}$	$r_{n_{eq,\text{A3}}}$, $r_{n_{eq,\text{Lea01}}}$
Bracketed ($\pm 0.05g$)	log-log	0.05, 0.12	0.11, 0.19	0.11, 0.15
Significant (5 to 95%)	log-log	0.52, 0.39	0.48, 0.40	0.58, 0.51
Significant (5 to 75%)	log-log	0.59, 0.47	0.56, 0.50	0.64, 0.57

Table 4.9: Strength of Correlation Between Various Measures of Duration and n_{eq} (CEUS Dataset)

Duration	Relationship	$r_{n_{eq,\mathrm{A1}}}$, $r_{n_{eq,\mathrm{Sea75}}}$	$r_{n_{eq,\mathrm{A2}}}$, $r_{n_{eq,\mathrm{Lea01}}}$	$r_{n_{eq,\mathrm{A3}}}$, $r_{n_{eq,\mathrm{Lea01}}}$
Bracketed ($\pm 0.05g$)	log-log	0.13, 0.11	0.19, 0.18	0.26, 0.26
Significant (5 to 95%)	log-log	0.61, 0.53	0.60, 0.59	0.62, 0.61
Significant (5 to 75%)	log-log	0.65, 0.59	0.60, 0.62	0.64, 0.65

4.11 Figures

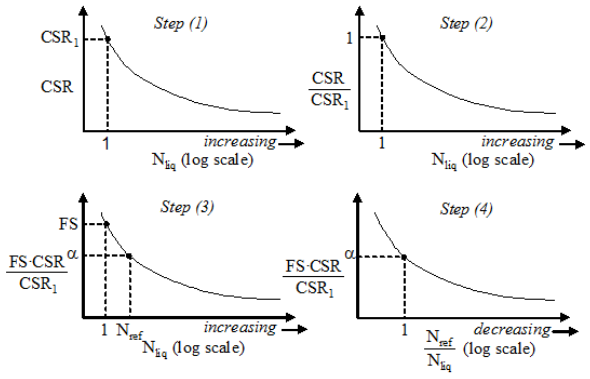


Figure 4.1: The sequence of steps used by Seed et al. (1975) to normalize a CSR-$N_{liq.}$ curve. Step 1: Designate the CSR required to induce liquefaction in one cycle as CSR_1; Step 2: Normalize the ordinate values by dividing CSR by CSR_1; Step 3: Apply a FS to CSR_1 (i.e., divide CSR_1 by FS) and determine the number of cycles required to induce liquefaction (N_{ref}) corresponding to the normalized peak amplitude (α) of the uniform cyclic load; and Step 4: Normalize the abscissa values as N_{ref}/N_{liq}.

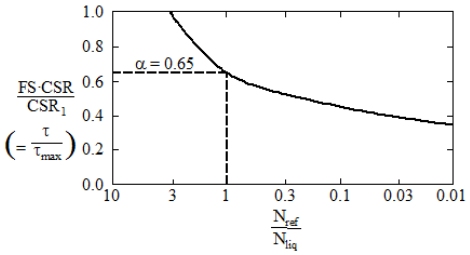


Figure 4.2: Normalized CSR-N_{liq} curve ($FS = 1.5$) used by Seed et al. (1975) to compute n_{eq}. Such curves are commonly referred to as weighting factor (WF) curves. The CSR-$N_{liq.}$ curve used to develop the WF curve came from De Alba et al. (1975): large-scale, undrained, cyclic simple shear tests performed on reconstituted Monterey No. 0 sand specimens having a relative density of approximately 65%. (adapted from Seed et al. 1975)

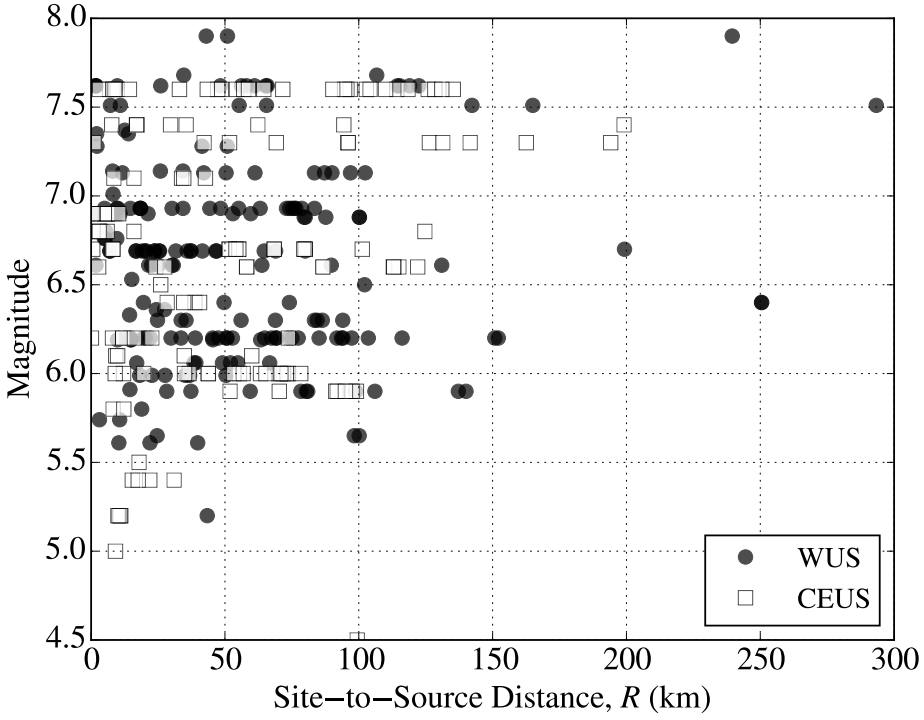


Figure 4.3: Magnitude versus distance distribution of the motions used in this study.

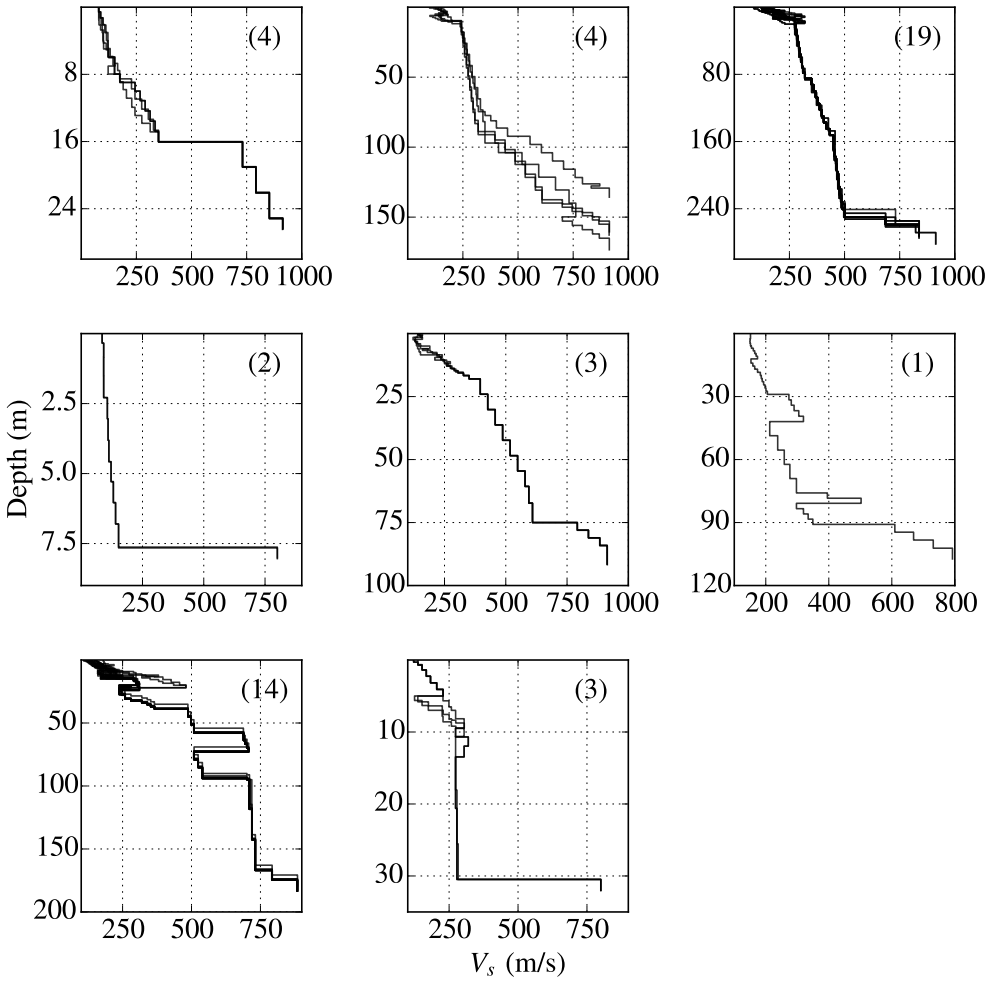


Figure 4.4: WUS profiles used in this study. The number in the corner of each subplot is the number of profiles in that grouping.

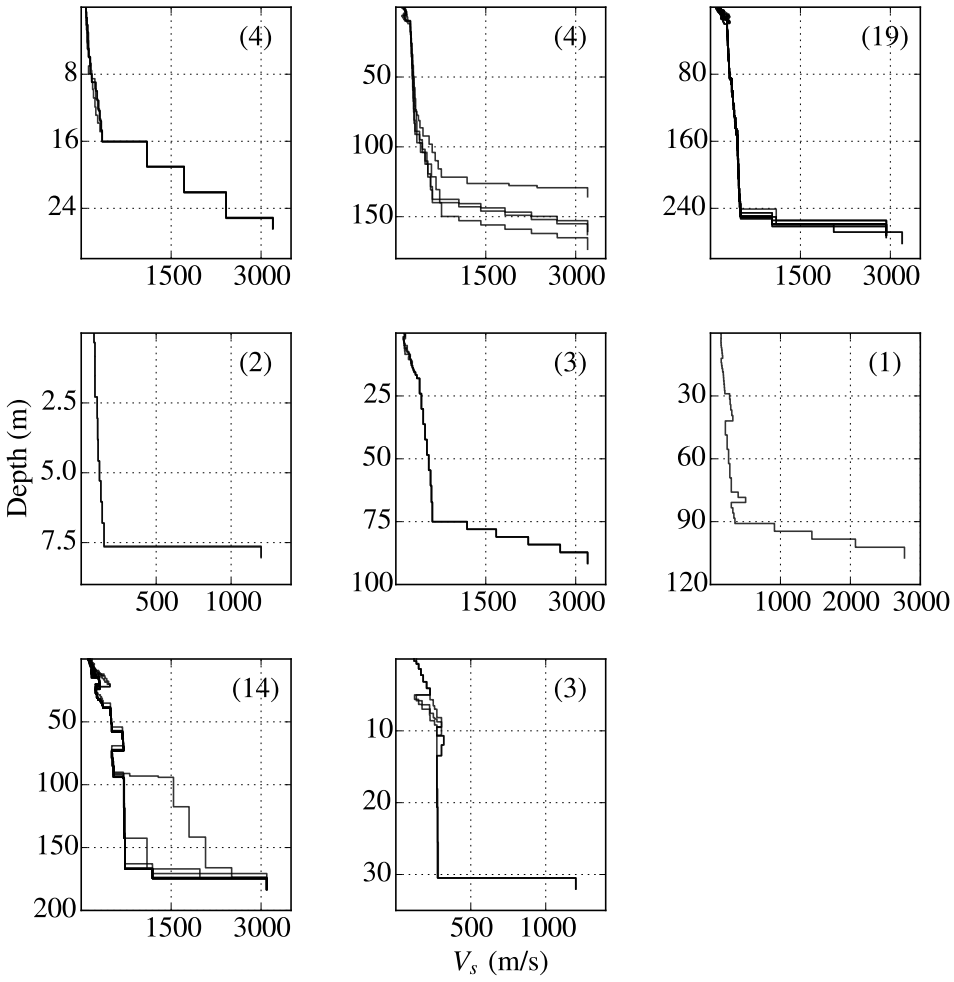


Figure 4.5: CEUS profiles used in this study. The number in the corner of each subplot is the number of profiles in that grouping.

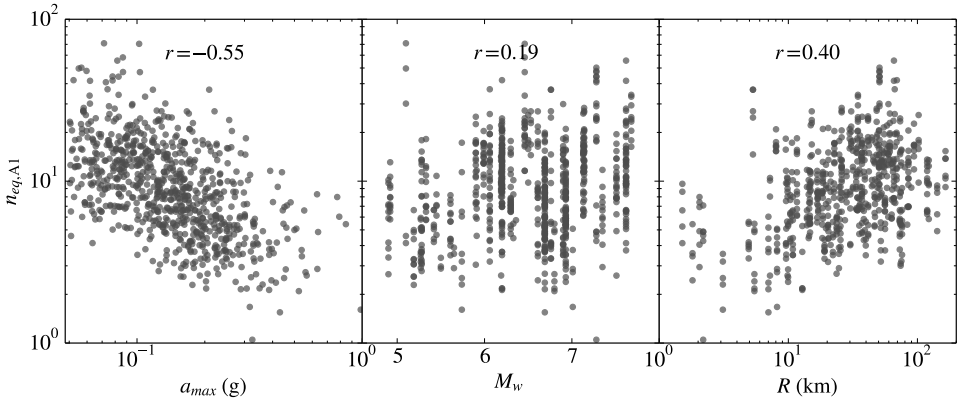


Figure 4.6: Scatterplots of a_{max}, M_w, and R versus $n_{eq,A1}$ from the WUS database. The number of equivalent cycles has a much higher correlation to the a_{max} at the ground surface than to M_w.

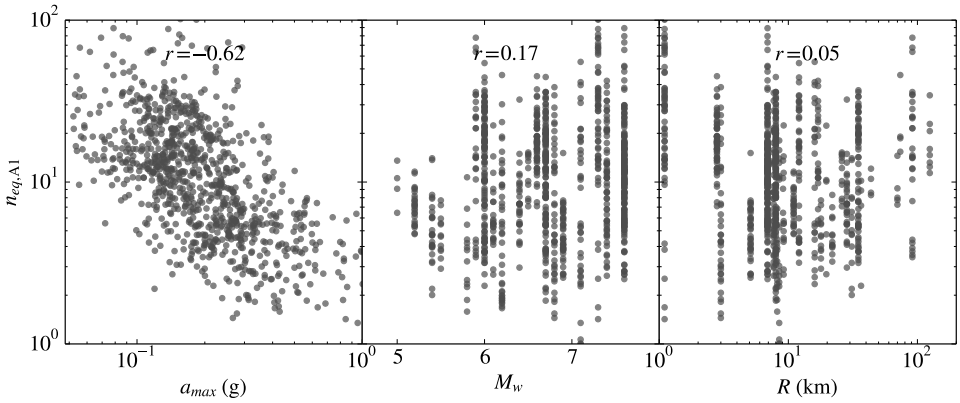


Figure 4.7: Scatterplots of a_{max}, M_w, and R versus $n_{eq,A1}$ from the CEUS database. The number of equivalent cycles has a much higher correlation to the a_{max} at the ground surface than to M_w.

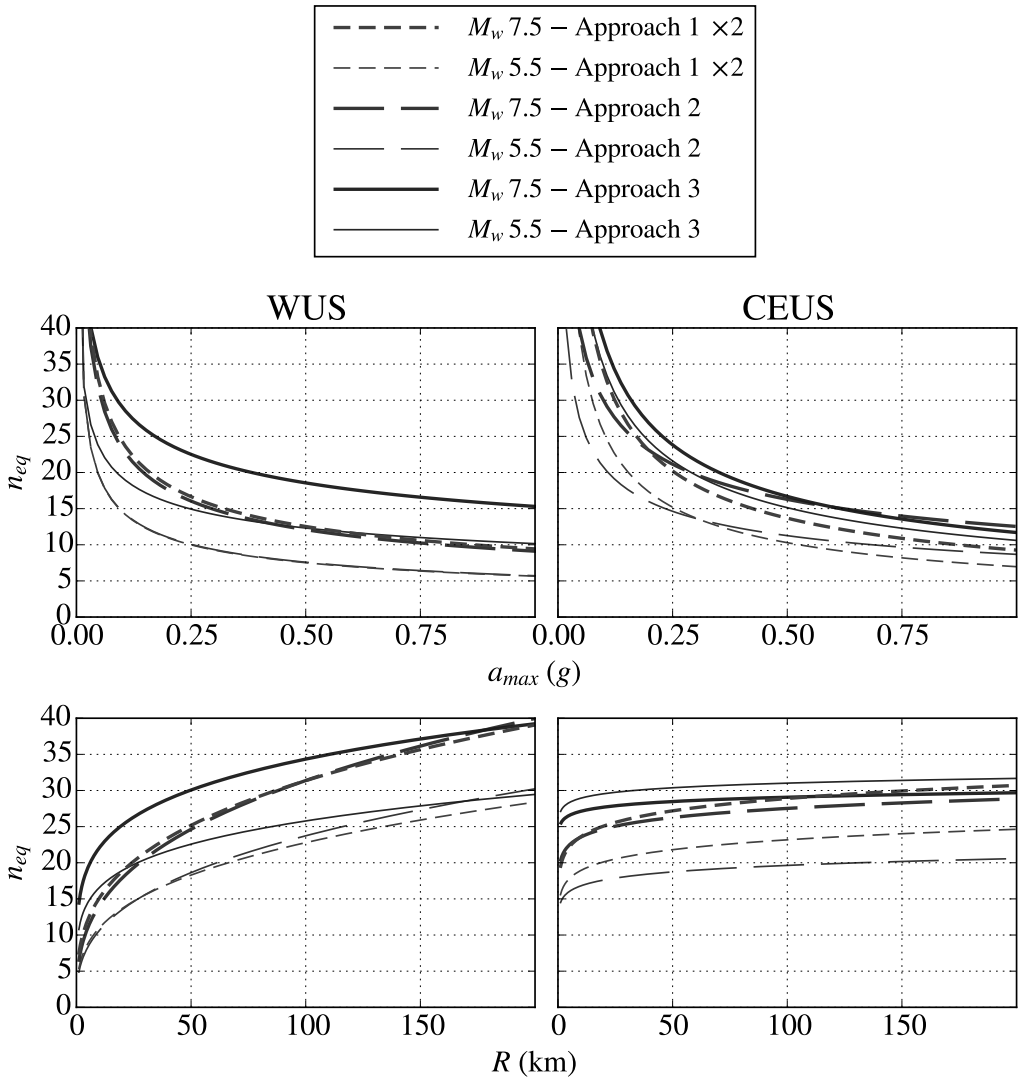


Figure 4.8: Values of n_{eq} predicted by the correlations proposed herein. The top plots show the relationship using Eqn. 4.15 and the bottom figures are for Eqn. 4.17. The n_{eq} correlations corresponding to the WUS are to the left; CEUS correlations are plotted on the right.

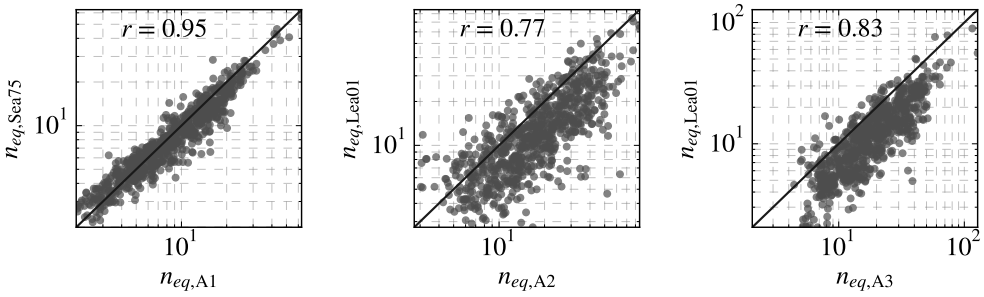


Figure 4.9: Comparison of n_{eq} values computed using Seed et al. (1975) and Liu et al. (2001) (i.e., $n_{eq,\text{Sea75}}$ and $n_{eq,\text{Lea01}}$, respectively) with values computed using Approaches 1, 2, and 3 for 0 to 4 m depths (i.e., $n_{eq,\text{A1}}$, $n_{eq,\text{A1}}$, and $n_{eq,\text{A1}}$, respectively) for WUS.

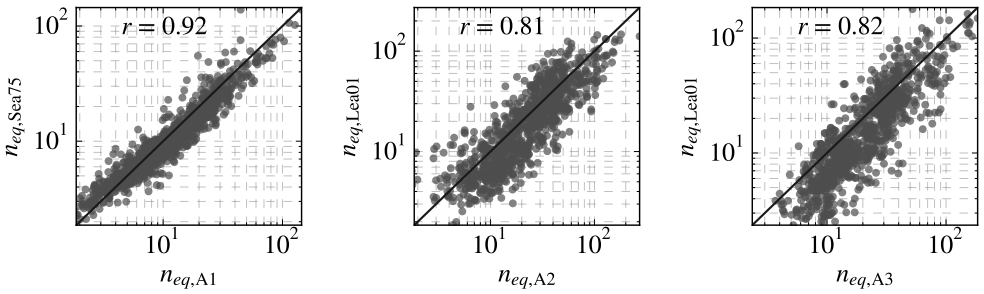


Figure 4.10: Comparison of n_{eq} values computed using Seed et al. (1975) and Liu et al. (2001) (i.e., $n_{eq,\text{Sea75}}$ and $n_{eq,\text{Lea01}}$, respectively) with values computed using Approaches 1, 2, and 3 for 0 to 4 m depths (i.e., $n_{eq,\text{A1}}$, $n_{eq,\text{A1}}$, and $n_{eq,\text{A1}}$, respectively) for CEUS.

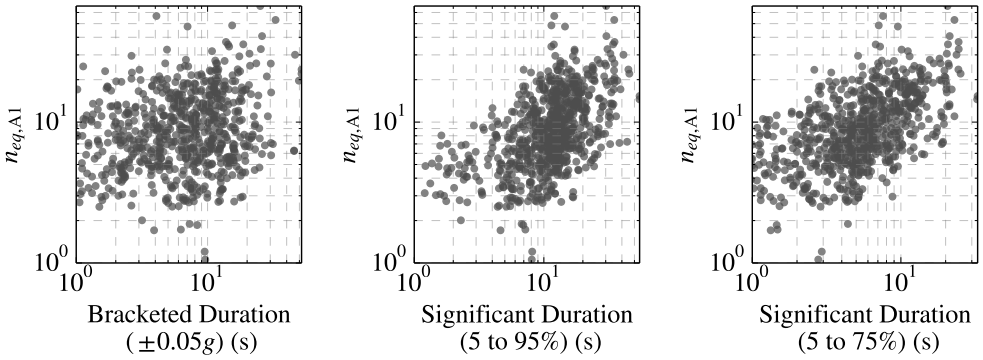


Figure 4.11: Plots of $n_{eq,\mathrm{A1}}$ for 0 to 4 m depths versus other ground motion duration metrics: bracketed duration ($\pm 0.05g$ threshold), 5-95% significant duration, and 5-75% significant duration for WUS.

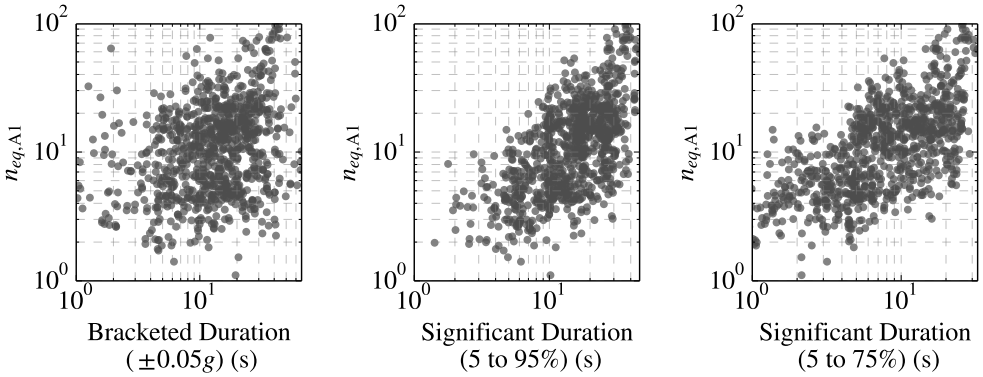


Figure 4.12: Plots of $n_{eq,\mathrm{A1}}$ for 0 to 4 m depths versus other ground motion duration metrics: bracketed duration ($\pm 0.05g$ threshold), 5-95% significant duration, and 5-75% significant duration for CEUS.

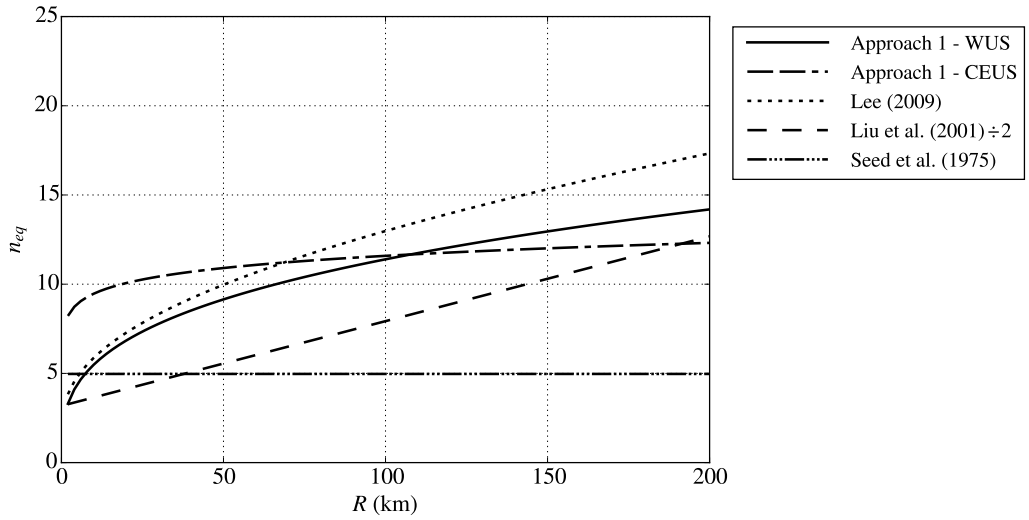


Figure 4.13: Comparison of n_{eq} correlations for $M = 5.5$. "WUS" and "CEUS" refer to the WUS and CEUS correlations developed herein from approach 1 ($n_{eq,\mathrm{A1}}$ values).

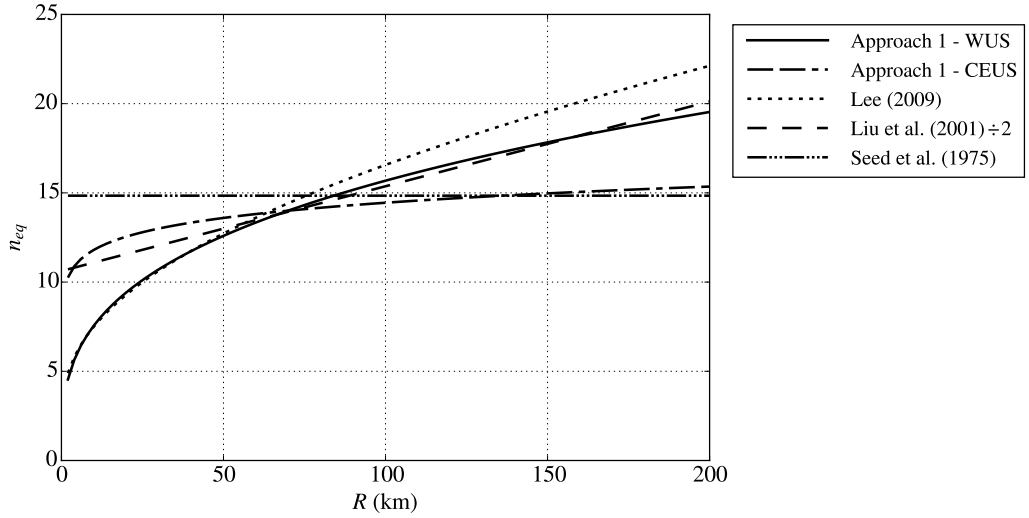


Figure 4.14: Comparison of n_{eq} correlations for $M = 7.5$. "WUS" and "CEUS" refer to the WUS and CEUS correlations developed herein from approach 1 ($n_{eq,\mathrm{A1}}$ values).

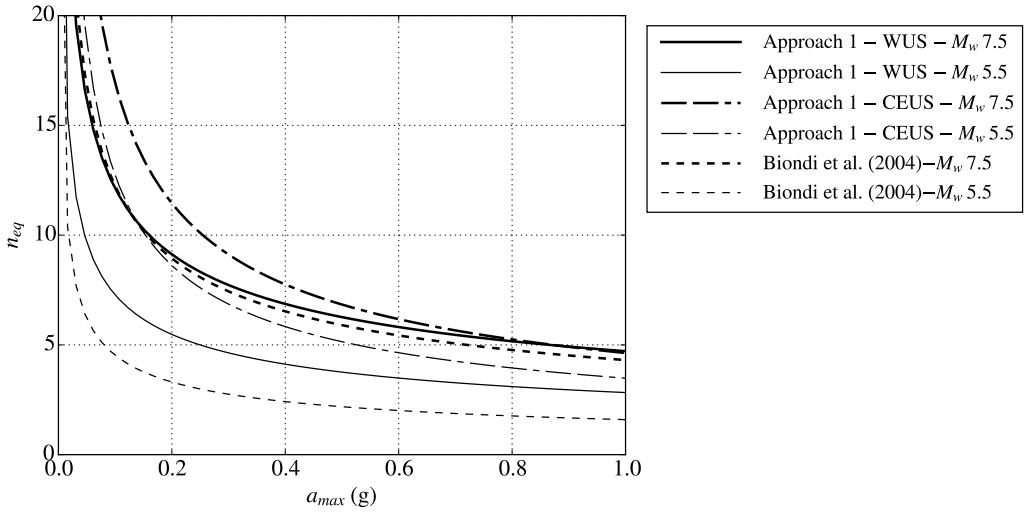


Figure 4.15: Comparison of functional form #1 (approach 1, $n_{eq,\mathrm{A1}}$) and the Biondi et al. (2004) n_{eq} correlations.

4.12 Electronic Supplement

4.12.1 NGA Database (WUS)

The distance given in the tables, R, is the closest distance to the rupture area (an *nan* indicates that the value is not known). The units of PGA and R are g and km, respectively.

Table 4.10: WUS (NGA) Earthquake Motions

No.	Names	Event	M_w	PGA	R	Mechanism
1	1011_WON095 1011_WON185	Northridge -01	6.69	0.13	20.3	Reverse
2	1012_LA0000 1012_LA0090	Northridge -01	6.69	0.32	19.1	Reverse
3	1021_L04000 1021_L04090	Northridge -01	6.69	0.08	31.7	Reverse
4	1023_L09090 1023_L09000	Northridge -01	6.69	0.17	25.4	Reverse
5	1027_LV1000 1027_LV1090	Northridge -01	6.69	0.08	37.2	Reverse
6	1029_LV3000 1029_LV3090	Northridge -01	6.69	0.09	37.3	Reverse
7	1033_LIT090 1033_LIT180	Northridge -01	6.69	0.07	46.6	Reverse
8	1041_MTW000 1041_MTW090	Northridge -01	6.69	0.17	35.9	Reverse
9	1050_PAC265 1050_PAC175	Northridge -01	6.69	0.41	7.0	Reverse
10	1051_PUL104 1051_PUL194	Northridge -01	6.69	1.43	7.0	Reverse
11	1060_CUC090 1060_CUC180	Northridge -01	6.69	0.06	80.0	Reverse
12	1074_SAN090 1074_SAN180	Northridge -01	6.69	0.09	41.6	Reverse
13	1078_5108-090 1078_5108-360	Northridge -01	6.69	0.25	16.7	Reverse

No.	Names	Event	M_w	PGA	R	Mechanism
14	1091_VAS000 1091_VAS090	Northridge -01	6.69	0.14	23.6	Reverse
15	1096_WWJ090 1096_WWJ180	Northridge -01	6.69	0.05	64.7	Reverse
16	1142_IZ1090 1142_IZ1000	Dinar, Turkey	6.4	0.0	250.5	Normal
17	1154_BRS090 1154_BRS180	Kocaeli, Turkey	7.51	0.06	65.5	Strike-Slip
18	1159_ERG090 1159_ERG180	Kocaeli, Turkey	7.51	0.1	142.3	Strike-Slip
19	1165_IZT090 1165_IZT180	Kocaeli, Turkey	7.51	0.2	7.2	Strike-Slip
20	1168_MNS000 1168_MNS090	Kocaeli, Turkey	7.51	0.01	293.4	Strike-Slip
21	1169_MSK000 1169_MSK090	Kocaeli, Turkey	7.51	0.04	55.3	Strike-Slip
22	1172_TKR090 1172_TKR180	Kocaeli, Turkey	7.51	0.04	165.0	Strike-Slip
23	124_A-FLT000 124_A-FLT270	Friuli, Italy-01	6.5	0.03	102.2	Reverse
24	1257_HWA003-W 1257_HWA003-N	Chi-Chi, Taiwan	7.62	0.09	56.1	Reverse-Oblique
25	126_GAZ000 126_GAZ090	Gazli, USSR	6.8	0.64	5.5	Unknown
26	133_B-SRO000 133_B-SRO270	Friuli, Italy-02	5.91	0.1	14.5	Reverse
27	1347_ILA063-N 1347_ILA063-W	Chi-Chi, Taiwan	7.62	0.09	61.1	Reverse-Oblique
28	1352_KAU003-N 1352_KAU003-W	Chi-Chi, Taiwan	7.62	0.02	114.4	Reverse-Oblique
29	139_DAY-LN 139_DAY-TR	Tabas, Iran	7.35	0.35	13.9	Reverse
30	143_TAB-LN 143_TAB-TR	Tabas, Iran	7.35	0.81	2.0	Reverse
31	1440_TAP065-E 1440_TAP065-N	Chi-Chi, Taiwan	7.62	0.03	122.5	Reverse-Oblique

No.	Names	Event	M_w	PGA	R	Mechanism
32	1446_TAP077-N 1446_TAP077-W	Chi-Chi, Taiwan	7.62	0.03	119.0	Reverse-Oblique
33	146_G01230 146_G01320	Coyote Lake	5.74	0.12	10.7	Strike-Slip
34	150_G06230 150_G06320	Coyote Lake	5.74	0.4	3.1	Strike-Slip
35	1518_TCU085-E 1518_TCU085-N	Chi-Chi, Taiwan	7.62	0.06	58.1	Reverse-Oblique
36	1529_TCU102-E 1529_TCU102-N	Chi-Chi, Taiwan	7.62	0.24	1.5	Reverse-Oblique
37	155_F-BEV-EW 155_F-BEV-NS	Norcia, Italy	5.9	0.03	nan	Normal
38	1551_TCU138-N 1551_TCU138-W	Chi-Chi, Taiwan	7.62	0.21	9.8	Reverse-Oblique
39	156_F-CSC-EW 156_F-CSC-NS	Norcia, Italy	5.9	0.19	nan	Normal
40	1577_TTN025-E 1577_TTN025-N	Chi-Chi, Taiwan	7.62	0.04	65.8	Reverse-Oblique
41	1585_TTN040-N 1585_TTN040-W	Chi-Chi, Taiwan	7.62	0.03	48.3	Reverse-Oblique
42	1587_TTN042-N 1587_TTN042-W	Chi-Chi, Taiwan	7.62	0.06	65.3	Reverse-Oblique
43	1613_1060-E 1613_1060-N	Duzce, Turkey	7.14	0.04	25.9	Strike-Slip
44	1618_531-E 1618_531-N	Duzce, Turkey	7.14	0.14	8.0	Strike-Slip
45	1619_MDR000 1619_MDR090	Duzce, Turkey	7.14	0.09	34.3	Strike-Slip
46	1626_212V5180 1626_212V5090	Sitka, Alaska	7.68	0.09	34.6	Strike-Slip
47	164_H-CPE147 164_H-CPE237	Imperial Valley-06	6.53	0.18	15.2	Strike-Slip
48	1645_mtwin000 1645_mtwin090	Sierra Madre	5.61	0.23	10.4	Reverse
49	1649_vquez000 1649_vquez090	Sierra Madre	5.61	0.11	39.8	Reverse

No.	Names	Event	M_w	PGA	R	Mechanism
50	1691_ANACA000	Northridge	5.28	0.01	nan	Reverse
	1691_ANACA270	-06				
51	1696_HOW060	Northridge	5.28	0.07	nan	Reverse
	1696_HOW330	-06				
52	1709_0GPN00E	Northridge	5.28	0.04	nan	Reverse
	1709_0GPN90W	-06				
53	1715_WON095	Northridge	5.28	0.05	nan	Reverse
	1715_WON185	-06				
54	1718_LITTL090	Northridge	5.28	0.01	nan	Reverse
	1718_LITTL360	-06				
55	1720_MCS025	Northridge	5.28	0.05	nan	Reverse
	1720_MCS115	-06				
56	1727_RANCH180	Northridge	5.28	0.01	nan	Reverse
	1727_RANCH090	-06				
57	1741_Lsm2000	Little	5.65	0.1	24.7	Normal
	1741_Lsm2270	Skull				
	1741_Lsm2270	Mtn,NV				
58	1745_Lsm6000	Little	5.65	0.01	100.2	Normal
	1745_Lsm6270	Skull				
	1745_Lsm6270	Mtn,NV				
59	1747_Lsm8270	Little	5.65	0.01	98.3	Normal
	1747_Lsm8000	Skull				
	1747_Lsm8000	Mtn,NV				
60	1767_12674090	Hector	7.13	0.02	83.4	Strike-Slip
	1767_12674360	Mine				
61	1786_22T04090	Hector	7.13	0.08	61.2	Strike-Slip
	1786_22T04180	Mine				
62	1795_12647090	Hector	7.13	0.08	50.4	Strike-Slip
	1795_12647180	Mine				
63	1836_22161090	Hector	7.13	0.06	42.1	Strike-Slip
	1836_22161360	Mine				
64	1917_02467360	Anza-02	4.92	0.01	nan	Normal-Oblique
	1917_02467090					
65	1942_12116036	Anza-02	4.92	0.06	nan	Normal-Oblique
	1942_12116126					
66	1961_13095026	Anza-02	4.92	0.03	nan	Normal-Oblique

No.	Names	Event	M_w	PGA	R	Mechanism
	1961_13095116					
67	1972_RVB090	Anza-02	4.92	0.01	nan	Normal-Oblique
	1972_RVB360					
68	2017_2017090	Gilroy	4.9	0.01	nan	Strike-Slip
	2017_2017360					
69	2019_47006067	Gilroy	4.9	0.23	nan	Strike-Slip
	2019_47006337					
70	2021_57383090	Gilroy	4.9	0.09	nan	Strike-Slip
	2021_57383360					
71	2032_2014090	Gilroy	4.9	0.01	nan	Strike-Slip
	2032_2014180					
72	2046_0438090	Gilroy	4.9	0.02	nan	Strike-Slip
	2046_0438360					
73	2091_ps07039	Nenana	6.7	0.01	199.3	Strike-Slip
	2091_ps07309	Mountain,				
	2091_ps07309	Alaska				
74	2118_1727090	Denali,	7.9	0.01	239.5	Strike-Slip
	2118_1727360	Alaska				
75	222_B-LMO265	Livermore-	5.42	0.23	nan	Strike-Slip
	222_B-LMO355	02				
76	225_PFT045	Anza	5.19	0.12	nan	Strike-Slip
	225_PFT135	(Horse				
	225_PFT135	Canyon)-01				
77	226_TVY045	Anza	5.19	0.11	nan	Strike-Slip
	226_TVY135	(Horse				
	226_TVY135	Canyon)-01				
78	2296_ILA063-N	Chi-Chi,	5.9	0.01	80.4	Reverse
	2296_ILA063-W	Taiwan-02				
79	23_GGP010	San	5.28	0.11	nan	Reverse
	23_GGP100	Francisco				
80	2336_TAP077-N	Chi-Chi,	5.9	0.01	140.1	Reverse
	2336_TAP077-W	Taiwan-02				
81	2367_TCU045-E	Chi-Chi,	5.9	0.02	59.4	Reverse
	2367_TCU045-N	Taiwan-02				
82	2396_TCU085-E	Chi-Chi,	5.9	0.01	78.4	Reverse
	2396_TCU085-N	Taiwan-02				

No.	Names	Event	M_w	PGA	R	Mechanism
83	2423_TCU129-E 2423_TCU129-N	Chi-Chi, Taiwan-02	5.9	0.12	28.3	Reverse
84	2427_TCU138-N 2427_TCU138-W	Chi-Chi, Taiwan-02	5.9	0.04	37.3	Reverse
85	2439_TTN025-E 2439_TTN025-N	Chi-Chi, Taiwan-02	5.9	0.01	106.0	Reverse
86	2445_TTN040-N 2445_TTN040-W	Chi-Chi, Taiwan-02	5.9	0.01	80.9	Reverse
87	2447_TTN042-N 2447_TTN042-W	Chi-Chi, Taiwan-02	5.9	0.01	98.8	Reverse
88	2601_TCU045-E 2601_TCU045-N	Chi-Chi, Taiwan-03	6.2	0.01	77.4	Reverse
89	2633_TCU085-E 2633_TCU085-N	Chi-Chi, Taiwan-03	6.2	0.0	103.6	Reverse
90	2640_TCU102-E 2640_TCU102-N	Chi-Chi, Taiwan-03	6.2	0.02	45.4	Reverse
91	265_CPE045 265_CPE315	Victoria, Mexico	6.33	0.57	14.4	Strike-Slip
92	2658_TCU129-E 2658_TCU129-N	Chi-Chi, Taiwan-03	6.2	0.61	12.8	Reverse
93	2661_TCU138-W 2661_TCU138-N	Chi-Chi, Taiwan-03	6.2	0.13	22.1	Reverse
94	2677_TTN025-E 2677_TTN025-N	Chi-Chi, Taiwan-03	6.2	0.01	97.4	Reverse
95	2685_TTN040-W 2685_TTN040-N	Chi-Chi, Taiwan-03	6.2	0.01	75.1	Reverse
96	2687_TTN042-N 2687_TTN042-W	Chi-Chi, Taiwan-03	6.2	0.01	93.6	Reverse
97	2805_KAU003-N 2805_KAU003-W	Chi-Chi, Taiwan-04	6.2	0.01	116.2	Strike-Slip
98	283_A-ARI000 283_A-ARI270	Irpinia, Italy-01	6.9	0.04	52.9	Normal
99	284_A-AUL000 284_A-AUL270	Irpinia, Italy-01	6.9	0.06	9.6	Normal
100	285_A-BAG000 285_A-BAG270	Irpinia, Italy-01	6.9	0.16	8.2	Normal

No.	Names	Event	M_w	PGA	R	Mechanism
101	286_A-BIS000	Irpinia,	6.9	0.09	21.3	Normal
	286_A-BIS270	Italy-01				
102	2877_TCU102-E	Chi-Chi,	6.2	0.01	64.8	Strike-Slip
	2877_TCU102-N	Taiwan-04				
103	2897_TCU138-N	Chi-Chi,	6.2	0.04	33.6	Strike-Slip
	2897_TCU138-W	Taiwan-04				
104	2919_TTN025-E	Chi-Chi,	6.2	0.02	69.3	Strike-Slip
	2919_TTN025-N	Taiwan-04				
105	292_A-STU000	Irpinia,	6.9	0.29	10.8	Normal
	292_A-STU270	Italy-01				
106	2927_TTN040-N	Chi-Chi,	6.2	0.02	50.8	Strike-Slip
	2927_TTN040-W	Taiwan-04				
107	2929_TTN042-W	Chi-Chi,	6.2	0.03	69.0	Strike-Slip
	2929_TTN042-N	Taiwan-04				
108	293_A-TDG000	Irpinia,	6.9	0.05	59.6	Normal
	293_A-TDG270	Italy-01				
109	295_B-AUL000	Irpinia,	6.2	0.02	29.9	Normal
	295_B-AUL270	Italy-02				
110	296_B-BAG000	Irpinia,	6.2	0.05	19.6	Normal
	296_B-BAG270	Italy-02				
111	297_B-BIS000	Irpinia,	6.2	0.07	14.7	Normal
	297_B-BIS270	Italy-02				
112	2996_HWA003-N	Chi-Chi,	6.2	0.03	50.4	Reverse
	2996_HWA003-W	Taiwan-05				
113	303_B-STU000	Irpinia,	6.2	0.08	20.4	Normal
	303_B-STU270	Italy-02				
114	3139_TAP077-N	Chi-Chi,	6.2	0.01	152.1	Reverse
	3139_TAP077-W	Taiwan-05				
115	3172_TCU045-E	Chi-Chi,	6.2	0.05	73.5	Reverse
	3172_TCU045-N	Taiwan-05				
116	3194_TCU085-E	Chi-Chi,	6.2	0.01	91.8	Reverse
	3194_TCU085-N	Taiwan-05				
117	3202_TCU102-E	Chi-Chi,	6.2	0.05	52.8	Reverse
	3202_TCU102-N	Taiwan-05				
118	3217_TCU129-E	Chi-Chi,	6.2	0.39	38.9	Reverse
	3217_TCU129-N	Taiwan-05				

No.	Names	Event	M_w	PGA	R	Mechanism
119	3220_TCU138-N 3220_TCU138-W	Chi-Chi, Taiwan-05	6.2	0.18	47.5	Reverse
120	3241_TTN025-E 3241_TTN025-N	Chi-Chi, Taiwan-05	6.2	0.04	94.1	Reverse
121	3249_TTN040-N 3249_TTN040-W	Chi-Chi, Taiwan-05	6.2	0.04	67.4	Reverse
122	3251_TTN042-W 3251_TTN042-N	Chi-Chi, Taiwan-05	6.2	0.05	85.2	Reverse
123	3325_HWA003-N 3325_HWA003-W	Chi-Chi, Taiwan-06	6.3	0.03	56.0	Reverse
124	3390_ILA063-N 3390_ILA063-W	Chi-Chi, Taiwan-06	6.3	0.01	84.5	Reverse
125	3479_TCU085-E 3479_TCU085-N	Chi-Chi, Taiwan-06	6.3	0.01	83.4	Reverse
126	3489_TCU102-E 3489_TCU102-N	Chi-Chi, Taiwan-06	6.3	0.05	35.5	Reverse
127	3507_TCU129-E 3507_TCU129-N	Chi-Chi, Taiwan-06	6.3	0.26	24.8	Reverse
128	3509_TCU138-N 3509_TCU138-W	Chi-Chi, Taiwan-06	6.3	0.06	33.6	Reverse
129	3532_TTN025-E 3532_TTN025-N	Chi-Chi, Taiwan-06	6.3	0.02	94.0	Reverse
130	3540_TTN040-N 3540_TTN040-W	Chi-Chi, Taiwan-06	6.3	0.02	68.8	Reverse
131	3542_TTN042-N 3542_TTN042-W	Chi-Chi, Taiwan-06	6.3	0.03	86.4	Reverse
132	369_H-SCN045 369_H-SCN315	Coalinga-0 1	6.36	0.15	27.5	Reverse
133	43_CSM095 43_CSM185	Lytle Creek	5.33	0.07	nan	Reverse-Oblique
134	443_CEM000 443_CEM090 443_CEM090	Borah Peak, ID-02	5.1	0.02	nan	Normal
135	444_HAU000 444_HAU090 444_HAU090	Borah Peak, ID-02	5.1	0.03	nan	Normal

No.	Names	Event	M_w	PGA	R	Mechanism
136	45_DCF090 45_DCF180	Lytle Creek	5.33	0.13	nan	Reverse-Oblique
137	454_GIL067 454_GIL337	Morgan Hill	6.19	0.1	14.8	Strike-Slip
138	455_G01230 455_G01320	Morgan Hill	6.19	0.08	14.9	Strike-Slip
139	459_G06000 459_G06090	Morgan Hill	6.19	0.28	9.9	Strike-Slip
140	476_LOB050 476_LOB320	Morgan Hill	6.19	0.06	45.5	Strike-Slip
141	49_SAD003 49_SAD273	Lytle Creek	5.33	0.03	nan	Reverse-Oblique
142	495_S1010 495_S1280	Nahanni, Canada	6.76	1.06	9.6	Reverse
143	496_S2240 496_S2330	Nahanni, Canada	6.76	0.38	4.9	Reverse
144	497_S3270 497_S3360	Nahanni, Canada	6.76	0.15	5.3	Reverse
145	501_D-SG3205 501_D-SG3295	Hollister-04	5.45	0.06	nan	Strike-Slip
146	511_ARM270 511_ARM360	N. Palm Springs	6.06	0.12	38.4	Reverse-Oblique
147	512_ATL270 512_ATL360	N. Palm Springs	6.06	0.1	52.1	Reverse-Oblique
148	525_LMR252 525_LMR162	N. Palm Springs	6.06	0.05	66.7	Reverse-Oblique
149	528_H01000 528_H01090	N. Palm Springs	6.06	0.05	54.8	Reverse-Oblique
150	536_ARS270 536_ARS360	N. Palm Springs	6.06	0.11	39.1	Reverse-Oblique
151	537_SIL000 537_SIL090	N. Palm Springs	6.06	0.12	17.0	Reverse-Oblique
152	541_H02000 541_H02090	N. Palm Springs	6.06	0.08	49.1	Reverse-Oblique
153	585_CPE161 585_CPE251	Baja California	5.5	1.27	nan	Strike-Slip

No.	Names	Event	M_w	PGA	R	Mechanism
154	59_CSM095	San	6.61	0.02	89.7	Reverse
	59_CSM185	Fernando				
155	63_FTR056	San	6.61	0.09	30.2	Reverse
	63_FTR326	Fernando				
156	631_A-CHL030	Whittier	5.99	0.03	35.2	Reverse-Oblique
	631_A-CHL120	Narrows-01				
157	643_A-WON075	Whittier	5.99	0.04	27.6	Reverse-Oblique
	643_A-WON165	Narrows-01				
158	661_A-ANG000	Whittier	5.99	0.08	36.8	Reverse-Oblique
	661_A-ANG090	Narrows-01				
159	663_A-MTW000	Whittier	5.99	0.16	22.7	Reverse-Oblique
	663_A-MTW090	Narrows-01				
160	67_ISD014	San	6.61	0.01	131.0	Reverse
	67_ISD284	Fernando				
161	703_A-VAS000	Whittier	5.99	0.06	50.4	Reverse-Oblique
	703_A-VAS090	Narrows-01				
162	715_B-MTW000	Whittier	5.27	0.15	nan	Reverse-Oblique
	715_B-MTW090	Narrows-02				
163	72_L04111	San	6.61	0.16	25.1	Reverse
	72_L04201	Fernando				
164	73_L09021	San	6.61	0.14	22.6	Reverse
	73_L09291	Fernando				
165	763_GIL337	Loma	6.93	0.33	10.0	Reverse-Oblique
	763_GIL067	Prieta				
166	765_G01000	Loma	6.93	0.44	9.6	Reverse-Oblique
	765_G01090	Prieta				
167	769_G06000	Loma	6.93	0.16	18.3	Reverse-Oblique
	769_G06090	Prieta				
168	77_PUL164	San	6.61	1.16	1.8	Reverse
	77_PUL254	Fernando				
169	782_MCH000	Loma	6.93	0.07	44.3	Reverse-Oblique
	782_MCH090	Prieta				
170	788_PJH045	Loma	6.93	0.07	73.0	Reverse-Oblique
	788_PJH315	Prieta				
171	789_PTB207	Loma	6.93	0.07	83.4	Reverse-Oblique
	789_PTB297	Prieta				

No.	Names	Event	M_w	PGA	R	Mechanism
172	791_SG3261	Loma	6.93	0.07	34.3	Reverse-Oblique
	791_SG3351	Prieta				
173	795_PHT270	Loma	6.93	0.05	76.1	Reverse-Oblique
	795_PHT360	Prieta				
174	797_RIN000	Loma	6.93	0.09	74.1	Reverse-Oblique
	797_RIN090	Prieta				
175	798_TLH000	Loma	6.93	0.06	76.5	Reverse-Oblique
	798_TLH090	Prieta				
176	801_SJTE225	Loma	6.93	0.28	14.7	Reverse-Oblique
	801_SJTE315	Prieta				
177	804_SSF115	Loma	6.93	0.08	63.2	Reverse-Oblique
	804_SSF205	Prieta				
178	809_UC2000	Loma	6.93	0.34	18.5	Reverse-Oblique
	809_UC2090	Prieta				
179	810_LOB000	Loma	6.93	0.46	18.4	Reverse-Oblique
	810_LOB090	Prieta				
180	813_YBI000	Loma	6.93	0.06	75.2	Reverse-Oblique
	813_YBI090	Prieta				
181	828_PET000	Cape	7.01	0.62	8.2	Reverse
	828_PET090	Mendocino				
182	87_SAD003	San	6.61	0.17	30.7	Reverse
	87_SAD273	Fernando				
183	879_LCN260	Landers	7.28	0.72	2.2	Strike-Slip
	879_LCN345					
184	89_TEH090	San	6.61	0.04	63.8	Reverse
	89_TEH180	Fernando				
185	891_SIL000	Landers	7.28	0.05	50.8	Strike-Slip
	891_SIL090					
186	897_29P000	Landers	7.28	0.07	41.4	Strike-Slip
	897_29P090					
187	922_PPC090	Big	6.46	0.03	nan	Strike-Slip
	922_PPC180	Bear-01				
188	925_RCD090	Big	6.46	0.04	nan	Strike-Slip
	925_RCD180	Bear-01				
189	934_SVP090	Big	6.46	0.06	nan	Strike-Slip
	934_SVP360	Bear-01				

No.	Names	Event	M_w	PGA	R	Mechanism
190	938_WBR090	Big	6.46	0.08	nan	Strike-Slip
	938_WBR360	Bear-01				
191	943_ACI000	Northridge	6.69	0.05	68.9	Reverse
	943_ACI270	-01				
192	946_ATB000	Northridge	6.69	0.06	46.9	Reverse
	946_ATB090	-01				
193	957_HOW060	Northridge	6.69	0.14	16.9	Reverse
	957_HOW330	-01				
194	989_CHL070	Northridge	6.69	0.21	20.5	Reverse
	989_CHL160	-01				
195	994_0141-270	Northridge	6.69	0.25	23.8	Reverse
	994_0141-360	-01				

4.12.2 McGuire et al. (2001) Database (CEUS)

The distance given in the tables, R, is the closest distance to the rupture area (an *nan* indicates that the value is not known). The units of R and PGA are km and g, respectively.

Table 4.11: Central-Eastern United States (CEUS) Earthquake Motions

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
1	1125A54E	Saguenay	5.9	0.01	91.4	M55D050.100
	1125A54N			0.01		
2	1125A61E	Saguenay	5.9	0.01	91.9	M55D050.100
	1125A61N			0.01		
3	1125A64E	Saguenay	5.9	0.01	99.1	M55D050.100
	1125A64N			0.01		
4	1125S08L	Saguenay	5.9	0.13	97.5	M55D050.100
	1125S08T			0.06		
5	1125S16L	Saguenay	5.9	0.11	51.9	M55D050.100
	1125S16T			0.13		
6	1125S17L	Saguenay	5.9	0.17	70.3	M55D050.100
	1125S17T			0.1		

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
7	1125S20L	Saguenay	5.9	0.15	95.0	M55D050.100
	1125S20T			0.1		
8	29P000	Landers	7.3	0.21	42.2	M75D010.050
	29P090			0.18		
9	A-CSH000	Whittier Narrows	6.0	0.06	70.9	M65D050.100
	A-CSH090			0.07		
10	A-GRN180	Whittier Narrows	6.0	0.73	9.0	M65D000.010
	A-GRN270			0.53		
11	A-GRV060	Whittier Narrows	6.0	0.56	12.1	M55D000.050
	A-GRV330			0.8		
12	A-HMC180	Helena, Montana	6.2	0.4	8.0	M65D000.010
	A-HMC270			0.44		
13	A-LVL000	Mammoth Lakes	6.0	0.23	19.7	M65D010.050
	A-LVL090			0.13		
14	A-MAL180	Whittier Narrows	6.0	0.1	65.3	M65D050.100
	A-MAL270			0.09		
15	A-ORR000	Whittier Narrows	6.0	0.13	78.3	M65D050.100
	A-ORR090			0.11		
16	A-RIV180	Whittier Narrows	6.0	0.15	56.8	M65D050.100
	A-RIV270			0.13		
17	A-SON033	Borrego Mtn	6.8	0.11	124.7	M65D100.200
	A-SON303			0.1		
18	A-VIR200	Whittier Narrows	6.0	0.12	53.3	M65D050.100
	A-VIR290			0.08		
19	A54–N	Saguenay	4.5	0.0	99.4	M55D050.100
	A54–E			0.0		
20	ABY000	Landers	7.3	0.37	69.2	M75D050.100
	ABY090			0.43		
21	AMB–X	Georgia, USSR	6.2	0.04	73.7	M65D050.100
	AMB–Y			0.04		
22	ARC000	Kocaeli, Turkey	7.4	0.3	17.0	M75D010.050
	ARC090			0.21		
23	ARS270	N. Palm Springs	6.0	0.29	43.8	M65D010.050
	ARS360			0.3		
24	ATL270	N. Palm Springs	6.0	0.21	55.4	M55D050.100
	ATL360			0.16		

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
25	B-A3E146	Livermore	5.4	0.08	31.0	M55D000.050
	B-A3E236			0.04		
26	B-KOD180	Livermore	5.4	0.34	17.6	M55D000.050
	B-KOD270			0.18		
27	BOL000	Duzce, Turkey	7.1	1.64	16.0	M75D010.050
	BOL090			1.76		
28	BRN000	Loma Prieta	6.9	1.3	10.3	M75D000.010
	BRN090			1.24		
29	C-ATC270	Coalinga	5.2	0.66	11.0	M55D000.050
	C-ATC360			0.72		
30	C-ATP270	Coalinga	5.2	0.94	11.0	M55D000.050
	C-ATP360			0.56		
31	C-OLC270	Coalinga	5.2	0.7	10.0	M55D000.050
	C-OLC360			0.65		
32	C-TSM270	Coalinga	5.2	0.37	10.4	M55D000.050
	C-TSM360			0.34		
33	C08050	Parkfield	6.1	0.59	9.2	M65D000.010
	C08320			0.54		
34	CAD250	Santa Barbara	6.0	0.12	36.6	M55D000.050
	CAD340			0.07		
35	CHY080-N	Chi-Chi, Taiwan	7.6	2.11	6.9	M75D000.010
	CHY080-W			2.75		
36	CLS000	Loma Prieta	6.9	1.33	5.1	M65D000.010
	CLS090			1.08		
37	CLS000	Loma Prieta	6.9	1.3	5.1	M75D000.010
	CLS090			1.21		
38	CLS220	Morgan Hill	6.2	0.16	22.7	M65D010.050
	CLS310			0.21		
39	CPE045	Victoria, Mexico	6.1	2.07	34.8	M65D010.050
	CPE315			1.62		
40	CPM000	Cape Mendocino	7.1	4.72	8.5	M75D000.010
	CPM090			3.24		
41	CSM095	San Fernando	6.6	0.05	86.6	M65D050.100
	CSM185			0.04		
42	CSM095	San Fernando	6.6	0.06	86.6	M65D100.200
	CSM185			0.04		

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
43	CUC090	Northridge	6.7	0.15	80.0	M65D050.100
	CUC180			0.1		
44	CUC090	Northridge	6.7	0.18	80.0	M65D100.200
	CUC180			0.12		
45	CYC195	Morgan Hill	6.2	1.61	0.1	M65D000.010
	CYC285			3.37		
46	D-OLC270	Coalinga	5.8	1.59	8.2	M55D000.050
	D-OLC360			0.72		
47	D-PLM270	Coalinga	5.8	0.48	12.2	M55D000.050
	D-PLM360			0.51		
48	DAY-LN	Tabas, Iran	7.4	0.99	17.0	M75D010.050
	DAY-TR			0.95		
49	DCF090	Lytle Creek	5.4	0.29	21.9	M55D000.050
	DCF180			0.29		
50	FER-L1	Tabas, Iran	7.4	0.28	94.4	M75D050.100
	FER-T1			0.36		
51	FSD172	San Fernando	6.6	0.34	27.5	M65D010.050
	FSD262			0.37		
52	G06000	Morgan Hill	6.2	0.47	11.8	M65D000.010
	G06090			0.61		
53	G06000	Morgan Hill	6.2	0.46	11.8	M65D010.050
	G06090			0.61		
54	GAZ000	Gazli, USSR	6.8	1.76	3.0	M65D000.010
	GAZ090			2.35		
55	GAZ000	Gazli, USSR	6.8	1.77	3.0	M75D000.010
	GAZ090			1.79		
56	GBZ000	Kocaeli, Turkey	7.4	0.45	17.0	M75D010.050
	GBZ270			0.34		
57	GIL067	Morgan Hill	6.2	0.28	16.2	M65D010.050
	GIL337			0.2		
58	GRN180	Landers	7.3	0.14	141.6	M75D100.200
	GRN270			0.12		
59	GYN000	Kocaeli, Turkey	7.4	0.31	35.5	M75D010.050
	GYN090			0.28		
60	H-PG3000	Coalinga	6.4	0.25	38.8	M65D010.050
	H-PG3090			0.22		

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
61	H-SUP045	Imperial Valley	6.5	0.23	26.0	M65D010.050
	H-SUP135			0.53		
62	H-TM2000	Coalinga	6.4	0.05	40.5	M65D010.050
	H-TM2090			0.08		
63	H-VC4000	Coalinga	6.4	0.11	34.6	M65D010.050
	H-VC4090			0.09		
64	H-XCV075	Mammoth Lakes	5.0	0.06	9.1	M55D000.050
	H-XCV165			0.11		
65	H-Z11000	Coalinga	6.4	0.19	28.4	M65D010.050
	H-Z11090			0.17		
66	H01000	N. Palm Springs	6.0	0.11	63.3	M55D050.100
	H01090			0.1		
67	HCP045	N. Palm Springs	6.0	0.51	34.9	M55D000.050
	HCP135			0.29		
68	HWA023-N	Chi-Chi, Taiwan	7.6	0.07	57.0	M75D050.100
	HWA023-W			0.07		
69	HWA026-N	Chi-Chi, Taiwan	7.6	0.14	58.8	M75D050.100
	HWA026-W			0.2		
70	HWA056-N	Chi-Chi, Taiwan	7.6	0.2	48.7	M75D010.050
	HWA056-W			0.21		
71	ILA031-N	Chi-Chi, Taiwan	7.6	0.15	94.7	M75D050.100
	ILA031-W			0.12		
72	ILA051-N	Chi-Chi, Taiwan	7.6	0.06	90.3	M75D050.100
	ILA051-W			0.15		
73	ILA063-N	Chi-Chi, Taiwan	7.6	0.22	71.6	M75D050.100
	ILA063-W			0.23		
74	ISD014	San Fernando	6.6	0.02	113.0	M65D100.200
	ISD284			0.02		
75	IZN180	Kocaeli, Turkey	7.4	0.24	29.7	M75D010.050
	IZN090			0.17		
76	IZT180	Kocaeli, Turkey	7.4	0.4	7.7	M75D000.010
	IZT090			0.35		
77	KAU078-N	Chi-Chi, Taiwan	7.6	0.07	102.8	M75D100.200
	KAU078-W			0.11		
78	KBU000	Kobe	6.9	0.6	0.2	M75D000.010
	KBU090			0.71		

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
79	KJM000	Northridge	6.7	2.12	0.6	M65D000.010
	KJM090			1.76		
80	KJM000	Kobe	6.9	1.92	0.6	M75D000.010
	KJM090			1.56		
81	KSH-L1	Tabas, Iran	7.4	0.14	199.1	M75D100.200
	KSH-T1			0.14		
82	L-BPL070	Mammoth Lakes	6.0	0.19	43.7	M65D010.050
	L-BPL160			0.26		
83	L04111	San Fernando	6.6	0.49	24.2	M65D010.050
	L04201			0.42		
84	LCN260	Landers	7.3	1.84	1.1	M75D000.010
	LCN345			2.25		
85	LGP000	Loma Prieta	6.9	1.65	6.1	M65D000.010
	LGP090			1.52		
86	LGP000	Loma Prieta	6.9	1.64	6.1	M75D000.010
	LGP090			1.64		
87	LMR162	N. Palm Springs	6.0	0.13	73.7	M55D050.100
	LMR252			0.12		
88	MA1130	San Fernando	6.6	0.02	115.0	M65D100.200
	MA1220			0.03		
89	MA2130	San Fernando	6.6	0.03	113.0	M65D100.200
	MA2220			0.02		
90	MA3130	San Fernando	6.6	0.02	113.0	M65D100.200
	MA3220			0.02		
91	MCD000	Kocaeli, Turkey	7.4	0.13	62.3	M75D050.100
	MCD090			0.13		
92	MDR000	Duzce, Turkey	7.1	0.2	34.6	M75D010.050
	MDR090			0.12		
93	MEL090	Northridge	6.7	0.15	51.6	M65D050.100
	MEL180			0.07		
94	MEL090	Northridge	6.7	0.19	51.6	M65D100.200
	MEL180			0.08		
95	MEL090	Landers	7.3	0.09	126.4	M75D100.200
	MEL180			0.06		
96	NSK-E	Chi-Chi, Taiwan	7.6	0.13	64.5	M75D050.100
	NSK-N			0.12		

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
97	PAC175	Northridge	6.7	1.38	8.0	M65D000.010
	PAC265			1.26		
98	PKC090	Northridge	6.7	0.75	8.2	M65D000.010
	PKC360			0.98		
99	PLC000	Landers	7.3	0.16	95.9	M75D050.100
	PLC090			0.16		
100	PLC258	N. Palm Springs	6.0	0.14	71.9	M55D050.100
	PLC348			0.12		
101	PNG-E	Chi-Chi, Taiwan	7.6	0.11	114.2	M75D100.200
	PNG-N			0.11		
102	PUL104	Northridge	6.7	3.94	8.0	M65D000.010
	PUL194			3.59		
103	RAN000	Northridge	6.7	0.2	55.2	M65D050.100
	RAN090			0.16		
104	RAN000	Northridge	6.7	0.24	55.2	M65D100.200
	RAN090			0.19		
105	RIV180	N. Palm Springs	6.0	0.12	71.1	M55D050.100
	RIV270			0.1		
106	RIV180	Northridge	6.7	0.18	101.3	M65D100.200
	RIV270			0.26		
107	RIV180	Landers	7.3	0.13	96.1	M75D050.100
	RIV270			0.15		
108	S1010	Nahanni	6.8	0.98	6.0	M65D000.010
	S1280			1.1		
109	S3270	Nahanni	6.8	0.15	16.0	M65D010.050
	S3360			0.14		
110	SER000	Northridge	6.7	0.1	79.5	M65D050.100
	SER270			0.1		
111	SER000	Northridge	6.7	0.12	79.5	M65D100.200
	SER270			0.12		
112	SER000	Landers	7.3	0.09	131.4	M75D100.200
	SER270			0.12		
113	SHL000	Cape Mendocino	7.1	0.65	33.8	M75D010.050
	SHL090			0.58		
114	SIL000	Landers	7.3	0.15	51.7	M75D050.100
	SIL090			0.14		

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
115	SKR180	Duzce, Turkey	7.1	0.05	42.7	M75D010.050
	SKR090			0.04		
116	SLO234	Southern Calif	6.0	0.07	70.0	M55D050.100
	SLO324			0.11		
117	SLO234	Parkfield	6.1	0.03	60.0	M65D050.100
	SLO324			0.03		
118	SOD015	San Fernando	6.6	0.16	58.1	M65D050.100
	SOD285			0.19		
119	SOD015	San Fernando	6.6	0.2	58.1	M65D100.200
	SOD285			0.23		
120	SON033	San Fernando	6.6	0.04	122.0	M65D100.200
	SON303			0.04		
121	SOR225	Northridge	6.7	0.15	54.1	M65D050.100
	SOR315			0.17		
122	SOR225	Northridge	6.7	0.19	54.1	M65D100.200
	SOR315			0.21		
123	SRO-NS	Fruili, Italy	5.5	0.04	17.9	M55D000.050
	SRO-WE			0.09		
124	SUL230	Landers	7.3	0.09	162.6	M75D100.200
	SUL320			0.08		
125	TAP035-N	Chi-Chi, Taiwan	7.6	0.22	96.8	M75D050.100
	TAP035-W			0.12		
126	TAP036-N	Chi-Chi, Taiwan	7.6	0.07	95.6	M75D050.100
	TAP036-W			0.05		
127	TAP059-N	Chi-Chi, Taiwan	7.6	0.09	125.9	M75D100.200
	TAP059-W			0.07		
128	TAP060-N	Chi-Chi, Taiwan	7.6	0.09	128.4	M75D100.200
	TAP060-W			0.08		
129	TAP067-N	Chi-Chi, Taiwan	7.6	0.09	104.2	M75D100.200
	TAP067-W			0.09		
130	TAP069-N	Chi-Chi, Taiwan	7.6	0.07	135.3	M75D100.200
	TAP069-W			0.06		
131	TAP072-N	Chi-Chi, Taiwan	7.6	0.16	110.0	M75D100.200
	TAP072-W			0.07		
132	TAP075-N	Chi-Chi, Taiwan	7.6	0.17	118.4	M75D100.200
	TAP075-W			0.2		

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
133	TAP078-N	Chi-Chi, Taiwan	7.6	0.09	131.0	M75D100.200
	TAP078-W			0.09		
134	TCU015-N	Chi-Chi, Taiwan	7.6	0.22	47.3	M75D010.050
	TCU015-W			0.19		
135	TCU025-N	Chi-Chi, Taiwan	7.6	0.12	54.3	M75D050.100
	TCU025-W			0.14		
136	TCU046-N	Chi-Chi, Taiwan	7.6	0.22	14.3	M75D010.050
	TCU046-W			0.34		
137	TCU047-N	Chi-Chi, Taiwan	7.6	1.17	33.0	M75D010.050
	TCU047-W			0.7		
138	TCU087-N	Chi-Chi, Taiwan	7.6	0.21	3.1	M75D000.010
	TCU087-W			0.25		
139	TCU089-N	Chi-Chi, Taiwan	7.6	0.44	8.2	M75D000.010
	TCU089-W			0.66		
140	TCU095-N	Chi-Chi, Taiwan	7.6	1.7	43.4	M75D010.050
	TCU095-W			0.78		
141	TCU120-N	Chi-Chi, Taiwan	7.6	0.38	8.1	M75D000.010
	TCU120-W			0.48		
142	TCU128-N	Chi-Chi, Taiwan	7.6	0.31	9.7	M75D000.010
	TCU128-W			0.27		
143	TCU136-E	Chi-Chi, Taiwan	7.6	0.28	8.9	M75D000.010
	TCU136-N			0.3		
144	TFS000	N. Palm Springs	6.0	0.23	73.2	M55D050.100
	TFS090			0.16		
145	TMB205	Parkfield	6.1	0.99	9.9	M65D000.010
	TMB295			0.69		
146	VIR200	Landers	7.3	0.06	194.1	M75D100.200
	VIR290			0.04		
147	WTW115	Lytle Creek	5.4	0.28	15.4	M55D000.050
	WTW205			0.32		
148	WWJ090	Northridge	6.7	0.11	68.4	M65D050.100
	WWJ180			0.08		
149	WWJ090	Northridge	6.7	0.16	68.4	M65D100.200
	WWJ180			0.12		

Chapter 5

Manuscript #3: A New Stress Reduction Coefficient Relationship for Liquefaction Triggering Analyses

The authors of the following technical note intend to submit it to ASCE's *Journal of Geotechnical and Geoenvironmental Engineering*.

Samuel Lasley made the following contributions to this technical note:

- wrote the first draft of the note under the general direction of the other two authors.
- conceived the functional form for the proposed r_d relationship and obtained all regression coefficients.
- Created all tables and figures except for the first figure.
- coded and performed all comparisons with the other r_d relationships

Dr. Green made the following contributions to this technical note:

- provided the impetus to examine and develop a new r_d relationship
- provided valuable feedback to the first drafts of note and polished the final draft

- wrote the section detailing the creation of CEUS motions by McGuire et al. (2001).
- created the first figure showing the soil column

Dr. Rodriguez-Marek made the following contributions to this technical note:

- provided guidance on regression and curve fitting. This included the use of a likelihood function and pseudo-mixed effects modeling (which was not included in the final version of the note).

A New Stress Reduction Coefficient Relationship for Liquefaction Triggering Analyses

Samuel J. Lasley[1], Russell A. Green[2], Adrian Rodriguez-Marek[3]

[1] Doctoral Candidate, Department of Civil and Environmental Engineering, Virginia Tech, Blacksburg, Virginia, U.S.A.; slasley@vt.edu

[2] Professor, Department of Civil and Environmental Engineering, Virginia Tech, Blacksburg, Virginia, U.S.A.; rugreen@vt.edu

[3] Professor, Department of Civil and Environmental Engineering, Virginia Tech, Blacksburg, Virginia, U.S.A.; adrianrm@vt.edu

5.1 Abstract

Since its inception in the early 1970s the stress-based simplified procedure has become the standard of practice worldwide for evaluating liquefaction triggering potential. Central to this procedure is the stress reduction coefficient, r_d, which allows the computation of the seismically induced stresses at depth in a soil profile without the need to perform a numerical site response analysis. Proposed herein is a new r_d relationship that was developed from equivalent-linear site response analyses performed on soil profiles representative of those in the liquefaction case history databases. The input motions used in the analyses are representative of those from shallow crustal earthquakes. Two variants of the r_d relationship are presented that allow it to be used when the profile's shear wave velocities are either known or unknown, with the former yielding values having less uncertainty. Additionally, calibration coefficients are provided for both active and stable continental tectonic regimes. In comparison with other r_d relationships that are commonly used, the relationship proposed herein yields values having less bias and uncertainty.

5.2 Introduction

Herein a new stress reduction coefficient, r_d, relationship is proposed that yields values that have less bias and uncertainty than those computed by commonly used relationships. The stress reduction coefficient is defined as the ratio of the maximum seismically induced shear stress at the base of a flexible soil column to that at the base of a rigid soil column, both having depth z (Seed and Idriss, 1971). Because the stress at the base of a rigid soil column can be computed using Newton's second law, r_d allows the computation of the seismically induced stresses at depth in a soil profile (i.e., flexible soil column) without the need to perform a numerical site response analysis (Figure 5.1). As a result, r_d is integral to the stress-based simplified procedure which is the worldwide standard of practice for evaluating liquefaction triggering potential.

The first r_d relationship was proposed by Seed and Idriss (1971), and several others have been proposed since then, with the vast majority of these being empirically based; Cetin (2000) provides a detailed review of some of these latter relationships. In the study presented herein, equivalent-linear site response analyses were performed to develop an empirically derived r_d relationship. The site response analyses were performed on 50 soil profiles compiled by Cetin (2000) that are representative of those in the liquefaction case history databases. However, a larger set of recorded input motions were used in the analyses than were available at the time Cetin (2000) performed his study. Furthermore, in addition to using the Cetin (2000) profiles directly, the soil profiles were also modified to be representative of conditions in the central-eastern United States (CEUS). The input motions used in the analysis of this latter set of profiles were from the McGuire et al. (2001) CEUS database. Statistical analysis of the site response results were used to develop two variants of the r_d relationship that allow it to be used when a profile's shear wave velocities are either known or unknown. Additionally, calibration coefficients are provided for both active and stable continental tectonic regimes (e.g., western United States: WUS and CEUS).

In the following, brief descriptions of the earthquake motion databases and the soil profiles are given. Next, the proposed r_d relationship is presented, followed by a comparison with commonly used relationships.

5.3 Earthquake Motions

The earthquake motions used in the site response analyses in this study came from two different ground motion databases. The first database is the PEER NGA strong motion database (Chiou et al., 2008) which is composed of 3551 multi-component records from 173 shallow crustal earthquakes. Of these 3551 records, 194 pairs of horizontal rock motions recorded during 47 earthquakes were used. The average shear wave velocities of the upper 30 m (V_{S30}) of the all sites where the motions were recorded are greater than 650 m/s. The earthquake magnitudes associated with the motions range from 5.3 to 7.6, and site-to-source distances (closest distance from the recording site to the fault rupture plane, Chiou et al. 2008) range from 1.8 to 152 km. These shallow crustal earthquake motions are representative of those generated by earthquakes in much of the WUS and are henceforth referred to as "WUS motions."

The second ground motion database used in this study is the McGuire et al. (2001) CEUS database. Because few CEUS earthquake motions have been recorded, the majority of the 149 pairs of motions in the CEUS database are scaled WUS motions. The earthquake magnitudes associated with these motions range from 4.5 to 7.6 and site-to-source distances range from 0.1 to 199 km. The recorded motions include the 1988 Saguenay ($Mw5.9$ mainshock and $Mw4.5$ aftershock), the 1985 Nahanni ($Mw6.8$), and the 1989 New Madrid, MO ($Mw4.7$) earthquakes; a brief summary of the scaling procedure used to generate the remaining CEUS motions is provided below.

McGuire et al. (2001) scaled WUS motions for CEUS conditions using response spectral transfer functions generated from the single-corner frequency point source model in conjunction with random vibration theory (RVT) (e.g., Brune 1970; 1971; Boore 1983; McGuire et al. 2001; Silva and Lee 1987). The transfer functions account for the differences in seismic source, wave propagation path properties, and generic site effects between the WUS and CEUS regions. Many seismological publications have shown successful results of the RVT point source model for generating strong ground motions for both WUS and CEUS (Boore 1983; 1986; Hanks and McGuire 1981; McGuire et al. 1984; Schneider et al. 1993; Silva 1993). In generating the scaled CEUS motions, recorded WUS motions were used as "seed" motions in the spectral scaling process, resulting in scaled motions that have realistic characteristics. In this context, the stochastic point source model is a reliable and reasonable

approach for estimating spectral characteristics of strong ground motions for engineering analyses. The scaling method however, should be validated as additional recordings of stable continental motions become available.

Tables 5.5 and 5.6 in the electronic supplement list the WUS and CEUS motions, respectively, used in the analyses. Also, Figure 5.2 shows the distribution of earthquake events as a function of magnitude and distance for both databases.

5.4 Soil Profiles

The soil profiles used in the equivalent-linear site response analyses were compiled by Cetin (2000). The 50 well-characterized profiles are from post-earthquake site investigations in California. For each profile, Cetin provides a qualitative description of the layers (e.g. clay, fine sand) and the shear wave velocities. Additional soil layer details (unit weight, plasticity index, and at-rest lateral earth pressure coefficient) were randomly selected from a distribution of probable values corresponding to the qualitative description. The profiles were analyzed as-is with the WUS input motions. For simulated conditions representative of those in the CEUS, the shear wave velocity of the bedrock of each profile was increased by a factor of 3.5 and analyzed using the CEUS input motions. This factor of 3.5 brought the bedrock shear wave velocities into agreement with higher bedrock shear wave velocities of the CEUS. Several of the profiles had similar characteristics; Figures 5.3 and 5.4 show the grouped profiles for the WUS and CEUS, respectively.

5.5 Proposed r_d Relationship

Several functional forms for r_d were examined, and the following form was selected for its simplicity and shape (i.e., relatively low standard deviation of the regressed data):

$$r_d = (1 - \alpha) \exp\left(\frac{-z}{\beta}\right) + \alpha \tag{5.1}$$

where α is the limiting value of r_d at large depths and can range from 0 to 1. The variable β controls the curvature of the function at shallow depths, and z is the depth in meters. The term '$(1 - \alpha)$' scales the exponential so that r_d is equal to one at the ground surface.

For each combination of profile group and earthquake event, values of α and β were obtained using a non-linear least-squares curve-fitting algorithm (Jones et al. 2001). From the results of the curve-fitting algorithm, regressions were performed to predict α and β using magnitude (M_w), a_{max}, average shear wave velocity in the upper 12 meters of the profile (V_{S12}), V_{S30}, and distance to the fault rupture plane (R). Of these predictors, M_w and V_{S12} were found to be most strongly correlated with α and β.

Two different sets of expressions for α and β are proposed, one set being a function of M_w and V_{S12} and the other being soley a function of M_w. This allows the use of the r_d relationship for profiles having varying levels of characterization. The first set of expressions for α and β is:

$$\alpha_1 = \exp(b_1 + b_2 M_w + b_3 V_{S12}) \tag{5.2}$$

$$\beta_1 = \exp(b_4 + b_5 M_w + b_6 V_{S12}) \tag{5.3}$$

and the second set is:

$$\alpha_2 = \exp(b_1 + b_2 M_w) \tag{5.4}$$

$$\beta_2 = b_3 + b_4 M_w \tag{5.5}$$

where b_1 - b_6 are regression coefficients.

While the functional forms of the above two sets of expressions were developed using the least-squares curve fitting, the regression coefficients (Tables 5.1 and 5.2) were obtained by maximizing the likelihood, L, defined as:

$$L = \sum \left[\ln \left(\frac{1}{\sigma\sqrt{2\pi}} \right) - \frac{(r_{d,\text{ Predicted}} - r_{d,\text{ Actual}})^2}{2\sigma^2} \right] \tag{5.6}$$

where σ is an error parameter regressed as part of the likelihood maximization. As with any empirical relationship, care should be used when applying these equations for conditions outside the ranges from which they were regressed. For the earthquake databases, magnitudes ranged from 5.3 to 7.6 and 4.5 to 7.6, for the WUS and CEUS, respectively. Average shear wave velocities in the upper 12 meters ranged from 130 to 220 m/s for both data sets. In particular, erroneous values will result when β is less than or equal to zero.

5.6 Discussion

Figure 5.5 shows the trends in the proposed r_d relationship (using the second set of expressions for α and β) for magnitudes of 5.5 and 7.5 along with the r_d values predicted by a few commonly used r_d relationships. The Liao and Whitman (1986) relationship is solely a function of depth and was adopted for use in the in Youd et al. (2001) liquefaction evaluation procedures, which are widely used in practice. Cetin (2000) proposed two variants of an r_d relationship (Cetin-1 and Cetin-2), Cetin-1 being a function of M_w, a_{max}, depth and Cetin-2 being a function of M_w, a_{max}, V_{S12}, and depth. These relationships were adopted for use in the Cetin et al. (2004), Moss et al. (2006), and Kayen et al. (2013) liquefaction evaluation procedures. The Idriss (1999) r_d relationship is a function of M_w and depth and was adopted for use in the Idriss and Boulanger liquefaction evaluation procedures.

Relative to the relationship proposed herein, all three of the r_d relationships discussed above yield high values at shallow depths. Furthermore, the Idriss (1999) and Liao and Whitman (1986) relationships predict higher values of r_d for almost all depths shown relative to the proposed relationship. The Cetin (2000) relationship is less dependent on earthquake magnitude than the proposed relationship and yields values that fall in between the M5.5 and M7.5 curves of the proposed relationship; the proposed relationship predicts a greater range of r_d values overall. Because the proposed relationship generally predicts lower values of r_d, the use of this relationship will predict smaller values of τ_{max} at depth in a profile for a given value of a_{max}.

Figure 5.6 shows the predicted values of the relationships above a heat map of the actual r_d values from the WUS site response analyses. In the top row of the figure are the results from the proposed relationship using the two

different sets of expressions for α and β along with the results from the Liao and Whitman (1986) relationship. At the bottom of the figure are the Idriss (1999) and Cetin (2000) 1 & 2 relationships. The value E in the upper left corner of each subplot is the Nash-Sutcliffe (1970) model efficiency coefficient. For these comparisons, it is calculated as follows:

$$E = 1 - \frac{\sum(r_{d,\text{Actual}} - r_{d,\text{Predicted}})^2}{\sum(r_{d,\text{Actual}} - \overline{r_{d,\text{Actual}}})^2} \tag{5.7}$$

where $r_{d,\text{Actual}}$ are the values given by the results of the site response analyses, $r_{d,\text{Predicted}}$ are the values predicted by the relationship in question, and $\overline{r_{d,\text{Actual}}}$ is the mean of the $r_{d,\text{Actual}}$ values. The model efficiency coefficient value can range from 1 to $-\infty$. A value of 1 signifies that the model perfectly predicts; values of E less than zero signify that the mean value of $r_{d,\text{Actual}}$ is a better predictor than the model. Based on the model efficiency coefficient and the visual spread of the data points, the proposed r_d relationship performs better than the other relationships, with the Liao and Whitman (1986) and Cetin (2000) relationships having the poorest and best performances, respectively, of the three. The relatively good performance of the Cetin (2000) relationships is not surprising since they were derived from a more comprehensive site response study than the others.

From Figure 5.6 it appears that the relationship proposed herein gives poorer predictions at the shallower depths (<10m) where occurrence of liquefaction has been more pronounced. However, to test this, E values were computed for each of the relationships for 2-m depth increments ranging from 2 m to 12 m, as well as a 2 to 8 m increment. The results of this for WUS analyses are presented in Table 5.3. As may be observed from the tabulated values, the relationship proposed herein performs better than the others at all depths.

As shown in Figure 5.5, the CEUS r_d values tend to be lower than comparable WUS values. In comparing the predicted CEUS values using the relationship proposed herein to those predicted by the other relationships (Figure 5.7 and Table 5.4), the relative accuracy of the other relationships is poorer than when they are used for WUS conditions. This is not surprising given all the other relationships were developed using data that is representative of the WUS conditions.

Finally, although few liquefaction events have been recorded for depths greater than 20 m and the use of the simplified procedure may be question-

able at depths greater than this, the relationship proposed herein provides reasonable r_d values at depths up to and exceeding 100 meters.

5.7 Conclusion

Based on a large number of recent equivalent-linear site response analyses, a new relationship for the stress reduction coefficient (r_d) has been proposed. r_d plays an important role in the stress-based simplified liquefaction evaluation procedure because it allows the computation of the seismically induced stresses at depth in a soil profile without the need to perform a numerical site response analysis. Two different sets of expressions are presented for α and β, one being a function of M_w and V_{S12} and the other solely being a function of M_w. This allows the use of the r_d relationship for profiles having varying levels of characterization. Additionally, calibration coefficients are provided for both active and stable continental tectonic regimes. Compared to the r_d relationships proposed by Liao and Whitman (1986), Idriss (1999), and Cetin (2000), the relationship proposed herein yields values with less bias and uncertainty at all depths.

5.8 References

Boore, D.M. (1983). "Stochastic simulation of high-frequency ground motions based on seismological models of the radiated spectra." *B. Seismol. Soc. Am.*, 73(6A), 1865-1894.

Boore, D.M. (1986). "Short-period P- and S-wave radiation from large earthquakes: implications for spectral scaling relations", *B. Seismol. Soc. Am.*, 76(1), 43-64.

Brune, J.N. (1970). "Tectonic stress and spectra of seismic shear waves from earthquakes", *J. Geophys. Res.*, 75(26), 611-614.

Brune, J.N. (1971). "Correction", *J. Geophys. Res.*, 76(20), 1441-1450.

Boulanger, R.W., and Idriss, I.M. (2014). CPT and SPT based liquefaction triggering procedures. Report No. UCD/CGM-14.

Cetin, K. O. (2000). "Reliability-based assessment of seismic soil liquefaction initiation hazard." University of California, Berkeley, Berkeley, CA.

Cetin, K. O., Seed, R. B., Der Kiureghian, A., Tokimatsu, K., Harder, L. F., Kayen, R. E., and Moss, R. E. S. (2004). "Standard Penetration Test-Based Probabilistic and Deterministic Assessment of Seismic Soil Liquefaction Potential." *Journal of Geotechnical and Geoenvironmental Engineering*, 130(12), 1314–1340.

Chiou, B., Darragh, R., Gregor, N., and Silva, W. (2008). "NGA Project Strong-Motion Database." *Earthquake Spectra*, 24(1), 23-44.

Hanks, T.C. and McGuire, R.K. (1981). "The character of high-frequency strong ground motion." *B. Seismol. Soc. Am.*, 71(6), 2071-2095.

Idriss I. M. (1999) "An update to the Seed-Idriss simplified procedure for evaluating liquefaction potential.", *Proc., TRB Workshop on New Approaches to Liquefaction*, Publication No. FHWA-RD-99- 165, Federal Highway Administration.

Idriss, I. M., and Boulanger, R. W. (2008). *Soil Liquefaction During Earthquakes*. Earthquake Engineering Research Institute, Oakland, California.

Jones E., Oliphant E., Peterson P., et al. (2001). *SciPy: Open Source Scientific Tools for Python*, http://www.scipy.org/ [Online; accessed 2015-05-18]

Kayen, R., Moss, R.E.S., Thompson, E.M., Seed, R.B., Cetin, K.O., Der Kiureghian, A., Tanaka, Y., and Tokimatsu, K. (2013). "Shear-wave velocity–based probabilistic and deterministic assessment of seismic soil liquefaction potential." *Journal of Geotechnical and Geoenvironmental Engineering*, 139(3), 407–419.

Liao, S. S. C., and Whitman, R. V. (1986). "Catalogue of liquefaction and non-liquefaction occurrences during earthquakes." Res. Rep., Dept. of Civ. Engrg., Massachusetts Institute of Technology, Cambridge, Mass.

McGuire, R.K., Becker, A.M. and Donovan, N.C. (1984). "Spectral estimates of seismic shear waves", *Bull. Seismol. Soc. Am.*, 74(4), 1427-1440.

McGuire, R. K., Silva, W. J., and Costantino, C. J. (2001). Technical Basis for Revision of Regulatory Guidance on Design Ground Motions: Hazard- and Risk-consistent Ground Motion Spectra Guidelines. Division of Engi-

neering Technology, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, Washington, D.C.

Moss, R.E., Seed, R.B., Kayen, R.E., Stewart, J.P., Der Kiureghian, A., and Cetin, K.O. (2006). "CPT-based probabilistic and deterministic assessment of in situ seismic soil liquefaction potential." *Journal of Geotechnical and Geoenvironmental Engineering*, 132(8), 1032–1051.

Nash, J. E., and Sutcliffe, J. V. (1970). "River flow forecasting through conceptual models part I—A discussion of principles." *Journal of Hydrology*, 10(3), 282-290.

Schneider, J.F., Silva, W.J. and Stark, C. (1993). "Ground motion model for the 1989 M 6.9 Loma Prieta earthquake including effects of source, path, and site", *Earthq. Spectra*, 9(2), 251-287.

Seed, H. B., and Idriss, I. M. (1971). "Simplified procedure for evaluating soil liquefaction potential." *Journal of the Soil Mechanics and Foundations Division*, 97(9), 1249-1273.

Silva, W. J., and Lee, K. (1987). "WES RASCAL code for synthesizing earthquake ground motions: state-of-the-art for assessing earthquake hazards in the United States." Report 24, US Army Engineering Waterways Experiment Station Vicksburg, MS.

Silva, W.J. (1993). "Factors controlling strong ground motion and their associated uncertainties", Dynamic Analysis and Design Considerations for High-Level Nuclear Waste Repositories, San Francisco, CA, USA, 132-161.

Youd, T. L., Idriss, I. M., Andrus, R. D., Arango, I., Castro, G., Christian, J. T., Dobry, R., Finn, W. D. L., Harder, L. F., Hynes, M. E., Ishihara, K., Koester, J. P., Liao, S. S. C., Marcuson, W. F., Martin, G. R., Mitchell, J. K., Moriwaki, Y., Power, M. S., Robertson, P. K., Seed, R. B., and Stokoe, K. H. (2001). "Liquefaction Resistance of Soils: Summary Report from the 1996 NCEER and 1998 NCEER/NSF Workshops on Evaluation of Liquefaction Resistance of Soils." *Journal of Geotechnical and Geoenvironmental Engineering*, 127(10), 817-833.

5.9 Tables

Table 5.1: Regression Coefficients of the r_d Relationship for the WUS Database

Form	b_1	b_2	b_3	b_4	b_5	b_6	σ
1	-3.677	0.3454	-0.001303	-1.532	0.3868	0.01213	0.1351
2	-4.031	0.364	-28.47	7.798			0.1397

Table 5.2: Regression Coefficients of the r_d Relationship for the CEUS Database

Form	b_1	b_2	b_3	b_4	b_5	b_6	σ
1	-3.863	0.39	-0.001049	-1.439	0.3721	0.01056	0.1551
2	-4.369	0.439	-15.38	4.922			0.1586

Table 5.3: Values of E for the WUS Dataset

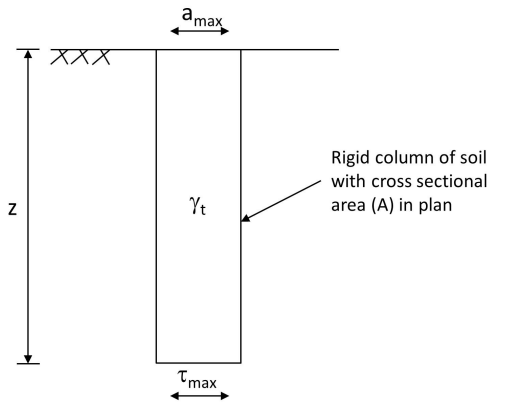
Depths	This Study-1	This Study-2	Liao and Whitman	Idriss	Cetin-1	Cetin-2
2-4m	0.28	0.16	-0.42	-0.23	-0.00	0.06
4-6m	0.28	0.15	-0.86	-0.47	-0.07	-0.03
6-8m	0.28	0.16	-1.28	-0.59	-0.00	0.00
8-10m	0.29	0.19	-1.66	-0.64	0.08	0.09
10-12m	0.29	0.19	-1.59	-0.62	0.12	0.15
2-8m	0.39	0.28	-0.67	-0.26	0.12	0.15
0-20m	0.62	0.57	-0.19	0.19	0.51	0.53

Table 5.4: Values of E for the CEUS Dataset

Depths	This Study-1	This Study-2	Liao and Whitman	Idriss	Cetin-1	Cetin-2
2-4m	0.24	0.13	-0.55	-0.43	-0.12	-0.11
4-6m	0.20	0.09	-1.02	-0.72	-0.19	-0.23
6-8m	0.20	0.11	-1.41	-0.85	-0.09	-0.18
8-10m	0.23	0.16	-1.68	-0.85	0.02	-0.04
10-12m	0.22	0.18	-1.46	-0.75	0.06	0.03
2-8m	0.31	0.21	-0.83	-0.50	-0.00	-0.04
0-20m	0.54	0.50	-0.31	-0.01	0.40	0.39

5.10 Figures

Maximum seismically induced shear stress at the base of a flexible soil column:



$$F_{max} = m \cdot a_{max}$$

$$= z \cdot A \cdot \frac{\gamma_t}{g} \cdot a_{max}$$

$$\tau_{rigid\ max} = \frac{F_{max}}{A} = \frac{a_{max}}{g} \cdot \sigma_v$$

$$r_d = \frac{\tau_{flexible\ max}}{\tau_{rigid\ max}}$$

$$\tau_{flexible\ max} = \frac{a_{max}}{g} \cdot \sigma_v \cdot r_d$$

F_{max} = maximum seismically induced inertial force in rigid soil column
m = mass of soil column of having a cross-sectional area A and length z
a_{max} = peak ground acceleration at the surface of the soil profile
γ_t = total unit weight of soil
g = coefficient of acceleration due to gravity
σ_v = total vertical stress at the base of the soil column
$\tau_{rigid\ max}$ = maximum seismically induced shear stress at the base of a rigid soil column
$\tau_{flexible\ max}$ = maximum seismically induced shear stress at the base of a flexible soil column

Figure 5.1: Stress reduction coefficient (r_d) is defined as the ratio of the maximum seismically induced shear stress at the base of a flexible soil column to that at the base of a rigid soil column, both having length z.

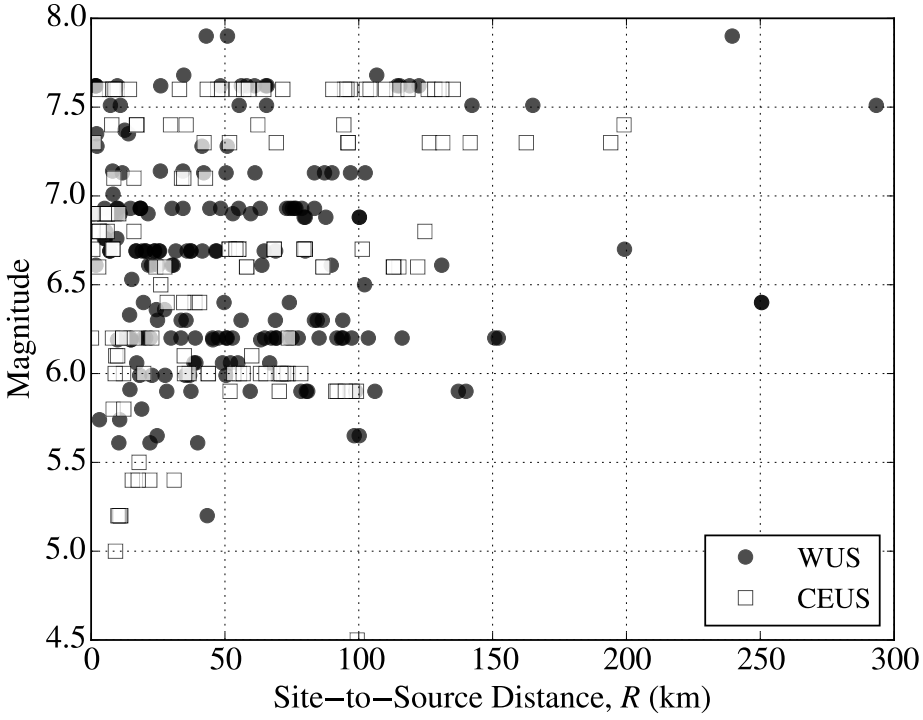


Figure 5.2: Magnitude versus distance distribution of the motions used in this study.

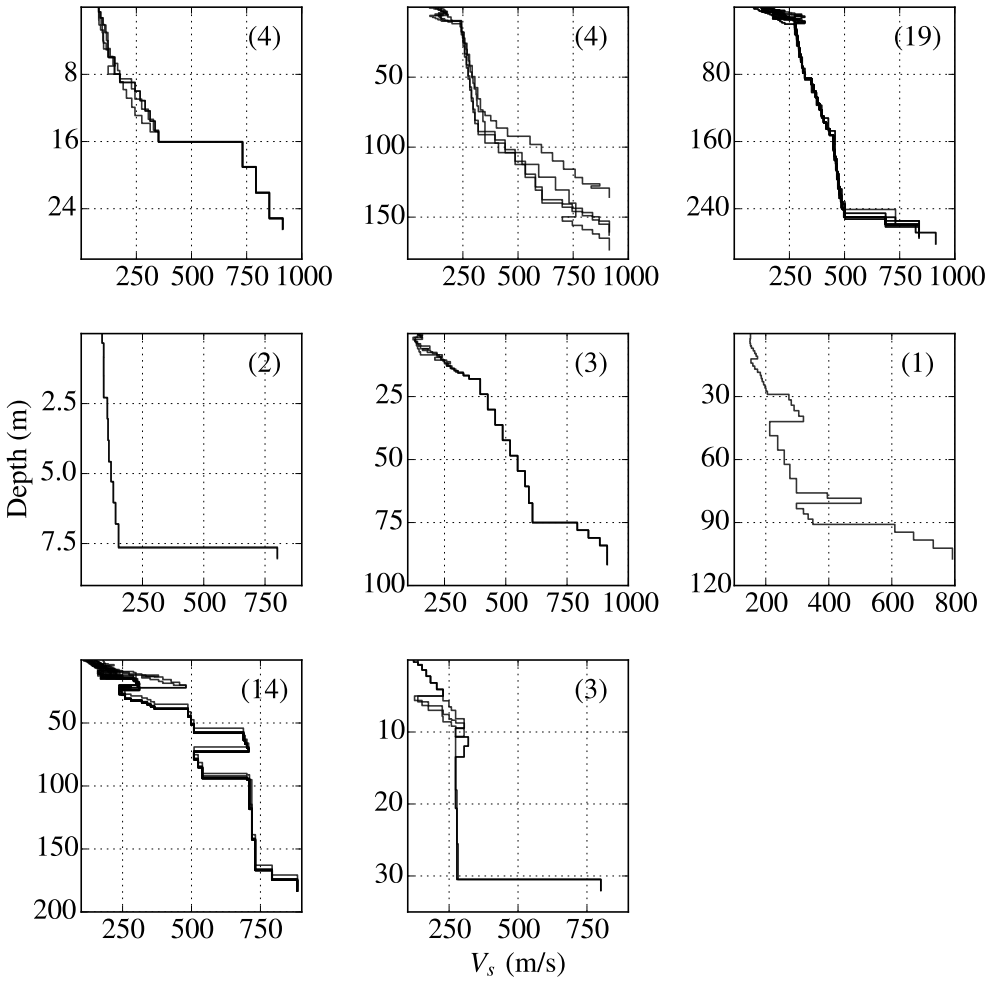


Figure 5.3: WUS profiles used in this study. The number in the corner of each subplot is the number of profiles in that grouping.

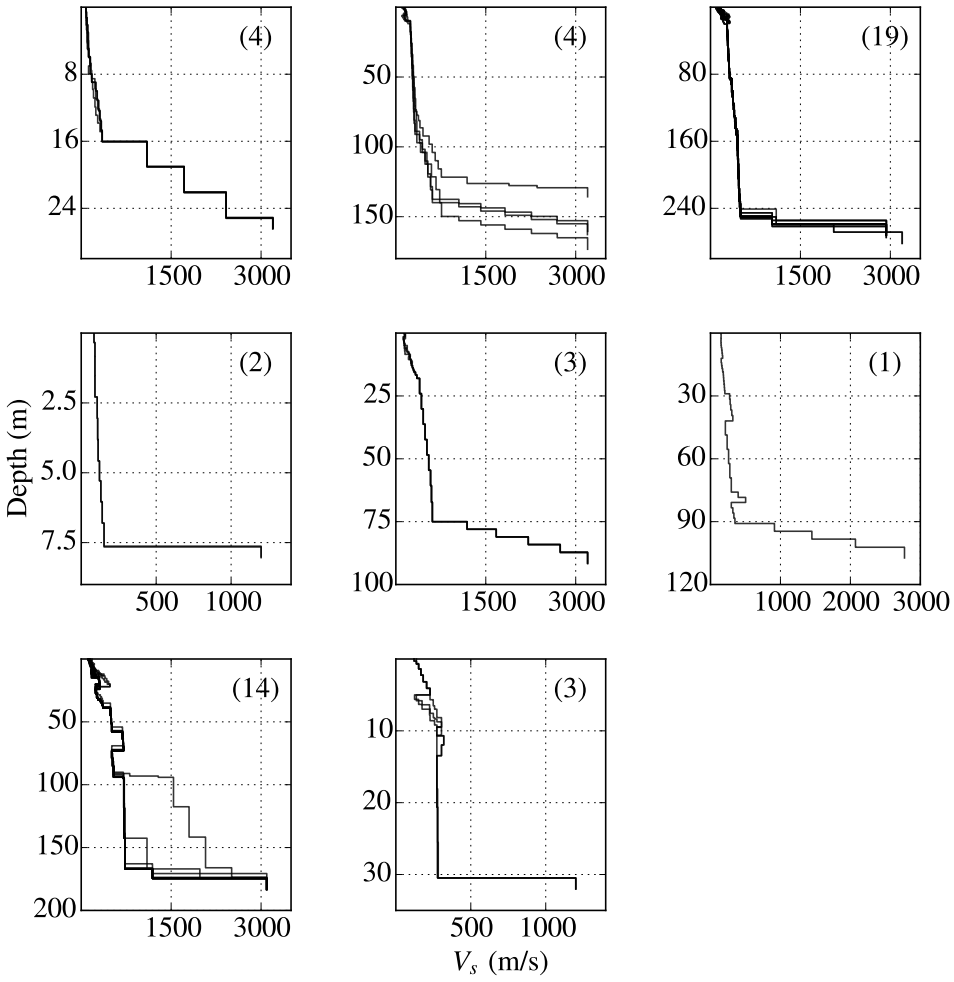


Figure 5.4: CEUS profiles used in this study. The number in the corner of each subplot is the number of profiles in that grouping.

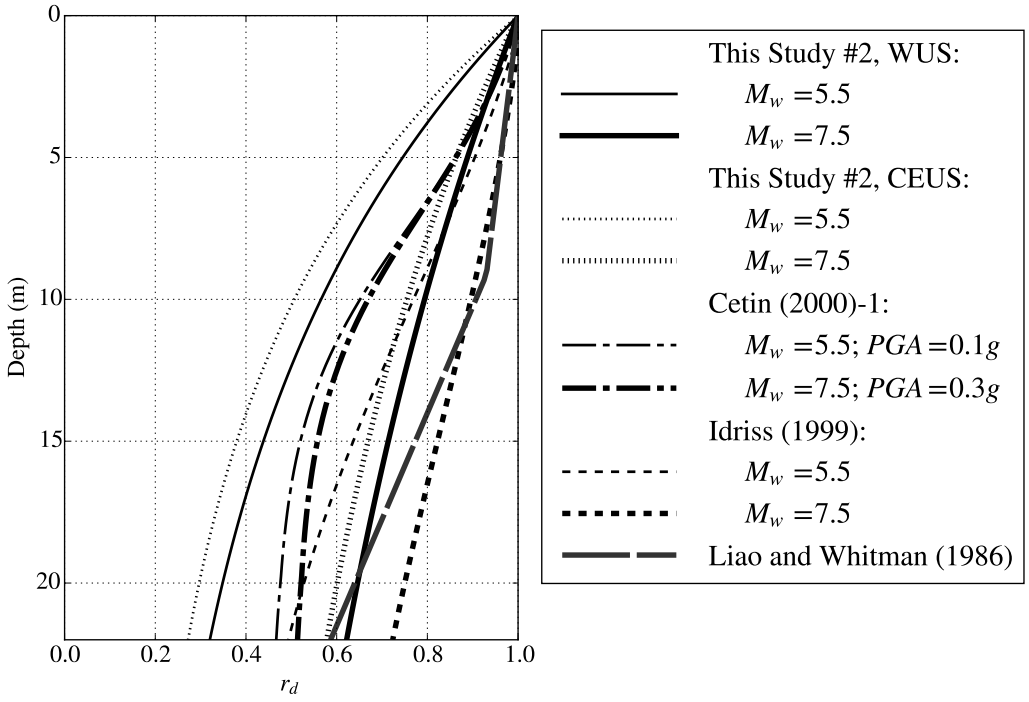


Figure 5.5: The proposed models at magnitudes of 5.5 and 7.5, compared with Idriss (1999), Liao and Whitman (1986), and Cetin (2000).

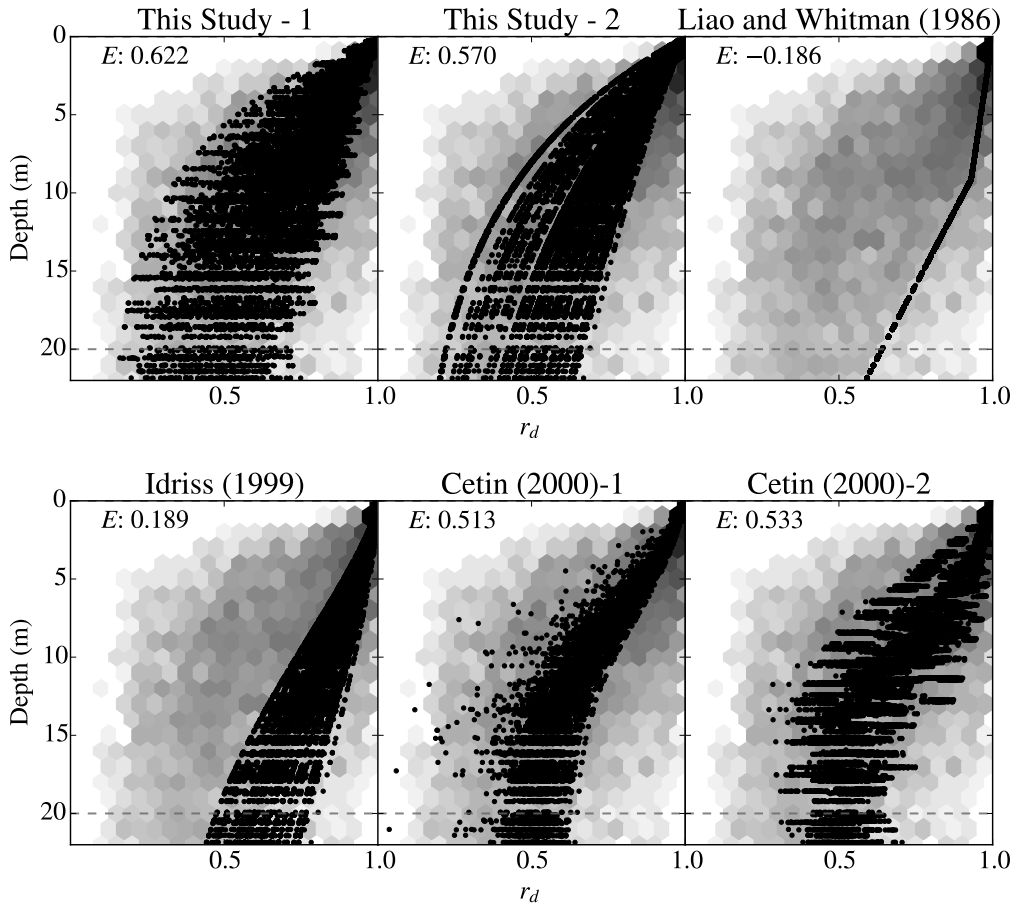


Figure 5.6: Predicted r_d values from several predictive equations above a heat map showing the density of $r_{d,\,\text{Actual}}$ values from the WUS dataset. The value E is the Nash-Sutcliffe (1970) model efficiency coefficient.

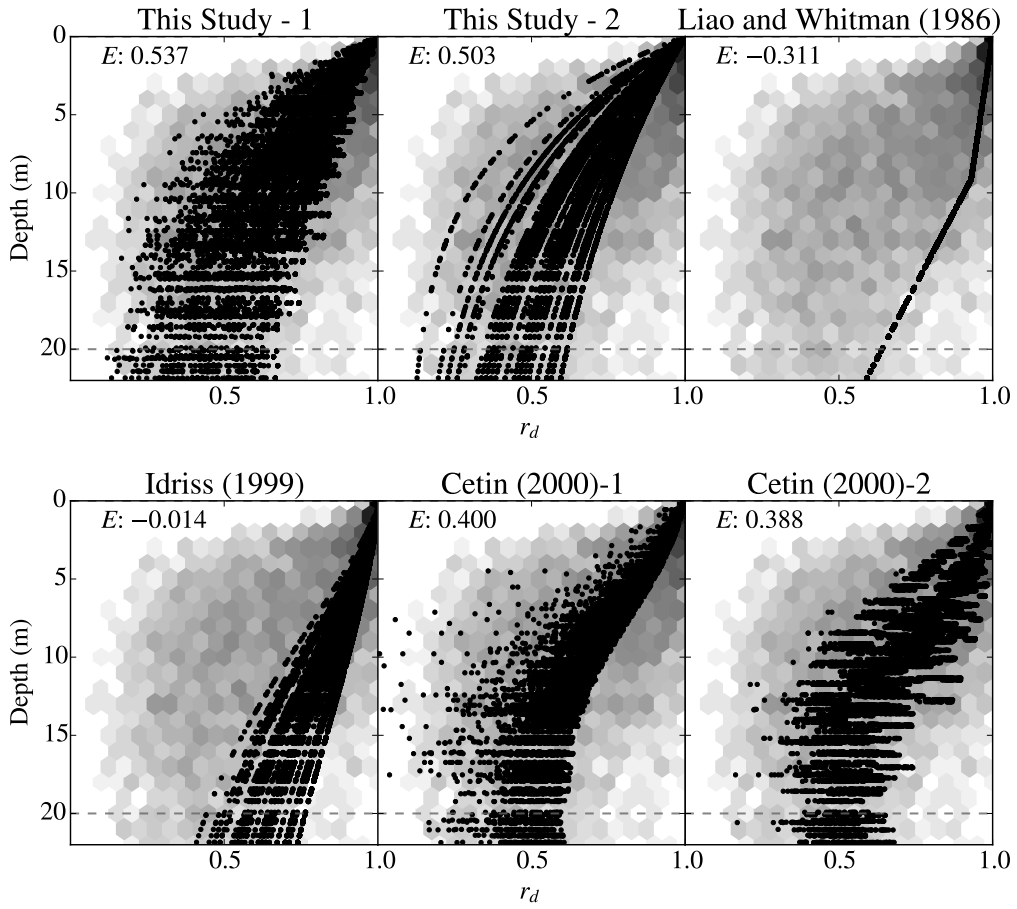


Figure 5.7: Predicted r_d values above a heat map showing the density of $r_{d,\,\text{Actual}}$ values from the CEUS dataset. The value E is the Nash-Sutcliffe (1970) model efficiency coefficient.

5.11 Electronic Supplement

5.11.1 NGA Database (WUS)

The distance given in the tables, R, is the closest distance to the rupture area (an *nan* indicates that the value is not known). The units of PGA and R are g and km, respectively.

Table 5.5: WUS (NGA) Earthquake Motions

No.	Names	Event	M_w	PGA	R	Mechanism
1	1011_WON095 1011_WON185	Northridge -01	6.69	0.13	20.3	Reverse
2	1012_LA0000 1012_LA0090	Northridge -01	6.69	0.32	19.1	Reverse
3	1021_L04000 1021_L04090	Northridge -01	6.69	0.08	31.7	Reverse
4	1023_L09090 1023_L09000	Northridge -01	6.69	0.17	25.4	Reverse
5	1027_LV1000 1027_LV1090	Northridge -01	6.69	0.08	37.2	Reverse
6	1029_LV3000 1029_LV3090	Northridge -01	6.69	0.09	37.3	Reverse
7	1033_LIT090 1033_LIT180	Northridge -01	6.69	0.07	46.6	Reverse
8	1041_MTW000 1041_MTW090	Northridge -01	6.69	0.17	35.9	Reverse
9	1050_PAC265 1050_PAC175	Northridge -01	6.69	0.41	7.0	Reverse
10	1051_PUL104 1051_PUL194	Northridge -01	6.69	1.43	7.0	Reverse
11	1060_CUC090 1060_CUC180	Northridge -01	6.69	0.06	80.0	Reverse
12	1074_SAN090 1074_SAN180	Northridge -01	6.69	0.09	41.6	Reverse
13	1078_5108-090 1078_5108-360	Northridge -01	6.69	0.25	16.7	Reverse

No.	Names	Event	M_w	PGA	R	Mechanism
14	1091_VAS000 1091_VAS090	Northridge -01	6.69	0.14	23.6	Reverse
15	1096_WWJ090 1096_WWJ180	Northridge -01	6.69	0.05	64.7	Reverse
16	1142_IZ1090 1142_IZ1000	Dinar, Turkey	6.4	0.0	250.5	Normal
17	1154_BRS090 1154_BRS180	Kocaeli, Turkey	7.51	0.06	65.5	Strike-Slip
18	1159_ERG090 1159_ERG180	Kocaeli, Turkey	7.51	0.1	142.3	Strike-Slip
19	1165_IZT090 1165_IZT180	Kocaeli, Turkey	7.51	0.2	7.2	Strike-Slip
20	1168_MNS000 1168_MNS090	Kocaeli, Turkey	7.51	0.01	293.4	Strike-Slip
21	1169_MSK000 1169_MSK090	Kocaeli, Turkey	7.51	0.04	55.3	Strike-Slip
22	1172_TKR090 1172_TKR180	Kocaeli, Turkey	7.51	0.04	165.0	Strike-Slip
23	124_A-FLT000 124_A-FLT270	Friuli, Italy-01	6.5	0.03	102.2	Reverse
24	1257_HWA003-W 1257_HWA003-N	Chi-Chi, Taiwan	7.62	0.09	56.1	Reverse-Oblique
25	126_GAZ000 126_GAZ090	Gazli, USSR	6.8	0.64	5.5	Unknown
26	133_B-SRO000 133_B-SRO270	Friuli, Italy-02	5.91	0.1	14.5	Reverse
27	1347_ILA063-N 1347_ILA063-W	Chi-Chi, Taiwan	7.62	0.09	61.1	Reverse-Oblique
28	1352_KAU003-N 1352_KAU003-W	Chi-Chi, Taiwan	7.62	0.02	114.4	Reverse-Oblique
29	139_DAY-LN 139_DAY-TR	Tabas, Iran	7.35	0.35	13.9	Reverse
30	143_TAB-LN 143_TAB-TR	Tabas, Iran	7.35	0.81	2.0	Reverse
31	1440_TAP065-E 1440_TAP065-N	Chi-Chi, Taiwan	7.62	0.03	122.5	Reverse-Oblique

No.	Names	Event	M_w	PGA	R	Mechanism
32	1446_TAP077-N 1446_TAP077-W	Chi-Chi, Taiwan	7.62	0.03	119.0	Reverse-Oblique
33	146_G01230 146_G01320	Coyote Lake	5.74	0.12	10.7	Strike-Slip
34	150_G06230 150_G06320	Coyote Lake	5.74	0.4	3.1	Strike-Slip
35	1518_TCU085-E 1518_TCU085-N	Chi-Chi, Taiwan	7.62	0.06	58.1	Reverse-Oblique
36	1529_TCU102-E 1529_TCU102-N	Chi-Chi, Taiwan	7.62	0.24	1.5	Reverse-Oblique
37	155_F-BEV-EW 155_F-BEV-NS	Norcia, Italy	5.9	0.03	nan	Normal
38	1551_TCU138-N 1551_TCU138-W	Chi-Chi, Taiwan	7.62	0.21	9.8	Reverse-Oblique
39	156_F-CSC-EW 156_F-CSC-NS	Norcia, Italy	5.9	0.19	nan	Normal
40	1577_TTN025-E 1577_TTN025-N	Chi-Chi, Taiwan	7.62	0.04	65.8	Reverse-Oblique
41	1585_TTN040-N 1585_TTN040-W	Chi-Chi, Taiwan	7.62	0.03	48.3	Reverse-Oblique
42	1587_TTN042-N 1587_TTN042-W	Chi-Chi, Taiwan	7.62	0.06	65.3	Reverse-Oblique
43	1613_1060-E 1613_1060-N	Duzce, Turkey	7.14	0.04	25.9	Strike-Slip
44	1618_531-E 1618_531-N	Duzce, Turkey	7.14	0.14	8.0	Strike-Slip
45	1619_MDR000 1619_MDR090	Duzce, Turkey	7.14	0.09	34.3	Strike-Slip
46	1626_212V5180 1626_212V5090	Sitka, Alaska	7.68	0.09	34.6	Strike-Slip
47	164_H-CPE147 164_H-CPE237	Imperial Valley-06	6.53	0.18	15.2	Strike-Slip
48	1645_mtwin000 1645_mtwin090	Sierra Madre	5.61	0.23	10.4	Reverse
49	1649_vquez000 1649_vquez090	Sierra Madre	5.61	0.11	39.8	Reverse

No.	Names	Event	M_w	PGA	R	Mechanism
50	1691_ANACA000	Northridge	5.28	0.01	nan	Reverse
	1691_ANACA270	-06				
51	1696_HOW060	Northridge	5.28	0.07	nan	Reverse
	1696_HOW330	-06				
52	1709_0GPN00E	Northridge	5.28	0.04	nan	Reverse
	1709_0GPN90W	-06				
53	1715_WON095	Northridge	5.28	0.05	nan	Reverse
	1715_WON185	-06				
54	1718_LITTL090	Northridge	5.28	0.01	nan	Reverse
	1718_LITTL360	-06				
55	1720_MCS025	Northridge	5.28	0.05	nan	Reverse
	1720_MCS115	-06				
56	1727_RANCH180	Northridge	5.28	0.01	nan	Reverse
	1727_RANCH090	-06				
57	1741_Lsm2000	Little	5.65	0.1	24.7	Normal
	1741_Lsm2270	Skull				
	1741_Lsm2270	Mtn,NV				
58	1745_Lsm6000	Little	5.65	0.01	100.2	Normal
	1745_Lsm6270	Skull				
	1745_Lsm6270	Mtn,NV				
59	1747_Lsm8270	Little	5.65	0.01	98.3	Normal
	1747_Lsm8000	Skull				
	1747_Lsm8000	Mtn,NV				
60	1767_12674090	Hector	7.13	0.02	83.4	Strike-Slip
	1767_12674360	Mine				
61	1786_22T04090	Hector	7.13	0.08	61.2	Strike-Slip
	1786_22T04180	Mine				
62	1795_12647090	Hector	7.13	0.08	50.4	Strike-Slip
	1795_12647180	Mine				
63	1836_22161090	Hector	7.13	0.06	42.1	Strike-Slip
	1836_22161360	Mine				
64	1917_02467360	Anza-02	4.92	0.01	nan	Normal-Oblique
	1917_02467090					
65	1942_12116036	Anza-02	4.92	0.06	nan	Normal-Oblique
	1942_12116126					
66	1961_13095026	Anza-02	4.92	0.03	nan	Normal-Oblique

No.	Names	Event	M_w	PGA	R	Mechanism
	1961_13095116					
67	1972_RVB090	Anza-02	4.92	0.01	nan	Normal-Oblique
	1972_RVB360					
68	2017_2017090	Gilroy	4.9	0.01	nan	Strike-Slip
	2017_2017360					
69	2019_47006067	Gilroy	4.9	0.23	nan	Strike-Slip
	2019_47006337					
70	2021_57383090	Gilroy	4.9	0.09	nan	Strike-Slip
	2021_57383360					
71	2032_2014090	Gilroy	4.9	0.01	nan	Strike-Slip
	2032_2014180					
72	2046_0438090	Gilroy	4.9	0.02	nan	Strike-Slip
	2046_0438360					
73	2091_ps07039	Nenana	6.7	0.01	199.3	Strike-Slip
	2091_ps07309	Mountain,				
	2091_ps07309	Alaska				
74	2118_1727090	Denali,	7.9	0.01	239.5	Strike-Slip
	2118_1727360	Alaska				
75	222_B-LMO265	Livermore-	5.42	0.23	nan	Strike-Slip
	222_B-LMO355	02				
76	225_PFT045	Anza	5.19	0.12	nan	Strike-Slip
	225_PFT135	(Horse				
	225_PFT135	Canyon)-01				
77	226_TVY045	Anza	5.19	0.11	nan	Strike-Slip
	226_TVY135	(Horse				
	226_TVY135	Canyon)-01				
78	2296_ILA063-N	Chi-Chi,	5.9	0.01	80.4	Reverse
	2296_ILA063-W	Taiwan-02				
79	23_GGP010	San	5.28	0.11	nan	Reverse
	23_GGP100	Francisco				
80	2336_TAP077-N	Chi-Chi,	5.9	0.01	140.1	Reverse
	2336_TAP077-W	Taiwan-02				
81	2367_TCU045-E	Chi-Chi,	5.9	0.02	59.4	Reverse
	2367_TCU045-N	Taiwan-02				
82	2396_TCU085-E	Chi-Chi,	5.9	0.01	78.4	Reverse
	2396_TCU085-N	Taiwan-02				

No.	Names	Event	M_w	PGA	R	Mechanism
83	2423_TCU129-E 2423_TCU129-N	Chi-Chi, Taiwan-02	5.9	0.12	28.3	Reverse
84	2427_TCU138-N 2427_TCU138-W	Chi-Chi, Taiwan-02	5.9	0.04	37.3	Reverse
85	2439_TTN025-E 2439_TTN025-N	Chi-Chi, Taiwan-02	5.9	0.01	106.0	Reverse
86	2445_TTN040-N 2445_TTN040-W	Chi-Chi, Taiwan-02	5.9	0.01	80.9	Reverse
87	2447_TTN042-N 2447_TTN042-W	Chi-Chi, Taiwan-02	5.9	0.01	98.8	Reverse
88	2601_TCU045-E 2601_TCU045-N	Chi-Chi, Taiwan-03	6.2	0.01	77.4	Reverse
89	2633_TCU085-E 2633_TCU085-N	Chi-Chi, Taiwan-03	6.2	0.0	103.6	Reverse
90	2640_TCU102-E 2640_TCU102-N	Chi-Chi, Taiwan-03	6.2	0.02	45.4	Reverse
91	265_CPE045 265_CPE315	Victoria, Mexico	6.33	0.57	14.4	Strike-Slip
92	2658_TCU129-E 2658_TCU129-N	Chi-Chi, Taiwan-03	6.2	0.61	12.8	Reverse
93	2661_TCU138-W 2661_TCU138-N	Chi-Chi, Taiwan-03	6.2	0.13	22.1	Reverse
94	2677_TTN025-E 2677_TTN025-N	Chi-Chi, Taiwan-03	6.2	0.01	97.4	Reverse
95	2685_TTN040-W 2685_TTN040-N	Chi-Chi, Taiwan-03	6.2	0.01	75.1	Reverse
96	2687_TTN042-N 2687_TTN042-W	Chi-Chi, Taiwan-03	6.2	0.01	93.6	Reverse
97	2805_KAU003-N 2805_KAU003-W	Chi-Chi, Taiwan-04	6.2	0.01	116.2	Strike-Slip
98	283_A-ARI000 283_A-ARI270	Irpinia, Italy-01	6.9	0.04	52.9	Normal
99	284_A-AUL000 284_A-AUL270	Irpinia, Italy-01	6.9	0.06	9.6	Normal
100	285_A-BAG000 285_A-BAG270	Irpinia, Italy-01	6.9	0.16	8.2	Normal

No.	Names	Event	M_w	PGA	R	Mechanism
101	286_A-BIS000	Irpinia,	6.9	0.09	21.3	Normal
	286_A-BIS270	Italy-01				
102	2877_TCU102-E	Chi-Chi,	6.2	0.01	64.8	Strike-Slip
	2877_TCU102-N	Taiwan-04				
103	2897_TCU138-N	Chi-Chi,	6.2	0.04	33.6	Strike-Slip
	2897_TCU138-W	Taiwan-04				
104	2919_TTN025-E	Chi-Chi,	6.2	0.02	69.3	Strike-Slip
	2919_TTN025-N	Taiwan-04				
105	292_A-STU000	Irpinia,	6.9	0.29	10.8	Normal
	292_A-STU270	Italy-01				
106	2927_TTN040-N	Chi-Chi,	6.2	0.02	50.8	Strike-Slip
	2927_TTN040-W	Taiwan-04				
107	2929_TTN042-W	Chi-Chi,	6.2	0.03	69.0	Strike-Slip
	2929_TTN042-N	Taiwan-04				
108	293_A-TDG000	Irpinia,	6.9	0.05	59.6	Normal
	293_A-TDG270	Italy-01				
109	295_B-AUL000	Irpinia,	6.2	0.02	29.9	Normal
	295_B-AUL270	Italy-02				
110	296_B-BAG000	Irpinia,	6.2	0.05	19.6	Normal
	296_B-BAG270	Italy-02				
111	297_B-BIS000	Irpinia,	6.2	0.07	14.7	Normal
	297_B-BIS270	Italy-02				
112	2996_HWA003-N	Chi-Chi,	6.2	0.03	50.4	Reverse
	2996_HWA003-W	Taiwan-05				
113	303_B-STU000	Irpinia,	6.2	0.08	20.4	Normal
	303_B-STU270	Italy-02				
114	3139_TAP077-N	Chi-Chi,	6.2	0.01	152.1	Reverse
	3139_TAP077-W	Taiwan-05				
115	3172_TCU045-E	Chi-Chi,	6.2	0.05	73.5	Reverse
	3172_TCU045-N	Taiwan-05				
116	3194_TCU085-E	Chi-Chi,	6.2	0.01	91.8	Reverse
	3194_TCU085-N	Taiwan-05				
117	3202_TCU102-E	Chi-Chi,	6.2	0.05	52.8	Reverse
	3202_TCU102-N	Taiwan-05				
118	3217_TCU129-E	Chi-Chi,	6.2	0.39	38.9	Reverse
	3217_TCU129-N	Taiwan-05				

No.	Names	Event	M_w	PGA	R	Mechanism
119	3220_TCU138-N 3220_TCU138-W	Chi-Chi, Taiwan-05	6.2	0.18	47.5	Reverse
120	3241_TTN025-E 3241_TTN025-N	Chi-Chi, Taiwan-05	6.2	0.04	94.1	Reverse
121	3249_TTN040-N 3249_TTN040-W	Chi-Chi, Taiwan-05	6.2	0.04	67.4	Reverse
122	3251_TTN042-W 3251_TTN042-N	Chi-Chi, Taiwan-05	6.2	0.05	85.2	Reverse
123	3325_HWA003-N 3325_HWA003-W	Chi-Chi, Taiwan-06	6.3	0.03	56.0	Reverse
124	3390_ILA063-N 3390_ILA063-W	Chi-Chi, Taiwan-06	6.3	0.01	84.5	Reverse
125	3479_TCU085-E 3479_TCU085-N	Chi-Chi, Taiwan-06	6.3	0.01	83.4	Reverse
126	3489_TCU102-E 3489_TCU102-N	Chi-Chi, Taiwan-06	6.3	0.05	35.5	Reverse
127	3507_TCU129-E 3507_TCU129-N	Chi-Chi, Taiwan-06	6.3	0.26	24.8	Reverse
128	3509_TCU138-N 3509_TCU138-W	Chi-Chi, Taiwan-06	6.3	0.06	33.6	Reverse
129	3532_TTN025-E 3532_TTN025-N	Chi-Chi, Taiwan-06	6.3	0.02	94.0	Reverse
130	3540_TTN040-N 3540_TTN040-W	Chi-Chi, Taiwan-06	6.3	0.02	68.8	Reverse
131	3542_TTN042-N 3542_TTN042-W	Chi-Chi, Taiwan-06	6.3	0.03	86.4	Reverse
132	369_H-SCN045 369_H-SCN315	Coalinga-0 1	6.36	0.15	27.5	Reverse
133	43_CSM095 43_CSM185	Lytle Creek	5.33	0.07	nan	Reverse-Oblique
134	443_CEM000 443_CEM090 443_CEM090	Borah Peak, ID-02	5.1	0.02	nan	Normal
135	444_HAU000 444_HAU090 444_HAU090	Borah Peak, ID-02	5.1	0.03	nan	Normal

No.	Names	Event	M_w	PGA	R	Mechanism
136	45_DCF090 45_DCF180	Lytle Creek	5.33	0.13	nan	Reverse-Oblique
137	454_GIL067 454_GIL337	Morgan Hill	6.19	0.1	14.8	Strike-Slip
138	455_G01230 455_G01320	Morgan Hill	6.19	0.08	14.9	Strike-Slip
139	459_G06000 459_G06090	Morgan Hill	6.19	0.28	9.9	Strike-Slip
140	476_LOB050 476_LOB320	Morgan Hill	6.19	0.06	45.5	Strike-Slip
141	49_SAD003 49_SAD273	Lytle Creek	5.33	0.03	nan	Reverse-Oblique
142	495_S1010 495_S1280	Nahanni, Canada	6.76	1.06	9.6	Reverse
143	496_S2240 496_S2330	Nahanni, Canada	6.76	0.38	4.9	Reverse
144	497_S3270 497_S3360	Nahanni, Canada	6.76	0.15	5.3	Reverse
145	501_D-SG3205 501_D-SG3295	Hollister-04	5.45	0.06	nan	Strike-Slip
146	511_ARM270 511_ARM360	N. Palm Springs	6.06	0.12	38.4	Reverse-Oblique
147	512_ATL270 512_ATL360	N. Palm Springs	6.06	0.1	52.1	Reverse-Oblique
148	525_LMR252 525_LMR162	N. Palm Springs	6.06	0.05	66.7	Reverse-Oblique
149	528_H01000 528_H01090	N. Palm Springs	6.06	0.05	54.8	Reverse-Oblique
150	536_ARS270 536_ARS360	N. Palm Springs	6.06	0.11	39.1	Reverse-Oblique
151	537_SIL000 537_SIL090	N. Palm Springs	6.06	0.12	17.0	Reverse-Oblique
152	541_H02000 541_H02090	N. Palm Springs	6.06	0.08	49.1	Reverse-Oblique
153	585_CPE161 585_CPE251	Baja California	5.5	1.27	nan	Strike-Slip

No.	Names	Event	M_w	PGA	R	Mechanism
154	59_CSM095	San	6.61	0.02	89.7	Reverse
	59_CSM185	Fernando				
155	63_FTR056	San	6.61	0.09	30.2	Reverse
	63_FTR326	Fernando				
156	631_A-CHL030	Whittier	5.99	0.03	35.2	Reverse-Oblique
	631_A-CHL120	Narrows-01				
157	643_A-WON075	Whittier	5.99	0.04	27.6	Reverse-Oblique
	643_A-WON165	Narrows-01				
158	661_A-ANG000	Whittier	5.99	0.08	36.8	Reverse-Oblique
	661_A-ANG090	Narrows-01				
159	663_A-MTW000	Whittier	5.99	0.16	22.7	Reverse-Oblique
	663_A-MTW090	Narrows-01				
160	67_ISD014	San	6.61	0.01	131.0	Reverse
	67_ISD284	Fernando				
161	703_A-VAS000	Whittier	5.99	0.06	50.4	Reverse-Oblique
	703_A-VAS090	Narrows-01				
162	715_B-MTW000	Whittier	5.27	0.15	nan	Reverse-Oblique
	715_B-MTW090	Narrows-02				
163	72_L04111	San	6.61	0.16	25.1	Reverse
	72_L04201	Fernando				
164	73_L09021	San	6.61	0.14	22.6	Reverse
	73_L09291	Fernando				
165	763_GIL337	Loma	6.93	0.33	10.0	Reverse-Oblique
	763_GIL067	Prieta				
166	765_G01000	Loma	6.93	0.44	9.6	Reverse-Oblique
	765_G01090	Prieta				
167	769_G06000	Loma	6.93	0.16	18.3	Reverse-Oblique
	769_G06090	Prieta				
168	77_PUL164	San	6.61	1.16	1.8	Reverse
	77_PUL254	Fernando				
169	782_MCH000	Loma	6.93	0.07	44.3	Reverse-Oblique
	782_MCH090	Prieta				
170	788_PJH045	Loma	6.93	0.07	73.0	Reverse-Oblique
	788_PJH315	Prieta				
171	789_PTB207	Loma	6.93	0.07	83.4	Reverse-Oblique
	789_PTB297	Prieta				

No.	Names	Event	M_w	PGA	R	Mechanism
172	791_SG3261	Loma	6.93	0.07	34.3	Reverse-Oblique
	791_SG3351	Prieta				
173	795_PHT270	Loma	6.93	0.05	76.1	Reverse-Oblique
	795_PHT360	Prieta				
174	797_RIN000	Loma	6.93	0.09	74.1	Reverse-Oblique
	797_RIN090	Prieta				
175	798_TLH000	Loma	6.93	0.06	76.5	Reverse-Oblique
	798_TLH090	Prieta				
176	801_SJTE225	Loma	6.93	0.28	14.7	Reverse-Oblique
	801_SJTE315	Prieta				
177	804_SSF115	Loma	6.93	0.08	63.2	Reverse-Oblique
	804_SSF205	Prieta				
178	809_UC2000	Loma	6.93	0.34	18.5	Reverse-Oblique
	809_UC2090	Prieta				
179	810_LOB000	Loma	6.93	0.46	18.4	Reverse-Oblique
	810_LOB090	Prieta				
180	813_YBI000	Loma	6.93	0.06	75.2	Reverse-Oblique
	813_YBI090	Prieta				
181	828_PET000	Cape	7.01	0.62	8.2	Reverse
	828_PET090	Mendocino				
182	87_SAD003	San	6.61	0.17	30.7	Reverse
	87_SAD273	Fernando				
183	879_LCN260	Landers	7.28	0.72	2.2	Strike-Slip
	879_LCN345					
184	89_TEH090	San	6.61	0.04	63.8	Reverse
	89_TEH180	Fernando				
185	891_SIL000	Landers	7.28	0.05	50.8	Strike-Slip
	891_SIL090					
186	897_29P000	Landers	7.28	0.07	41.4	Strike-Slip
	897_29P090					
187	922_PPC090	Big	6.46	0.03	nan	Strike-Slip
	922_PPC180	Bear-01				
188	925_RCD090	Big	6.46	0.04	nan	Strike-Slip
	925_RCD180	Bear-01				
189	934_SVP090	Big	6.46	0.06	nan	Strike-Slip
	934_SVP360	Bear-01				

No.	Names	Event	M_w	PGA	R	Mechanism
190	938_WBR090	Big	6.46	0.08	nan	Strike-Slip
	938_WBR360	Bear-01				
191	943_ACI000	Northridge	6.69	0.05	68.9	Reverse
	943_ACI270	-01				
192	946_ATB000	Northridge	6.69	0.06	46.9	Reverse
	946_ATB090	-01				
193	957_HOW060	Northridge	6.69	0.14	16.9	Reverse
	957_HOW330	-01				
194	989_CHL070	Northridge	6.69	0.21	20.5	Reverse
	989_CHL160	-01				
195	994_0141-270	Northridge	6.69	0.25	23.8	Reverse
	994_0141-360	-01				

5.11.2 McGuire et al. (2001) Database (CEUS)

The distance given in the tables, R, is the closest distance to the rupture area (an *nan* indicates that the value is not known). The units of R and PGA are km and g, respectively.

Table 5.6: Central-Eastern United States (CEUS) Earthquake Motions

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
1	1125A54E	Saguenay	5.9	0.01	91.4	M55D050.100
	1125A54N			0.01		
2	1125A61E	Saguenay	5.9	0.01	91.9	M55D050.100
	1125A61N			0.01		
3	1125A64E	Saguenay	5.9	0.01	99.1	M55D050.100
	1125A64N			0.01		
4	1125S08L	Saguenay	5.9	0.13	97.5	M55D050.100
	1125S08T			0.06		
5	1125S16L	Saguenay	5.9	0.11	51.9	M55D050.100
	1125S16T			0.13		
6	1125S17L	Saguenay	5.9	0.17	70.3	M55D050.100
	1125S17T			0.1		

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
7	1125S20L	Saguenay	5.9	0.15	95.0	M55D050.100
	1125S20T			0.1		
8	29P000	Landers	7.3	0.21	42.2	M75D010.050
	29P090			0.18		
9	A-CSH000	Whittier Narrows	6.0	0.06	70.9	M65D050.100
	A-CSH090			0.07		
10	A-GRN180	Whittier Narrows	6.0	0.73	9.0	M65D000.010
	A-GRN270			0.53		
11	A-GRV060	Whittier Narrows	6.0	0.56	12.1	M55D000.050
	A-GRV330			0.8		
12	A-HMC180	Helena, Montana	6.2	0.4	8.0	M65D000.010
	A-HMC270			0.44		
13	A-LVL000	Mammoth Lakes	6.0	0.23	19.7	M65D010.050
	A-LVL090			0.13		
14	A-MAL180	Whittier Narrows	6.0	0.1	65.3	M65D050.100
	A-MAL270			0.09		
15	A-ORR000	Whittier Narrows	6.0	0.13	78.3	M65D050.100
	A-ORR090			0.11		
16	A-RIV180	Whittier Narrows	6.0	0.15	56.8	M65D050.100
	A-RIV270			0.13		
17	A-SON033	Borrego Mtn	6.8	0.11	124.7	M65D100.200
	A-SON303			0.1		
18	A-VIR200	Whittier Narrows	6.0	0.12	53.3	M65D050.100
	A-VIR290			0.08		
19	A54–N	Saguenay	4.5	0.0	99.4	M55D050.100
	A54–E			0.0		
20	ABY000	Landers	7.3	0.37	69.2	M75D050.100
	ABY090			0.43		
21	AMB–X	Georgia, USSR	6.2	0.04	73.7	M65D050.100
	AMB–Y			0.04		
22	ARC000	Kocaeli, Turkey	7.4	0.3	17.0	M75D010.050
	ARC090			0.21		
23	ARS270	N. Palm Springs	6.0	0.29	43.8	M65D010.050
	ARS360			0.3		
24	ATL270	N. Palm Springs	6.0	0.21	55.4	M55D050.100
	ATL360			0.16		

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
25	B-A3E146	Livermore	5.4	0.08	31.0	M55D000.050
	B-A3E236			0.04		
26	B-KOD180	Livermore	5.4	0.34	17.6	M55D000.050
	B-KOD270			0.18		
27	BOL000	Duzce, Turkey	7.1	1.64	16.0	M75D010.050
	BOL090			1.76		
28	BRN000	Loma Prieta	6.9	1.3	10.3	M75D000.010
	BRN090			1.24		
29	C-ATC270	Coalinga	5.2	0.66	11.0	M55D000.050
	C-ATC360			0.72		
30	C-ATP270	Coalinga	5.2	0.94	11.0	M55D000.050
	C-ATP360			0.56		
31	C-OLC270	Coalinga	5.2	0.7	10.0	M55D000.050
	C-OLC360			0.65		
32	C-TSM270	Coalinga	5.2	0.37	10.4	M55D000.050
	C-TSM360			0.34		
33	C08050	Parkfield	6.1	0.59	9.2	M65D000.010
	C08320			0.54		
34	CAD250	Santa Barbara	6.0	0.12	36.6	M55D000.050
	CAD340			0.07		
35	CHY080-N	Chi-Chi, Taiwan	7.6	2.11	6.9	M75D000.010
	CHY080-W			2.75		
36	CLS000	Loma Prieta	6.9	1.33	5.1	M65D000.010
	CLS090			1.08		
37	CLS000	Loma Prieta	6.9	1.3	5.1	M75D000.010
	CLS090			1.21		
38	CLS220	Morgan Hill	6.2	0.16	22.7	M65D010.050
	CLS310			0.21		
39	CPE045	Victoria, Mexico	6.1	2.07	34.8	M65D010.050
	CPE315			1.62		
40	CPM000	Cape Mendocino	7.1	4.72	8.5	M75D000.010
	CPM090			3.24		
41	CSM095	San Fernando	6.6	0.05	86.6	M65D050.100
	CSM185			0.04		
42	CSM095	San Fernando	6.6	0.06	86.6	M65D100.200
	CSM185			0.04		

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
43	CUC090	Northridge	6.7	0.15	80.0	M65D050.100
	CUC180			0.1		
44	CUC090	Northridge	6.7	0.18	80.0	M65D100.200
	CUC180			0.12		
45	CYC195	Morgan Hill	6.2	1.61	0.1	M65D000.010
	CYC285			3.37		
46	D-OLC270	Coalinga	5.8	1.59	8.2	M55D000.050
	D-OLC360			0.72		
47	D-PLM270	Coalinga	5.8	0.48	12.2	M55D000.050
	D-PLM360			0.51		
48	DAY-LN	Tabas, Iran	7.4	0.99	17.0	M75D010.050
	DAY-TR			0.95		
49	DCF090	Lytle Creek	5.4	0.29	21.9	M55D000.050
	DCF180			0.29		
50	FER-L1	Tabas, Iran	7.4	0.28	94.4	M75D050.100
	FER-T1			0.36		
51	FSD172	San Fernando	6.6	0.34	27.5	M65D010.050
	FSD262			0.37		
52	G06000	Morgan Hill	6.2	0.47	11.8	M65D000.010
	G06090			0.61		
53	G06000	Morgan Hill	6.2	0.46	11.8	M65D010.050
	G06090			0.61		
54	GAZ000	Gazli, USSR	6.8	1.76	3.0	M65D000.010
	GAZ090			2.35		
55	GAZ000	Gazli, USSR	6.8	1.77	3.0	M75D000.010
	GAZ090			1.79		
56	GBZ000	Kocaeli, Turkey	7.4	0.45	17.0	M75D010.050
	GBZ270			0.34		
57	GIL067	Morgan Hill	6.2	0.28	16.2	M65D010.050
	GIL337			0.2		
58	GRN180	Landers	7.3	0.14	141.6	M75D100.200
	GRN270			0.12		
59	GYN000	Kocaeli, Turkey	7.4	0.31	35.5	M75D010.050
	GYN090			0.28		
60	H-PG3000	Coalinga	6.4	0.25	38.8	M65D010.050
	H-PG3090			0.22		

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
61	H-SUP045	Imperial Valley	6.5	0.23	26.0	M65D010.050
	H-SUP135			0.53		
62	H-TM2000	Coalinga	6.4	0.05	40.5	M65D010.050
	H-TM2090			0.08		
63	H-VC4000	Coalinga	6.4	0.11	34.6	M65D010.050
	H-VC4090			0.09		
64	H-XCV075	Mammoth Lakes	5.0	0.06	9.1	M55D000.050
	H-XCV165			0.11		
65	H-Z11000	Coalinga	6.4	0.19	28.4	M65D010.050
	H-Z11090			0.17		
66	H01000	N. Palm Springs	6.0	0.11	63.3	M55D050.100
	H01090			0.1		
67	HCP045	N. Palm Springs	6.0	0.51	34.9	M55D000.050
	HCP135			0.29		
68	HWA023-N	Chi-Chi, Taiwan	7.6	0.07	57.0	M75D050.100
	HWA023-W			0.07		
69	HWA026-N	Chi-Chi, Taiwan	7.6	0.14	58.8	M75D050.100
	HWA026-W			0.2		
70	HWA056-N	Chi-Chi, Taiwan	7.6	0.2	48.7	M75D010.050
	HWA056-W			0.21		
71	ILA031-N	Chi-Chi, Taiwan	7.6	0.15	94.7	M75D050.100
	ILA031-W			0.12		
72	ILA051-N	Chi-Chi, Taiwan	7.6	0.06	90.3	M75D050.100
	ILA051-W			0.15		
73	ILA063-N	Chi-Chi, Taiwan	7.6	0.22	71.6	M75D050.100
	ILA063-W			0.23		
74	ISD014	San Fernando	6.6	0.02	113.0	M65D100.200
	ISD284			0.02		
75	IZN180	Kocaeli, Turkey	7.4	0.24	29.7	M75D010.050
	IZN090			0.17		
76	IZT180	Kocaeli, Turkey	7.4	0.4	7.7	M75D000.010
	IZT090			0.35		
77	KAU078-N	Chi-Chi, Taiwan	7.6	0.07	102.8	M75D100.200
	KAU078-W			0.11		
78	KBU000	Kobe	6.9	0.6	0.2	M75D000.010
	KBU090			0.71		

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
79	KJM000	Northridge	6.7	2.12	0.6	M65D000.010
	KJM090			1.76		
80	KJM000	Kobe	6.9	1.92	0.6	M75D000.010
	KJM090			1.56		
81	KSH-L1	Tabas, Iran	7.4	0.14	199.1	M75D100.200
	KSH-T1			0.14		
82	L-BPL070	Mammoth Lakes	6.0	0.19	43.7	M65D010.050
	L-BPL160			0.26		
83	L04111	San Fernando	6.6	0.49	24.2	M65D010.050
	L04201			0.42		
84	LCN260	Landers	7.3	1.84	1.1	M75D000.010
	LCN345			2.25		
85	LGP000	Loma Prieta	6.9	1.65	6.1	M65D000.010
	LGP090			1.52		
86	LGP000	Loma Prieta	6.9	1.64	6.1	M75D000.010
	LGP090			1.64		
87	LMR162	N. Palm Springs	6.0	0.13	73.7	M55D050.100
	LMR252			0.12		
88	MA1130	San Fernando	6.6	0.02	115.0	M65D100.200
	MA1220			0.03		
89	MA2130	San Fernando	6.6	0.03	113.0	M65D100.200
	MA2220			0.02		
90	MA3130	San Fernando	6.6	0.02	113.0	M65D100.200
	MA3220			0.02		
91	MCD000	Kocaeli, Turkey	7.4	0.13	62.3	M75D050.100
	MCD090			0.13		
92	MDR000	Duzce, Turkey	7.1	0.2	34.6	M75D010.050
	MDR090			0.12		
93	MEL090	Northridge	6.7	0.15	51.6	M65D050.100
	MEL180			0.07		
94	MEL090	Northridge	6.7	0.19	51.6	M65D100.200
	MEL180			0.08		
95	MEL090	Landers	7.3	0.09	126.4	M75D100.200
	MEL180			0.06		
96	NSK-E	Chi-Chi, Taiwan	7.6	0.13	64.5	M75D050.100
	NSK-N			0.12		

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
97	PAC175	Northridge	6.7	1.38	8.0	M65D000.010
	PAC265			1.26		
98	PKC090	Northridge	6.7	0.75	8.2	M65D000.010
	PKC360			0.98		
99	PLC000	Landers	7.3	0.16	95.9	M75D050.100
	PLC090			0.16		
100	PLC258	N. Palm Springs	6.0	0.14	71.9	M55D050.100
	PLC348			0.12		
101	PNG-E	Chi-Chi, Taiwan	7.6	0.11	114.2	M75D100.200
	PNG-N			0.11		
102	PUL104	Northridge	6.7	3.94	8.0	M65D000.010
	PUL194			3.59		
103	RAN000	Northridge	6.7	0.2	55.2	M65D050.100
	RAN090			0.16		
104	RAN000	Northridge	6.7	0.24	55.2	M65D100.200
	RAN090			0.19		
105	RIV180	N. Palm Springs	6.0	0.12	71.1	M55D050.100
	RIV270			0.1		
106	RIV180	Northridge	6.7	0.18	101.3	M65D100.200
	RIV270			0.26		
107	RIV180	Landers	7.3	0.13	96.1	M75D050.100
	RIV270			0.15		
108	S1010	Nahanni	6.8	0.98	6.0	M65D000.010
	S1280			1.1		
109	S3270	Nahanni	6.8	0.15	16.0	M65D010.050
	S3360			0.14		
110	SER000	Northridge	6.7	0.1	79.5	M65D050.100
	SER270			0.1		
111	SER000	Northridge	6.7	0.12	79.5	M65D100.200
	SER270			0.12		
112	SER000	Landers	7.3	0.09	131.4	M75D100.200
	SER270			0.12		
113	SHL000	Cape Mendocino	7.1	0.65	33.8	M75D010.050
	SHL090			0.58		
114	SIL000	Landers	7.3	0.15	51.7	M75D050.100
	SIL090			0.14		

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
115	SKR180	Duzce, Turkey	7.1	0.05	42.7	M75D010.050
	SKR090			0.04		
116	SLO234	Southern Calif	6.0	0.07	70.0	M55D050.100
	SLO324			0.11		
117	SLO234	Parkfield	6.1	0.03	60.0	M65D050.100
	SLO324			0.03		
118	SOD015	San Fernando	6.6	0.16	58.1	M65D050.100
	SOD285			0.19		
119	SOD015	San Fernando	6.6	0.2	58.1	M65D100.200
	SOD285			0.23		
120	SON033	San Fernando	6.6	0.04	122.0	M65D100.200
	SON303			0.04		
121	SOR225	Northridge	6.7	0.15	54.1	M65D050.100
	SOR315			0.17		
122	SOR225	Northridge	6.7	0.19	54.1	M65D100.200
	SOR315			0.21		
123	SRO-NS	Fruili, Italy	5.5	0.04	17.9	M55D000.050
	SRO-WE			0.09		
124	SUL230	Landers	7.3	0.09	162.6	M75D100.200
	SUL320			0.08		
125	TAP035-N	Chi-Chi, Taiwan	7.6	0.22	96.8	M75D050.100
	TAP035-W			0.12		
126	TAP036-N	Chi-Chi, Taiwan	7.6	0.07	95.6	M75D050.100
	TAP036-W			0.05		
127	TAP059-N	Chi-Chi, Taiwan	7.6	0.09	125.9	M75D100.200
	TAP059-W			0.07		
128	TAP060-N	Chi-Chi, Taiwan	7.6	0.09	128.4	M75D100.200
	TAP060-W			0.08		
129	TAP067-N	Chi-Chi, Taiwan	7.6	0.09	104.2	M75D100.200
	TAP067-W			0.09		
130	TAP069-N	Chi-Chi, Taiwan	7.6	0.07	135.3	M75D100.200
	TAP069-W			0.06		
131	TAP072-N	Chi-Chi, Taiwan	7.6	0.16	110.0	M75D100.200
	TAP072-W			0.07		
132	TAP075-N	Chi-Chi, Taiwan	7.6	0.17	118.4	M75D100.200
	TAP075-W			0.2		

No.	Names	Event	M_w	PGA	R	$M_w - R$ Bin
133	TAP078-N	Chi-Chi, Taiwan	7.6	0.09	131.0	M75D100.200
	TAP078-W			0.09		
134	TCU015-N	Chi-Chi, Taiwan	7.6	0.22	47.3	M75D010.050
	TCU015-W			0.19		
135	TCU025-N	Chi-Chi, Taiwan	7.6	0.12	54.3	M75D050.100
	TCU025-W			0.14		
136	TCU046-N	Chi-Chi, Taiwan	7.6	0.22	14.3	M75D010.050
	TCU046-W			0.34		
137	TCU047-N	Chi-Chi, Taiwan	7.6	1.17	33.0	M75D010.050
	TCU047-W			0.7		
138	TCU087-N	Chi-Chi, Taiwan	7.6	0.21	3.1	M75D000.010
	TCU087-W			0.25		
139	TCU089-N	Chi-Chi, Taiwan	7.6	0.44	8.2	M75D000.010
	TCU089-W			0.66		
140	TCU095-N	Chi-Chi, Taiwan	7.6	1.7	43.4	M75D010.050
	TCU095-W			0.78		
141	TCU120-N	Chi-Chi, Taiwan	7.6	0.38	8.1	M75D000.010
	TCU120-W			0.48		
142	TCU128-N	Chi-Chi, Taiwan	7.6	0.31	9.7	M75D000.010
	TCU128-W			0.27		
143	TCU136-E	Chi-Chi, Taiwan	7.6	0.28	8.9	M75D000.010
	TCU136-N			0.3		
144	TFS000	N. Palm Springs	6.0	0.23	73.2	M55D050.100
	TFS090			0.16		
145	TMB205	Parkfield	6.1	0.99	9.9	M65D000.010
	TMB295			0.69		
146	VIR200	Landers	7.3	0.06	194.1	M75D100.200
	VIR290			0.04		
147	WTW115	Lytle Creek	5.4	0.28	15.4	M55D000.050
	WTW205			0.32		
148	WWJ090	Northridge	6.7	0.11	68.4	M65D050.100
	WWJ180			0.08		
149	WWJ090	Northridge	6.7	0.16	68.4	M65D100.200
	WWJ180			0.12		

Chapter 6

Manuscript #4: An SPT Liquefaction Evaluation Procedure Based on Dissipated Energy

The authors of the following manuscript intend to submit it to ASCE's *Journal of Geotechnical and Geoenvironmental Engineering*.

Samuel Lasley made the following contributions to this manuscript:

- wrote the first draft of the manuscript under the general direction of the other two authors.
- performed all regression and analysis
- investigate various functional forms for the limit state curve
- performed comparisons with the stress-based procedures
- created all tables and figures

Dr. Green made the following contributions to this manuscript:

- provided the framework for the proposed energy-based liquefaction evaluation procedure and associated equivalent number of cycles correlation (see Green 2000)

- suggested the inclusion of the curve-fit to the Tokimatsu and Seed procedure
- suggested the case history databases
- suggested looking at comparisons between the proposed procedure and the stress-based procedures for extreme conditions

Dr. Rodriguez-Marek made the following contributions:

- Directed and instructed in the application of input parameter uncertainties to the limit state curve-fitting, including the first-order approximations
- Helped troubleshoot code and procedures
- aided in crafting the likelihood function for both including and excluding uncertainties
- Suggested the use of the Nash-Sutcliffe model efficiency coefficient

Dr. Green and Dr. Rodriguez-Marek both:

- provided feedback during all stages of the research
- edited the manuscript and provided valuable feedback between revisions

Dr. James K. Mitchell provided the motivation to develop an energy-based liquefaction evaluation procedure.

An SPT Liquefaction Evaluation Procedure Based on Dissipated Energy

Samuel J. Lasley[1], Russell A. Green[2], Adrian Rodriguez-Marek[3], James K. Mitchell[4]

[1] Doctoral Candidate, Department of Civil and Environmental Engineering, Virginia Tech, Blacksburg, Virginia, U.S.A.; slasley@vt.edu

[2] Professor, Department of Civil and Environmental Engineering, Virginia Tech, Blacksburg, Virginia, U.S.A.; rugreen@vt.edu

[3] Professor, Department of Civil and Environmental Engineering, Virginia Tech, Blacksburg, Virginia, U.S.A.; adrianrm@vt.edu

[4] University Distinguished Professor Emeritus, Department of Civil and Environmental Engineering, Virginia Tech, Blacksburg, Virginia, U.S.A.; jkm@vt.edu

6.1 Abstract

The stress-based simplified liquefaction evaluation procedure was proposed in 1971 and continues to be an important tool for geotechnical earthquake engineering. However, at its core it remains a semi-empirical procedure with uncertain application outside the case history database from which it was regressed. Proposed herein is a simplified liquefaction evaluation procedure based on dissipated energy and presented in a form that will be familiar to geotechnical earthquake engineering practitioners. The proposed form is a function of peak ground acceleration, earthquake magnitude, SPT blow-count, and fines content, and it returns probability of liquefaction and/or factor of safety against liquefaction. The predictive accuracy of the proposed procedure rivals that of other popular stress-based procedures and is compatible with the results of equivalent-linear site response analyses. By using dissipated energy to quantify demand instead of the cyclic stress ratio, the capacity curves of the proposed procedure are insensitive to the shape of the demand earthquake time history. Thus, the proposed procedure can utilize the response of any arbitrary loading time history, including those of non-earthquake origin. The result is a robust liquefaction evaluation procedure that is simple to implement and with a wider sphere of application.

6.2 Introduction

Proposed herein is a liquefaction evaluation procedure based on dissipated energy. Soil liquefaction continues to be a source of significant hazard in seismically-active regions of the world. The current state-of-the-practice in liquefaction evaluation is the stress-based simplified procedure. Most stress-based simplified procedures predict the probability of liquefaction (and/or factor of safety against liquefaction) from the peak ground acceleration, total and initial effective overburden stresses, earthquake magnitude, and some in-situ measure of soil state (e.g. SPT blow count, CPT tip resistance, shear wave velocity). These procedures are semi-empirical (based on liquefaction/no-liquefaction case histories) and their validity beyond the limits of the case history databases from which they were derived is uncertain. For example, few liquefaction case histories exist for the central-eastern United States and a stress-based simplified procedure derived from western United States liquefaction case histories may not be directly applicable.

Dissipated energy has been shown to be a good indicator of the soil breakdown that leads to liquefaction, and the dissipated energy required to initiate liquefaction is relatively insensitive to the shape of the loading function (Figueroa et al. 1994; Liang et al. 1995). The energy-based procedure proposed herein accepts the same inputs as the stress-based simplified procedures and predicts the occurrence of liquefaction for the same case history database with the same success rate. Additionally, the form of the proposed energy-based procedure allows for application to other tectonic regimes via a regime-specific number of equivalent cycles (n_{eq}) correlation which accounts for differences in ground motion shape and duration between tectonic regimes. The procedure proposed herein allows the results of equivalent-linear site response analyses to be directly applied, and thus can be used to evaluate liquefaction potential for any arbitrary loading, be it of earthquake origin or otherwise.

The approach adopted in this research is to use a liquefaction/no-liquefaction case history database to compute a capacity curve. The functional form for this curve is first explored via a program of laboratory testing. In this paper, brief summaries of the stress-based simplified and energy-based liquefaction evaluation procedures are given. The results of laboratory data are presented showing the relationship between normalized dissipated energy ($\Delta W/\sigma'_{v0}$) and relative density. Using the same general functional form from the labo-

ratory results, an empirical limit state (capacity) curve is regressed from an SPT database of liquefaction/no-liquefaction case histories. This proposed capacity curve is compared with the laboratory curve via a correlation between $N_{1,60cs}$ and relative density. The empirical field-based curve is recommended for practice and, as such, is also compared with two commonly-used stress-based liquefaction evaluation procedures. Finally, implications of the proposed procedure with regards to the effects of initial effective overburden stress are discussed.

6.3 Existing Stress-based Methods

The stress-based "simplified" liquefaction evaluation procedure was first proposed by Seed and Idriss (1971) and, separately, Whitman (1971). The demand term of the procedure, CSR, is determined using the peak ground acceleration of the expected earthquake:

$$CSR = \frac{\tau_{avg}}{\sigma'_{v0}} = 0.65 \cdot \frac{a_{max}}{g} \cdot \frac{\sigma_v}{\sigma'_{v0}} \cdot r_d \tag{6.1}$$

where τ_{avg} is the average seismically-induced shear stress, a_{max} is the peak ground acceleration, g is the acceleration due to gravity, σ_v is the total overburden stress, σ'_{v0} is the initial effective overburden stress, and r_d is the stress reduction coefficient (assumed by Whitman to be 1).

Over the years, this simplified procedure has moved from being laboratory-based to field-based (e.g. SPT, CPT) and undergone various improvements. These improvements can be summarized in the following equation for the factor of safety (Youd et al. 2001):

$$FS = \frac{CRR}{CSR} \cdot MSF \cdot K_\sigma \cdot K_\alpha \tag{6.2}$$

where CRR is the cyclic resistance ratio for a $M7.5$ earthquake, MSF is the magnitude scaling factor, K_σ is the overburden correction factor, and K_α is a correction factor for non-level ground conditions.

Cetin et al. (2004) proposed a probabilistic SPT procedure which allowed uncertainties in input parameters and the location of the CRR capacity curve

to be quantified. Cetin et al. obtained their capacity curve by means of the maximization of a likelihood function. The same likelihood function is used in the research presented in this paper. Idriss and Boulanger (2008) presented deterministic, stress-based simplified liquefaction evaluation procedures for the SPT and CPT. Their procedures retain the same general form of Youd et al. (2001) but provide updated relationships for r_d, MSF, and K_σ. Boulanger and Idriss (2012) presented a probabilistic update to the 2008 deterministic SPT procedure.

6.4 Dissipated Energy and Liquefaction

Cyclic loading of soil causes a dissipation of energy. The dissipated energy per unit volume of the soil is the cumulative area bound by stress-strain hysteresis loops. Nemat-Nasser and Shokooh (1977) first introduced the idea of using dissipated energy to estimate liquefaction initiation, and numerous studies since have been performed to validate the idea. The results of laboratory testing (Simcock et al. 1983; Law et al. 1990; Green et al. 2000) have produced several pore water pressure models based on dissipated energy. The laboratory results have also shown that dissipated energy is a good predictor of excess pore water pressure generation, irrespective of the shape of the loading function (Figueroa et al. 1994; Liang et al. 1995) or whether an initial static shear stress is applied (Jafarian et al. 2012).

Numerous energy-based liquefaction evaluation procedures have been proposed over the years, based on both field case histories (e.g. Davis and Berrill 1982; Triunfac 1995; Green 2001; Mayfield 2007) and laboratory testing (e.g. Liang 1995; Davis and Berrill 1996; Kokusho 2013). Forward analyses of field-based procedures rely on attenuation relationships (e.g. radiated energy, cumulative absolute velocity, peak ground acceleration), and, perhaps with the exception of peak ground acceleration, these attenuation relationships are not available for all tectonic regimes and have difficulty predicting near-field effects or site-specific conditions. Moreover, few of these relationships exist in the literature. Forward analyses using the energy-based procedures that derive their capacity curves from laboratory test results require that lab tests are performed on soil specimens sampled from the site of interest and/or non-linear effective-stress site response analyses. A few lab-based procedures use equivalent-linear site response analyses to quantify demand and then

compare this demand with a capacity curve derived from laboratory results. However, this comparison is not appropriate since the equivalent-linear response assumes single values of shear modulus and damping for the entire time history, neglecting the effect of soil softening on the total dissipated energy to liquefaction (Green 2001).

The liquefaction evaluation procedure proposed herein is based on the procedure developed by Green (2001) and overcomes the limitations of the other energy-based procedures. Although the functional form of the capacity curve is inspired by the results of laboratory testing, it is not directly based on laboratory test results, removing the need for laboratory tests in forward analyses. Instead, it is based on field case histories and, for forward analyses, relies on peak ground acceleration attenuation relationships and number of equivalent cycles correlations. Both of these have a long history in stress-based simplified liquefaction procedures, and peak ground acceleration attenuation relationships are well-developed for various tectonic regimes. A new number-of-equivalent-cycles correlation has been developed by the authors (Lasley et al. 2015a) for both active shallow-crustal (e.g. western United States) and stable continental (e.g. central-eastern United States) tectonic regimes. This n_{eq} correlation is based on dissipated energy and low-cycle fatigue. Finally, the procedure proposed herein has been developed in a format intended to be familiar to users of traditional stress-based procedures.

6.5 Laboratory Approach

Stress-controlled constant-volume (Finn et al. 1979) cyclic simple shear tests were performed by the authors on Monterey 0/30 sand. These tests were performed on specimens prepared via dry pluviation (Vaid and Negussey 1988) with relative densities ranging from approximately 30 to 80%, and cyclically loaded while overburden stresses of 60, 100, and 250 kPa were imposed on the specimens. Liquefaction was deemed to have occurred when a shear strain of 3.75% single-amplitude was exceeded (Vaid and Sivathayalan 1996; Sivathayalan and Ha 2011).

Figure 6.1 shows the stress-strain hysteresis loops and the shear stress versus normal stress plot for one of the tests. For each of the tests, the dissipated energy to liquefaction was computed. Figure 6.2 shows the accumulation of dissipated energy as a function of the cycle ratio ($N/N_{\text{liq.}}$). In addition to

the dissipated energy to liquefaction ($\Delta W_{\text{Effect.}}$), another value of dissipated energy, ΔW_{Total}, was obtained for each test by drawing a best fit line through the accumulated dissipated energy of the first half of the test. The value of this line at a cycle ratio of 1 is the dissipated energy corresponding to total stress analyses. This value of dissipated energy removes the dissipated energies due to the softening of the soil profile with increasing pore water pressure (Green 2000). It is especially useful when comparing lab results with those from equivalent-linear site response analyses because the equivalent-linear site response does not allow for soil softening as the loading progresses (i.e. G_γ and D_γ are constant for the entire loading time history of a given layer).

Figure 6.3 shows the cyclic stress ratio versus cycles to liquefaction for all of the tests with contour lines of equal normalized dissipated energy (corresponding to total stress analysis) to liquefaction. Dissipated energies were normalized by dividing the value by the initial effective normal stress, leaving a unit-less quantity. As can be seen in the figure, as relative density increases, the normalized dissipated energy to liquefaction also increases. Figure 6.4 illustrates the relationship between the normalized dissipated energy to liquefaction (ΔW_{Total}) and the relative density of the test specimen. A strong relationship between the logarithm of normalized dissipated energy and the square of relative density is readily apparent. This functional form inspired the subsequent limit state functional form between normalized dissipated energy and SPT blow counts.

6.6 Case History-Based Approach

Similar to recent stress-based liquefaction evaluation procedures, the proposed energy-based procedure is based on field data in the form of a liquefaction/no-liquefaction case history database. The case history database of Boulanger et al. (2012) was adopted for use in this study. This SPT database contains the details of 115 liquefaction, 112 no-liquefaction, and 3 marginal liquefaction cases. Boulanger et al. gathered case histories from the Seed et al. (1984) and Cetin et al. (2004) databases, but then reevaluated and corrected the magnitude (M_w), a_{max}, and SPT blow counts, as necessary. From this database M_w, a_{max}, overburden- and fines content-corrected SPT blow count ($N_{1,60cs}$), total overburden stress (σ_v),

and initial effective overburden stress (σ'_{v0}) were used to calibrate the limit state curve proposed herein. In addition to the data from the Boulanger et al. (2012) database, shear wave velocities corresponding to 46 case histories (25 liquefaction; 21 no liquefaction) were taken from the Kayen et al. (2013) shear wave velocity database for sites matched between the two databases.

6.6.1 Methodology

As mentioned, dissipated energy is the cumulative area bound by stress-strain hysteresis loops, and can be estimated using the trapezoidal rule (Green and Terri 2005):

$$\Delta W = \frac{1}{2}\sum_{i=1}^{j-1}\left((\tau_{i+1}+\tau_i)\cdot(\gamma_{i+1}-\gamma_i)\right) \tag{6.3}$$

where ΔW is the dissipated energy per unit volume of soil in units of stress; τ_i and γ_i are the stress and strain, respectively, at point i; and j is the number of data points in the stress/strain time history. (The dissipated energy referred to in the remaining sections of this paper is ΔW_{Total}, the dissipated energy corresponding to total stress analyses. For convenience, it will be referred to as dissipated energy or ΔW without the additional $_{\text{Total}}$ subscript.) Because the case history database does not contain full stress and strain time histories, the dissipated energy of each case history cannot be estimated using Eqn. 6.3. Instead, the dissipated energy is estimated by multiplying the dissipated energy in one equivalent cycle ($\Delta W_{1,eq}$) by the number of equivalent cycles (n_{eq}). The dissipated energy in one cycle is derived from the equation for damping ratio (D) (Jacobsen 1960):

$$D = \frac{1}{4\pi}\frac{\Delta W_{1,eq}}{W_{stored}} \tag{6.4}$$

where $\Delta W_{1,eq}$ is the dissipated energy from one cycle and W_{stored} is the stored energy which is equal to $1/2\tau\gamma$. Substituting in the definition of stored energy, letting $\tau = \tau_{avg}$ and $\gamma = \tau_{avg}/G_\gamma$, and rearranging terms, we

obtain:

$$\Delta W_{1,eq} = \frac{2\pi D_\gamma \tau_{avg}^2}{G_\gamma} \tag{6.5}$$

As previously mentioned, the quantity τ_{avg} is the amplitude of shear strain in one equivalent cycle and is estimated in the same way as the stress-based procedure (Eqn. 6.1). A new stress reduction coefficient was developed as part of this research (Lasley et al. 2015b), and takes the following form:

$$r_d = (1 - \alpha) \exp\left(\frac{-z}{\beta}\right) + \alpha \tag{6.6}$$

where z is in meters, and

$$\alpha = \exp(a_1 + a_2 \cdot M_w) \tag{6.7}$$

$$\beta = a_3 + a_4 \cdot M_w \tag{6.8}$$

Regression coefficients a_1-a_4 are given in Table 6.1.

The values of D_γ and G_γ are obtained from shear modulus reduction and damping versus strain curves in a way similar to equivalent-linear site response analysis: G_γ/G_{max} is function of shear strain, γ, and must be chosen to be consistent with $\tau_{avg}/G_\gamma = \gamma$. Tokimatsu and Seed (1987) proposed a direct approach to obtain these values. Figures 6.5 through 6.7 provide graphical implementations tailored for the liquefaction evaluation procedure proposed herein and correspond to clean sands, sands with high fines contents, and silts, respectively. (The diamonds in these figures indicate a threshold shear strain of 0.01%; no liquefaction is expected for shear strains below the threshold.) Thus, for given values of τ_{avg}, small strain shear modulus (G_{max}), and mean effective stress, the quantity $\Delta W_{1,eq}/G_{max}$ is readily obtained. Furthermore, the curves in Figures 6.5 and 6.6 have been fit with a fifth-order polynomial to ease computations (an appropriate fit was not obtained for the silt curve in Figure 6.7). For a given mean effective stress,

the dissipated energy in one cycle divided by the small-strain shear modulus can be approximated as:

$$\log_{10}\left(\frac{\Delta W_{1,eq}}{G_{max}}\right) = d_1\Gamma^5 + d_2\Gamma^4 + d_3\Gamma^3 + d_4\Gamma^2 + d_5\Gamma + d_6 \tag{6.9}$$

where:

$$\Gamma = \log_{10}\left(0.65 \cdot a_{max} \cdot \sigma_v \cdot r_d/(g \cdot G_{max})\right);\ \sim -5 \leq \Gamma \leq -3 \tag{6.10}$$

and the regression coefficients (d_1-d_6) for a range of mean effective stresses are given in Table 6.2 for clean sands and Table 6.3 for sands with high fines content. In obtaining G_γ and D_γ values from the shear modulus reduction and damping curves, a lower limit of 40 kPa was imposed for the initial effective overburden stress. This roughly corresponds to the lowest mean initial effective stress ($\sigma'_{m0} = 0.25$ atm) curve given by Darendeli and Stokoe (2001).

The small-strain shear modulus, G_{max} is calculated from the shear wave velocity, where available, or estimated using the corrected SPT blow count (Seed et al. 1986; Ohta and Goto 1976):

$$G_{max} = 440 \cdot N_{1,60cs}^{\frac{1}{3}} \cdot P_a \cdot \sqrt{\frac{\sigma'_{m0}}{P_a}} \tag{6.11}$$

where P_a is standard atmospheric pressure, and σ'_{m0} is the mean initial effective stress ($\sigma'_{m0} = (1 + 2K_o)\sigma'_{v0}/3$) in the same units as P_a. Recent research (Wair et al. 2012) has suggested that N_{60}, the SPT blow count uncorrected for overburden stress, should be used in relationships of this kind instead of $N_{1,60}$. However, the overburden stress-corrected SPT blow count is used here as presented by Seed et al. (1986).

The number of equivalent cycles are calculated using a low-cycle fatigue (Green and Terri 2005) correlation developed as part of this research (Lasley et al. 2015a). Dissipated energies from equivalent-linear site response analyses were used as the damage metric to obtain the n_{eq} correlations. Thus, the n_{eq} values are consistent with the procedure proposed herein:

$$n_{eq} = \exp\left(b_1 + b_2 \ln(a_{max}) + b_3 M_w\right) \tag{6.12}$$

Regression coefficients (b_1-b_4) for active shallow-crustal tectonic regimes (e.g. the western United States) and for stable continental tectonic regimes (e.g. the central-eastern United States) were provided by Lasley et al. 2015a and are given in Table 6.4. For either tectonic regime, the number of cycles is multiplied by 2 to account for bi-directional motions. The normalized dissipated energy demand term, $\Delta W/\sigma'_{v0}$, can then be calculated as:

$$\frac{\Delta W}{\sigma'_{v0}} = \left(\frac{\Delta W_{1,eq}}{G_{max}}\right) \cdot \frac{G_{max}}{\sigma'_{v0}} \cdot (2 \cdot n_{eq}) \tag{6.13}$$

Alternatively, if G_γ and D_γ are obtained without using either Figure 6.5 or Eqns. 6.9 and 6.10 (e.g. for conditions of mean initial effective stresses greater than 200 kPa), the normalized dissipated energy is calculated as:

$$\frac{\Delta W}{\sigma'_{v0}} = \frac{2\pi \cdot D_\gamma \left[0.65 \cdot a_{max} \cdot \sigma_v \cdot r_d\right]^2}{\sigma'_{v0} \cdot G_\gamma \cdot g^2} \cdot (2 \cdot n_{eq}) \tag{6.14}$$

This is the same general form given by Green (2001), except the initial effective overburden stress is used here instead of the mean effective stress.

6.6.2 Definition of the Limit State Curve

The limit state curve delineates the liquefaction and no-liquefaction regions. The functional form proposed herein was chosen based on the laboratory liquefaction curve and the distribution of liquefaction/no-liquefaction case histories in normalized dissipated energy-SPT blow count space:

$$g = c_1 N_{1,60cs} - \ln\left(\frac{\Delta W}{\sigma'_{v0}}\right) + c_2 + \varepsilon \tag{6.15}$$

where c_1 and c_2 are regression coefficients and ε is an error term with a mean of zero and a standard deviation of σ_ε. If $g \leq 0$, liquefaction is predicted; if $g > 0$ no liquefaction is predicted.

For a given liquefaction case history, if it is assumed that there is no uncertainty in the input parameters and that the error term ε is normally

distributed, the probability of liquefaction is (Boulanger and Idriss 2012):

$$P[g \leq 0] = 1 - \Phi\left[\frac{g}{\sigma_\varepsilon}\right] \tag{6.16}$$

where $\Phi[\cdot]$ is the standard normal cumulative distribution function. Equal-probability curves for normalized dissipated energy are thus:

$$\left(\frac{\Delta W_{liq}}{\sigma'_{v0}}\right)_{P_{liq.}} = \exp\left(c_1 N_{1,60cs} + c_2 + \Phi^{-1}[P] \cdot \sigma_\varepsilon\right) \tag{6.17}$$

where ΔW_{liq} is the dissipated energy required to cause liquefaction under the given conditions. For deterministic forward analyses, a probability of 15% is commonly used to define the limit state curve (e.g. Cetin 2004; Boulanger and Idriss 2012), giving the following:

$$\left(\frac{\Delta W_{liq}}{\sigma'_{v0}}\right) = \exp\left(c_1 N_{1,60cs} + c_2 - 1.036 \cdot \sigma_\varepsilon\right) \tag{6.18}$$

This expression (Eqn. 6.18) is analogous to the CRR curve of stress-based procedures. Using this deterministic limit state curve, a factor of safety can be calculated as:

$$FS_{liq} = \frac{\ln\left(\frac{\Delta W_{liq}}{\sigma'_{v0}}\right) - c_{FS}}{\ln\left(\frac{\Delta W}{\sigma'_{v0}}\right) - c_{FS}} \tag{6.19}$$

where c_{FS} is equal to $1.2 \cdot c_2$. Because the capacity ($\ln(\Delta W_{liq}/\sigma'_{v0})$) and demand ($\ln(\Delta W/\sigma'_{v0})$) terms used to compute the factor of safety are logarithms whose values are negative, c_{FS} is necessary to obtain factor of safety values comparable to those of other simplified procedures.

Alternatively, uncertainties in the input parameters may be included in the calculation of the probability of liquefaction:

$$P[g \leq 0] = \int_\eta \int_\omega 1 - \Phi\left[\frac{g}{\sigma_\varepsilon}\right] f_N(\eta) f_W(\omega)\, d\eta d\omega \tag{6.20}$$

where η is the corrected SPT blow count ($N_{1,60cs}$), ω is the natural logarithm of the normalized dissipated energy ($\Delta W/\sigma'_{v0}$), and $f_N(\eta)$ and $f_W(\omega)$ are the probability density functions of the SPT blow count and the natural logarithm of normalized dissipated energy, respectively.

6.6.3 Regression Approach

The coefficients of g (Eqn. 6.15) are obtained from a maximum likelihood estimation. A likelihood function, by definition, returns the likelihood that the limit state model correctly predicts liquefaction for all liquefied sites, and no liquefaction for all the sites without liquefaction. Thus, the likelihood function is defined as (Cetin 2000; Cetin et al. 2002):

$$\mathcal{L} = \prod_{\text{Liq. Cases}} P[g \leq 0] \quad \times \prod_{\text{Nonliq. Cases}} 1 - P[g \leq 0] \tag{6.21}$$

However, the log-likelihood is used in this study. It is defined as:

$$\ln \mathcal{L} = \sum_{\text{Liq. Cases}} \ln \left(P[g \leq 0] \right) + \sum_{\text{Nonliq. Cases}} \ln \left(1 - P[g \leq 0] \right) \tag{6.22}$$

The log-likelihood can be maximized either by assuming that input values of normalized dissipated energy and SPT blow count are known exactly (using Eqn. 6.16), or by accounting for the uncertainties in each of the input parameters (Eqn. 6.20). The uncertainties in both the normalized dissipated energy and the clean-sand corrected SPT blow count were estimated using a first-order approximation (Ditlevsen and Madsen 1996). Table 6.5 summarizes the uncertainty values used in this estimation.

6.6.4 Performance of the Proposed Procedure

The regression coefficients for Eqn. 6.15 are given in Table 6.6 for four different scenarios:

1. Small-strain shear modulus values were estimated only from $N_{1,60cs}$ using Eqn. 6.11.

2. Small-strain shear modulus values were calculated from measured shear wave velocities, where available, and estimated from $N_{1,60cs}$ where measured shear wave velocity values were not available. The cases with measured shear wave velocities were weighted four times that of the others in the likelihood maximization.
3. Same as #1 except uncertainties in the values of $N_{1,60cs}$ and $\Delta W/\sigma'_{v0}$ were incorporated into the likelihood maximization using Eqn. 6.20.
4. Same as #2 except incorporating uncertainties as in #3.

Figure 6.8 shows the limit state curves obtained from scenario #1 for probabilities of 15, 50, and 85%. Similar to the stress-based simplified procedures, at low SPT blow counts the curves become flat and at high SPT blow counts the curves are nearly vertical. This latter fact indicates that at blow counts above $\sim$ 30, most soils are dense enough to experience dilation, inhibiting liquefaction. At low SPT blow counts and low energies, shear straining below the threshold shear strain will also preclude the initiation of liquefaction. In this paper, the limit state curves in these areas are plotted as dashed lines to highlight these conditions. Figure 6.9 shows the same limit state curves with the normalized dissipated energy on a logarithmic scale.

The median limit state curve ($P = 50\%$) is shown in Figures 6.10 through 6.13 with the liquefaction/no-liquefaction case histories for various ranges of fines content, initial effective overburden stress, peak ground acceleration, and magnitude. For all given ranges of fines content, initial effective overburden stress, peak ground acceleration, and magnitude, the dissipated-energy limit state curve does well dividing the liquefaction and no-liquefaction case histories.

Table 6.7 compares the predictive performance of the four scenarios for this case history database. Included also in the table are the Boulanger and Idriss (2012) and Cetin (2004) liquefaction procedures using their median ($P = 50\%$) liquefaction curves. The competing liquefaction evaluation scenarios/procedures are compared by examining the number of case histories that were correctly and incorrectly predicted. The incorrectly predicted points can be further separated as the number of false positive and number of false negative events. In this case, a false positive event is an event that was predicted to liquefy but did not; a false negative event was predicted to not liquefy but did liquefy. In addition to these counts, the predictive ability of the models was also evaluated using the Nash-Sutcliffe model efficiency

coefficient, E. For this comparison, the value of E is calculated as follows (Nash and Sutcliffe 1970):

$$E = 1 - \frac{\sum_{i=1}^{m}(g_{\text{Actual}} - \text{sign}(g_{\text{Predicted}}))^2}{\sum_{i=1}^{N}(g_{\text{Actual}} - \overline{g_{\text{Actual}}})^2} \tag{6.23}$$

where g_{Actual} indicates whether the case history liquefied and has values of -1, 0, or 1 for liquefaction, marginal liquefaction, and no liquefaction, respectively. The 'sign()' function returns values of -1, 0, or 1 depending on the sign of the input, and $g_{\text{Predicted}}$ is obtained using Eqn. 6.15 (with the regression values of scenario #1), the Boulanger and Idriss (2012) procedure, or the Cetin et al. (2004) procedure. The variable m is the number of case histories in the database. The Nash-Sutcliffe model efficiency coefficient, E, can range in value from 1 to -∞. A value of unity indicates the model is a perfect fit to the data, and a value of less than zero indicates that the mean value of the actual case history results ($\overline{g_{\text{Actual}}}$) is a better predictor than the model. Since the sign function is used in Eqn. 6.23, the value of E is simply a single-point summary of the number of correct and incorrect predictions.

As seen in Table 6.7, the Boulanger and Idriss (2012) and the Cetin et al. (2004) limit state curves predict almost equally well for this case history database, with the Boulanger and Idriss (2012) procedure predicting slightly better with 197 out of 230 case histories predicted correctly. This is not surprising because their limit state curve was regressed using this case history data. The Cetin et al. (2004) procedure predicted 195 case histories correctly. The dissipated energy scenarios 1 and 3 predicted 193 and 198 cases correctly, respectively. Scenario 3 and Boulanger and Idriss (2012) have the same number of false negatives (20) and nearly the same number of false positive predictions of 12 and 13, respectively. The Cetin et al. (2004) prediction, on the other hand, had only 9 false negative predictions, indicating a limit state curve shifted slightly toward lower CSR for a given SPT blow count, in comparison to the other two procedures.

Scenarios 2 and 4, using the measured shear wave velocities where available, had much poorer performance overall; the measured shear wave velocities introduce an additional source of scatter into the dissipated energy calculations. For all case histories with measured shear wave velocity, the G_{max}

values calculated from the measured shear wave velocities were less than those estimated using the Seed et al. (1986) equation (6.11). To investigate the effect of Eqn. 6.11 on the overall analyses, new regression coefficients for Eqn. 6.11 were obtained using the measured shear wave velocity values and the corresponding $N_{1,60cs}$ values. New regression coefficients for the limit state curve were obtained for all scenarios using this alternate G_{max}-$N_{1,60cs}$ correlation and numbered from 5 to 8. The effect of using the new correlation between G_{max} and $N_{1,60cs}$ is to increase the uncertainty in the location of the limit state curve (σ_ε) and decrease the accuracy for the scenarios that calculate G_{max} from $N_{1,60}$ alone (scenarios 1 & 5 and 3 & 7; see Tables 6.8 and 6.9). The opposite is true for the scenarios that use the measured shear wave velocities (scenarios 2 & 6 and 4 & 8); uncertainties were decreased and the accuracy was increased. The dissipated energy to liquefaction is increased for all cases using the alternate correlation, as shown by the changes in the value of c_2 between Tables 6.6 and 6.8. The differences in the value of c_1 are less pronounced. Because the original Seed et al. (1986) equation has precedence and the resulting limit state curves have a slightly higher success rate in prediction, the regression coefficients given for scenarios 1 and 3 are suggested for forward analyses with SPT blow counts. However, the limit state curves for scenarios 6 and 8 may be more accurate for forward analyses involving measured shear wave velocities.

When incorporating uncertainties into the likelihood function via Eqn. 6.20 (scenarios 3 and 4), the predictive ability (measured in terms of the prediction of liquefaction or no-liquefaction for each the case histories) was improved. The uncertainty in the location of the limit state curve (σ_ε) was reduced, likely because the uncertainty in the input parameters masks uncertainty in the location of the limit state curve. A similar reduction in uncertainty was experienced by Boulanger and Idriss (2012).

6.6.5 Performance Compared to Boulanger and Idriss (2012)

Since the liquefaction evaluation procedure proposed herein was developed from the same case history database as Boulanger and Idriss (2012), it is fitting that the two procedures are compared in more detail. The two procedures give similar results, and, for this case history database, have very

similar success rates in prediction. In this section, the median liquefaction curves ($P = 50\%$; $g = 0$) are used as the liquefaction criteria (i.e. a case history was predicted to liquefy if the calculated probability of liquefaction was greater than 50%).

Despite the similarity in overall success rate, the two procedures disagree in a few cases. Figure 6.14 shows each procedure with its limit state curves corresponding to probabilities of liquefaction of 15, 50, and 85% (using the regression coefficients and uncertainty of scenario #1). The small white squares indicate the case histories on which the two procedures agree and are correct. The stars and dots indicate case histories for which the two procedures wrongly agree and disagree, respectively. For these latter two cases, the marker is color-coded to show the reported liquefaction: dark markers are cases that were reported to have liquefied and white markers (except for the square markers) are case histories that reported no liquefaction. Almost all of the cases where the two procedures disagree or are both wrong fall within the bounds set by the 15 and 85% probability curves. Some exceptions are given by a few no-liquefaction case histories at low SPT blow counts (white dots at 5-10 blows/30cm). The Boulanger and Idriss (2012) procedure correctly predicted no-liquefaction in these cases; all of these cases are below the $P = 50\%$ curve and four are below the $P = 15\%$ curve. The energy-based procedure, however, predicts liquefaction for these cases.

Figure 6.15 shows how the two procedures compare for different values of SPT blow count, peak ground acceleration, fines content, initial effective overburden stress, and magnitude. The locations of the points in the y-direction indicate whether the case history reported no liquefaction, marginal liquefaction, or liquefaction. It can be seen that the Boulanger and Idriss (2012) procedure more often correctly predicts no liquefaction at low SPT blow counts and low peak ground acceleration where the procedure proposed herein more often correctly predicts liquefaction at low SPT blow counts and low peak ground acceleration. Similarly, the Boulanger and Idriss procedure appears to be biased towards predicting liquefaction at low fines content. No bias is apparent for either procedure for the range of initial effective overburden stress. Finally, the procedure proposed herein appears to better predict liquefaction at low magnitudes when compared to the Boulanger and Idriss (2012) procedure. However, this result may be related to the aforementioned trends at low SPT blow counts and peak ground accelerations.

6.6.6 Behavior at High Effective Overburden Pressures

Youd et al. (2001) and most stress-based procedures since then have suggested the use of K_σ to correct the soil capacity (CRR) for initial effective overburden stresses other than 1 atm. Although the Cetin et al. (2004) procedure directly incorporates the correction as part of the limit state curve, Boulanger and Idriss (2012) maintain a separate K_σ term to apply to the CRR or factor of safety. The procedure proposed herein, however, does not require an overburden stress correction factor since the normalized dissipated energy required for liquefaction is constant over a range of initial effective overburden stresses, for a constant value of $N_{1,60cs}$.

To illustrate the consistency of the procedure proposed herein, dissipated energy principles were used to back-calculate an overburden stress correction factor for stress-based procedures. An inherent part of this is the shear modulus reduction and damping curves of Darendeli and Stokoe (2001) that are a function of mean initial effective stress. The back-calculated correction factor was obtained by scaling the applied shear stress, τ, until the normalized dissipated energy in one hysteresis loop for a given initial effective overburden stress was equal to the normalized dissipated energy in a hysteresis loop for an overburden stress of 1 atm and a given shear stress, $\tau_{\sigma'_{v0}=1\text{ atm}}$. The correction factor is simply the ratio of these stresses, normalized by initial effective overburden stress:

$$K_\sigma = \frac{\tau/\sigma'_{v0}}{\tau_{\sigma'_{v0}=1\text{ atm}}/1\text{ atm}} \tag{6.24}$$

Figure 6.16 shows the K_σ curves back-calculated from the normalized dissipated energy along with the Hynes and Olsen (1999) and Boulanger and Idriss (2004) curves for a range of soil densities. The Hynes and Olsen (1999) curves were suggested by the Youd et al. (2001) procedure, and the Boulanger and Idriss (2004) curves were suggested by the Idriss and Boulanger (2008) liquefaction procedure. The dissipated-energy-derived curves are insensitive to soil density but are affected by the parameters used to obtain the shear modulus reduction and damping curves. The figure shows a derived K_σ curve for 'clean sand' and another for 'sand with high fines content' as given by the Darendeli and Stokoe (2001) degradation curves. The Hynes and Olsen

(1999) curves are given for relative densities between 40 and 60% only; thus only the Boulanger and Idriss (2004) curves are shown for a relative density of 30%. The dissipated-energy-derived curve for clean sand falls alongside the Hynes and Olsen (1999) curve for a relative density of 50%- right in the middle of the range given by the Hynes and Olsen curves. The derived curve for sand with high fines content lies between the curves given by Hynes and Olsen (1999) and those given by Boulanger and Idriss (2004).

While the normalized dissipated energy to liquefaction (or capacity term) is insensitive to overburden stress, the demand term is affected by the change in initial effective overburden stress and shows a different trend with depth when compared to the Cetin et al. (2004) and Boulanger and Idriss (2012) stress-based procedures. Figure 6.17 shows normalized demand, normalized capacity, and the factor of safety versus initial effective overburden stress for all three procedures for a soft profile and low intensity event. The corrected SPT blow count is constant with depth and the ground water table is shallow (0.5m). The demand and capacity are normalized by the respective demand and capacity at an initial effective overburden stress of 1 atm. For the stress-based procedures, the demand is the cyclic stress ratio (Eqn. 6.1).

All three demand curves show a peak value at low overburden stresses with the demand thereafter decreasing with depth. This peak is caused by the multiplication of the stress reduction coefficient that decreases with depth and the ratio of total and initial effective overburden stresses that increases with depth. The demand curve of the procedure proposed herein is also affected by the stiffening of the soil layer with increasing initial effective stress as predicted by the shear modulus reduction and damping curves (i.e. D_γ decreases and G_γ increases for a given shear strain in Eqn. 6.14).

Since the capacity curves in Figure 6.17 are normalized and incorporate the K_σ correction factor, the stress-based curves are the K_σ curves of the respective procedures. The capacity curve corresponding to the procedure proposed herein is constant with increasing initial effective overburden stress, reflecting the lack of overburden correction factor. The resulting factor of safety curves further illustrate the differences in the three procedures. For this earthquake scenario, the Boulanger and Idriss (2012)-predicted factor of safety briefly dips below the $FS = 1$ line, then returns to moderately-high factors of safety at high initial effective overburden stresses. The Cetin et al. (2004) procedure predicts liquefaction at nearly all initial effective over-

burden stresses, and the procedure proposed herein predicts low factors of safety at low initial effective overburden stresses, then a factor of safety of around unity as the initial effective overburden stresses increase.

Figure 6.18 shows the trends in demand, capacity, and factor of safety for an event of M_w7.5, $a_{max} = 0.35g$, and an SPT blow count of 30. In this case, the demand curve of the proposed procedure has a peak at higher initial effective overburden stresses than the other two due to the influence of the shear modulus reduction and damping curves. This results in a factor of safety that remains lower for a greater range of low initial effective overburden stresses but increases with increasing initial effective overburden stress. For both events shown in Figures 6.17 and 6.18, the factor of safety of the procedure proposed herein changes less abruptly with increasing initial effective overburden stress than the factors of safety from the two stress-based procedures.

6.6.7 Application with Site Response Analyses

With conventional stress-based simplified procedures, site response analyses may be utilized to obtain a better estimate of the maximum shear stress at depth, thus bypassing the stress reduction coefficient, r_d. However, if any of the components of the stress-based procedure (e.g. r_d, MSF) are biased, the cyclic stress ratio obtained from site response analyses should not be compared directly with the stress-based capacity curve.

In contrast, the energy-based procedure proposed herein was consistently developed with equivalent-linear site response analyses in mind. For example, both the number of equivalent cycles, n_{eq}, and r_d correlations were developed from the results of equivalent-linear site response analyses using 50 liquefaction site profiles (from Cetin 2000) and 194 pairs of horizontal rock motions from 47 earthquake events. For each layer of each site and motion combination, the number of equivalent cycles was calculated using dissipated energy as the damage metric (for details see Lasley et al. 2015a). Thus for a site that is well-characterized, equivalent-linear site response analysis can be performed, and the dissipated energy can be calculated using Eqn. 6.3. This dissipated energy incorporates the damaging effects of the entire motion, not just the peak value. Once normalized by the initial effective overburden stress, this normalized dissipated energy can be used directly in Eqns. 6.15,

6.16, and 6.19 to determine the predicted probability of liquefaction and factor of safety.

In this way, the energy-based procedure proposed herein could be used in more advanced site response analyses that modify the soil behavior once initial liquefaction occurs. For example, the strength of a soil element may be reduced to the residual strength once liquefaction occurs, thus influencing the response of adjacent soil elements. The proposed procedure provides a convenient indicator of initial liquefaction under these circumstances, provided that the total-stress dissipated energy is calculated in the analyses.

To ease the estimation of dissipated energy from equivalent-linear site response, both ShakeVT2 (Lasley et al. 2014; https://bitbucket.org/slasley/shakevt2.git) and *Strata* (Rathje and Kottke 2010; Kottke and Rathje 2008) have the option to compute the dissipated energy from an equivalent linear site response analysis and include the Darendeli and Stokoe (2001) shear modulus reduction and damping curves.

The easy application of equivalent-linear site response analyses also allows the proposed limit state curves to be used in conjunction with any cyclic loadings including non-earthquake seismic loadings such as those from remedial ground densification and other mechanical vibrations.

6.7 Comparison of Lab- and Field-Based Capacity Curves

In order to compare the lab-based capacity curve (Figure 6.4) to the field-based capacity curve, the following relationship was used (Meyerhof 1957; Idriss and Boulanger 2008):

$$D_r \approx 15\sqrt{N_{1,60}} \tag{6.25}$$

where D_r is in percent. Figure 6.19 shows the lab-based capacity curve plotted as a function of SPT blow count with the field-based curves (scenario 1) of 15, 50, and 85% probability of liquefaction. While both curves show similar trends of increasing dissipated energy to liquefaction with increasing SPT blow count, the slopes and intercepts disagree. Additional research is

needed to understand the differences between the curves. However, some of the difference may be attributed to the fact that the lab curve is a liquefaction triggering curve and the field curve is a manifestation curve.

6.8 Conclusion

Proposed herein is an energy-based simplified liquefaction evaluation procedure. Soil liquefaction poses a hazard to life and property in seismically-active regions. The current state-of-practice for evaluating liquefaction susceptibility is the use of any of a number of stress-based simplified procedures based on field case histories. Instead of using the cyclic stress ratio in conjunction with a magnitude scaling factor to quantify capacity and demand, the procedure proposed herein uses dissipated energy normalized by the initial effective overburden stress. This normalized dissipated energy has been shown in previous studies to be a good indicator of soil structure breakdown and consequent liquefaction.

To ease adoption, the proposed procedure is presented in a form similar to the stress-based procedures and is a function of SPT blow count, peak ground acceleration, and earthquake magnitude. The result of the procedure is a probability of liquefaction and/or a factor of safety against liquefaction. For forward analyses, the regression coefficients for scenario #3 are recommended. However, the uncertainty (σ_ε) in the location of the limit state curve should not be regarded as less than the value given for scenario #1. Thus the recommended capacity curve corresponding to a probability of liquefaction of 15% is:

$$\frac{\Delta W_{liq}}{\sigma'_{v0}} = \exp(0.30 N_{1,60cs} - 12.95) \tag{6.26}$$

For a given normalized dissipated energy and corrected SPT blow count, the probability of liquefaction can be estimated as:

$$P_{liq} = 1 - \Phi \left[\frac{0.30 N_{1,60cs} - \ln\left(\frac{\Delta W}{\sigma'_{v0}}\right) - 11.61}{1.29} \right] \tag{6.27}$$

where $\Phi[\cdot]$ is the standard normal cumulative distribution function.

Compared to the Cetin et al. (2004) and Boulanger and Idriss (2012) liquefaction evaluation procedures, the proposed energy-based procedure has a similar success rate when predicting the occurrence of liquefaction for the set of cases histories from which it was calibrated. Differences between this procedure and the others can be seen at high initial effective overburden stresses beyond those of the case history database. The difference is manifest as a greater reduction in demand as well as a constant capacity with increasing initial effective overburden stress.

Because the procedure proposed herein uses normalized dissipated energy for its demand and capacity terms, the resulting limit state curve is insensitive to the shape of the input ground motion. This means the proposed procedure is more applicable to other tectonic regimes and to non-earthquake sources of cyclic loading. Additionally, the procedure proposed herein has been designed for use with equivalent-linear site response analyses allowing for greater flexibility and detail in liquefaction evaluations.

6.9 Acknowledgements

This research is partially funded by National Science Foundation (NSF) grants CMMI-1030564 and CMMI-1306261. This support is gratefully acknowledged. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

6.10 References

Berrill, J. B., and Davis, R. O. (1985). "Energy dissipation and seismic liquefaction of sands: revised model." *Soils and Foundations*, 25(2), 106–118.

Boulanger, R. W. and Idriss, I. M., (2004). "State normalization of penetration resistances and the effect of overburden stress on liquefaction resistance," *Proceedings, 11th International Conference on Soil Dynamics and Earthquake Engineering, and 3rd International Conference on Earthquake*

Geotechnical Engineering, D. Doolin et al., eds., Stallion Press, Vol. 2, pp. 484–91.

Boulanger, R., and Idriss, I. (2012). "Probabilistic standard penetration test–based liquefaction–triggering procedure." *Journal of Geotechnical and Geoenvironmental Engineering*, ASCE, 138(10), 1185–1195.

Boulanger, R. W., Wilson, D. W., and Idriss, I. M. (2012). "Examination and reevalaution of SPT-based liquefaction triggering case histories." *Journal of Geotechnical and Geoenvironmental Engineering*, ASCE, 138(8), 898–909.

Cetin, K. Ö. (2000). "Reliability-based assessment of seismic soil liquefaction initiation hazard." University of California, Berkeley, Berkeley, CA.

Cetin, K. Ö., Der Kiureghian, A., and Seed, R. B. (2002). "Probabilistic models for the initiation of seismic soil liquefaction." *Structural Safety*, 24(1), 67–82.

Cetin, K. Ö., Seed, R. B., Der Kiureghian, A., Tokimatsu, K., Harder Jr, L. F., Kayen, R. E., and Moss, R. E. (2004). "Standard penetration test-based probabilistic and deterministic assessment of seismic soil liquefaction potential." *Journal of Geotechnical and Geoenvironmental Engineering*, 130(12), 1314–1340.

Darendeli, M. B., and Stokoe, K. H. (2001). Development of a new family of normalized modulus reduction and material damping curves. Geotechnical Engineering Report GD01-1, University of Texas at Austin.

Davis, R. O., and Berrill, J. B. (1982). "Energy dissipation and seismic liquefaction in sands." *Earthquake Engineering & Structural Dynamics*, 10(1), 59–68.

Davis, R. O., and Berrill, J. B. (1996). "Liquefaction susceptibility based on dissipated energy: a consistent design methodology." *Bulletin of the New Zealand National Society for Earthquake Engineering*, 29(2), 83–91.

Ditlevsen, O., and Madsen, H. O. (1996). *Structural Reliability Methods*. Wiley.

Figueroa, J. L., Saada, A. S., Liang, L., and Dahisaria, N. M. (1994). "Evaluation of soil liquefaction by energy principles." *Journal of Geotechnical Engineering*, 120(9), 1554–1569.

Finn, W. D. ., Vaid, Y. P., and Bhatia, S. K. (1979). "Constant volume cyclic

simple shear testing." *Proceedings of the Second International Conference on Microzonation for Safer Construction - Research and Application*, 839 – 851.

Green, R. A., Mitchell, J. K., and Polito, C. P. (2000). "An energy-based excess pore pressure generation model for cohesionless soils." *Proc.: Developments in Theoretical Geomechanics–The John Booker Memorial Symposium*, 383–390.

Green, R. A. (2001). *Energy-Based Evaluation and Remediation of Liquefiable Soils.* Ph.D. Dissertation, Virginia Polytechnic Institute and State University, Blacksburg, Virginia.

Green, R. A., and Terri, G. A. (2005). "Number of equivalent cycles concept for liquefaction evaluations—Revisited." *Journal of Geotechnical and Geoenvironmental Engineering*, ASCE, 131(4), 477–488.

Hynes, M. E., and Olsen, R. S. (1999). "Influence of confining stress on liquefaction resistance." *Proc., Int. Workshop on Phys. and Mech. of Soil Liquefaction*, Balkema, Rotterdam, The Netherlands, 145–152.

Idriss, I. M., and Boulanger, R. W. (2008). *Soil Liquefaction During Earthquakes.* Earthquake Engineering Research Institute, Oakland, California.

Jacobsen, L.S. (1960). "Damping in composite structures." *Proceedings of the Second World Conference on Earthquake Engineering*, 2, 1028-1044.

Jafarian, Y., Towhata, I., Baziar, M. H., Noorzad, A., and Bahmanpour, A. (2012). "Strain energy based evaluation of liquefaction and residual pore water pressure in sands using cyclic torsional shear experiments." *Soil Dynamics and Earthquake Engineering*, 35, 13–28.

Kayen, R., Moss, R. E. S., Thompson, E. M., Seed, R. B., Cetin, K. O., Kiureghian, A. D., Tanaka, Y., and Tokimatsu, K. (2013). "Shear-wave velocity–based probabilistic and deterministic assessment of seismic soil liquefaction potential." *Journal of Geotechnical and Geoenvironmental Engineering*, ASCE, 139(3), 407–419.

Kokusho, T. (2013). "Liquefaction potential evaluations: energy-based method versus stress-based method." *Canadian Geotechnical Journal*, 50(10), 1088–1099.

Kottke, A. R., and Rathje, E. M. (2008) "Technical Manual for Strata." PEER Report 2008/10. University of California, Berkeley, California.

Lasley, S. J., Green, R. A., and Rodriguez-Marek, A. (2014). "Comparison of equivalent-linear site response analysis software." *Proceedings of the 10th U.S. National Conference on Earthquake Engineering*, Earthquake Engineering Research Institute, Anchorage, AK.

Lasley, S.J., Green, R.A., and Rodriguez-Marek, A. (2015a) "Development of a number of equivalent cycles correlation based on dissipated energy." *In preparation.*

Lasley, S.J., Green, R.A., and Rodriguez-Marek, A. (2015b) "A new stress reduction coefficient relationship for liquefaction triggering analyses." *In preparation.*

Law, K. T., Cao, Y. L., and He, G. N. (1990). "An energy approach for assessing seismic liquefaction potential." *Canadian Geotechnical Journal*, 27(3), 320–329.

Liang, L. (1995). "Development of an energy method for evaluating the liquefaction potential of a soil deposit." Ph.D. Dissertation, Case Western Reserve University.

Liang, L., Figueroa, J. L., and Saada, A. S. (1995). "Liquefaction under random loading: Unit energy approach." *Journal of Geotechnical Engineering*, 121(11), 776–781.

Mayfield, R. T. (2007). "The Return Period of Soil Liquefaction." Ph.D. Dissertation, University of Washington, Department of Civil and Environmental Engineering.

Meyerhof, G. G., 1957. "Discussion on research on determining the density of sands by spoon penetration testing," *Proceedings, 4th International Conference on Soil Mechanics and Foundation Engineering*, London, Vol. 3, p. 110.

Moss, R. E. S. (2003). "CPT-Based Probabilistic Assessment of Seismic Soil Liquefaction Initiation." PhD Dissertation, University of California, Berkeley, CA.

Nash, J. E., and Sutcliffe, J. V. (1970). "River flow forecasting through conceptual models part I—A discussion of principles." *Journal of Hydrology*, 10(3), 282–290.

Nemat-Nasser, S., and Shokooh, A. (1979). "A unified approach to densification and liquefaction of cohesionless sand in cyclic shearing." *Canadian*

Geotechnical Journal, 16(4), 659–678.

Ohta, Y. and Goto, N. (1976). "Estimation of S-wave velocity in terms of characteristic indices of soil." *Butsuri-Tanko*, 29(4), 34-41.

Rathje, E.M., Kottke, A. (2010), "Strata," http://nees.org/resources/strata.

Rodriguez-Marek, A., Rathje, E. M., Bommer, J. J., Scherbaum, F., and Stafford, P. J. (2014). "Application of Single-Station Sigma and Site-Response Characterization in a Probabilistic Seismic-Hazard Analysis for a New Nuclear Site." *Bulletin of the Seismological Society of America*, 104(4), 1601–1619.

Seed, H. B., and Idriss, I. M. (1971). "Simplified procedure for evaluating soil liquefaction potential." *Journal of the Soil Mechanics and Foundations Division*, 97(9), 1249–1273.

Seed, H. B., Idriss, I. M., and Arango, I. (1983). "Evaluation of liquefaction potential using field performance data." *Journal of Geotechnical Engineering*, 109(3), 458–482.

Seed, H. B., Tokimatsu, K., Harder, L. F., Jr., and Chung, R. (1984). "The influence of SPT procedures in soil liquefaction resistance evaluations." Earthquake Eng. Research Center Rep. No. UCB/EERC-84/15, University of California at Berkeley, Berkeley, CA.

Seed, H. B., Wong, R. T., Idriss, I. M., and Tokimatsu, K. (1986). "Moduli and Damping Factors for Dynamic Analyses of Cohesionless Soils." *Journal of Geotechnical Engineering*, 112(11), 1016–1032.

Simcock, K. J., Davis, R. O., Berrill, J. B., and Mullenger, G. (1983). "Cyclic triaxial tests with continuous measurement of dissipated energy." *Geotechnical Testing Journal*, 6(1), 35–39.

Sivathayalan, S., and Ha, D. (2011). "Effect of static shear stress on the cyclic resistance of sands in simple shear loading." *Canadian Geotechnical Journal*, 48(10), 1471–1484.

Tokimatsu, K., and Seed, H. B. (1987). "Evaluation of settlements in sands due to earthquake shaking." *Journal of Geotechnical Engineering*, 113(8), 861–878.

Trifunac, M. D. (1995). "Empirical criteria for liquefaction in sands via standard penetration tests and seismic wave energy." *Soil Dynamics and Earthquake Engineering*, 14(6), 419–426.

Vaid, Y. P., and Negussey, D. (1988). "Preparation of reconstituted sand specimens." *Advanced Triaxial Testing of Soil and Rock*, ASTM STP, 977, 405–417.

Vaid, Y. P., and Sivathayalan, S. (1996). "Static and cyclic liquefaction potential of Fraser Delta sand in simple shear and triaxial tests." *Canadian Geotechnical Journal*, 33(2), 281–289.

Wair, B. R., DeJong, J. T., and Shantz, T. (2012). Guidelines for Estimation of Shear Wave Velocity Profiles. Pacific Earthquake Engineering Research Center, University of California, 95.

Whitman, R. V. (1971). "Resistance of Soil to Liquefaction and Settlement." *Soils and Foundations*, 11(4), 59–68.

Youd, T. L., Idriss, I. M., Andrus, R. D., Arango, I., Castro, G., Christian, J. T., Dobry, R., Finn, W. D. L., Harder, L. F., Hynes, M. E., Ishihara, K., Koester, J. P., Liao, S. S. C., Marcuson, W. F., Martin, G. R., Mitchell, J. K., Moriwaki, Y., Power, M. S., Robertson, P. K., Seed, R. B., and Stokoe, K. H. (2001). "Liquefaction resistance of soils: summary report from the 1996 NCEER and 1998 NCEER/NSF workshops on evaluation of liquefaction resistance of soils." *Journal of Geotechnical and Geoenvironmental Engineering*, ASCE, 127(10), 817–833.

6.11 Tables

Table 6.1: Regression Coefficients for r_d (Lasley et al. 2015b)

EQ Regime	a_1	a_2	a_3	a_4	σ
Active Shallow-Crustal	-4.031	0.364	-28.47	7.798	0.1397
Stable Continental	-4.369	0.439	-15.38	4.922	0.1586

Table 6.2: Regression Coefficients for the Fit of the $\Delta W_{1,eq}/G_{max}$ Curves (Clean Sand)

σ'_{m0} (kPa)	d_1	d_2	d_3	d_4	d_5	d_6
25	0.242	5.848	56.101	267.91	640.3145	608.05276
50	0.232	5.545	52.685	249.053	589.1456	553.24074
100	0.22	5.219	49.129	229.977	538.6714	500.38834
200	0.208	4.881	45.541	211.152	489.8841	450.27458

Table 6.3: Regression Coefficients for the Fit of the $\Delta W_{1,eq}/G_{max}$ Curves (Sand with High Fines Content)

σ'_{m0} (kPa)	d_1	d_2	d_3	d_4	d_5	d_6
25	-0.0249	-0.4056	-1.7385	3.74134	45.391	80.934471
50	0.015	0.4993	6.2671	38.1141	116.1858	135.54471
100	0.0401	1.043	10.814	56.2139	149.4765	156.46374
200	0.0544	1.33	12.98	63.487	158.8015	156.90162

Table 6.4: Regression Coefficients for n_{eq} (Lasley et al. 2015a)

EQ Regime	b_1	b_2	b_3	b_4
Active Shallow-Crustal	0.806	-0.298	-8.98e-4	0.254
Stable Continental	0.557	-0.390	-1.80e-4	0.315

EQ Regime	b_1	b_2	b_3	b_4

Table 6.5: Values Used in the First-Order Approximation of Uncertainty. μ denotes the mean, σ denotes the standard deviation, and δ is the coefficient of variation or σ/μ

Parameter	Value	Notes
σ_D	Varies	Darendeli and Stokoe (2001), function of shear strain
$\sigma_{G/G_{max}}$	Varies	Darendeli and Stokoe (2001)
$\rho_{D,G/G_{max}}$	-0.5	Rodriguez-Marek et al. (2014)
$\delta_{\ln PGA}$	0.3	Cetin (2000)
σ_{r_d}	0.15	From model regression
δ_{σ_v}	0.1	From assumed uncertainties in γ and gwt
$\delta_{\sigma'_{v0}}$	0.1	From assumed uncertainties in γ and gwt
$\rho_{\sigma_v,\sigma'_{v0}}$	-1	Based on uncertainty in the location of the water table
σ_γ	0.1	Moss (2003)
μ_γ	19.5 kN/m^2	Assumed; 17.0 above water table
δ_{V_s}	0.5	When calculated from SPT; 0.25 when measured (B.R. Cox, personal communication, 2015)
$\sigma_{\ln n_{eq}}$	0.53	From model regressions
$\rho_{r_d,\ln n_{eq}}$	-0.434	From model regressions/data
$\delta_{N_{1,60}}$	0.2	Or $\sigma = 3$; from Cetin (2000)
δ_{FC}	0.2	Cetin (2000)

Table 6.6: Regression Coefficients for the Proposed Limit State Curve

Scenario	c_1	c_2	σ_ϵ
#1 - $G_{max} = f(N_{1,60cs})$	0.2691	-11.35	1.292
#2 - $G_{max} = f(V_{s,\text{ meas.}}, N_{1,60cs})$	0.3141	-10.9	3.368
#3 - $G_{max} = f(N_{1,60cs})$, with Uncertainties	0.2991	-11.61	4.944e-07
#4 - $G_{max} = f(V_{s,\text{ meas.}}, N_{1,60cs})$, with Uncert.	0.266	-10.18	1.937

Table 6.7: Success Rate of Prediction of the Occurrence of Liquefaction for the Case History Database

Scenario/Procedure	Correct	False Pos.	False Neg.	E
#1 - $G_{max} = f(N_{1,60cs})$	193	17	20	0.388
#2 - $G_{max} = f(V_{s,\text{ meas.}}, N_{1,60cs})$	182	11	37	0.194
#3 - $G_{max} = f(N_{1,60cs})$, with Uncertainties	198	12	20	0.476
#4 - $G_{max} = f(V_{s,\text{ meas.}}, N_{1,60cs})$, with Uncert.	181	11	38	0.176
Boulanger and Idriss 2012	197	13	20	0.458
Cetin et al. 2004	195	26	9	0.423

Table 6.8: Regression Coefficients for the Proposed Limit State Curve Using an Alternate G_{max}-$N_{1,60}$ Correlation. The regression coefficients given in Table 6.6 are recommended for forward analyses using SPT blow count.

Scenario	c_1	c_2	σ_ϵ
#5 - $G_{max} = f(N_{1,60cs})$	0.2889	-9.58	1.61
#6 - $G_{max} = f(V_{s,\text{ meas.}}, N_{1,60cs})$	0.2773	-9.27	2.324
#7 - $G_{max} = f(N_{1,60cs})$, with Uncertainties	0.2841	-9.249	2.698e-06
#8 - $G_{max} = f(V_{s,\text{ meas.}}, N_{1,60cs})$, with Uncert.	0.2771	-9.38	1.395

Table 6.9: Success Rate of Prediction of the Occurrence of Liquefaction of Limit State Curves Using Alternative G_{max}-$N_{1,60}$ Correlation

Scenario/Procedure	Correct	False Pos.	False Neg.	E
#5 - $G_{max} = f(N_{1,60cs})$	193	15	22	0.388
#6 - $G_{max} = f(V_{s,\text{ meas.}}, N_{1,60cs})$	193	14	23	0.388
#7 - $G_{max} = f(N_{1,60cs})$, with Uncertainties	194	11	25	0.405
#8 - $G_{max} = f(V_{s,\text{ meas.}}, N_{1,60cs})$, with Uncert.	192	16	22	0.37

6.12 Figures

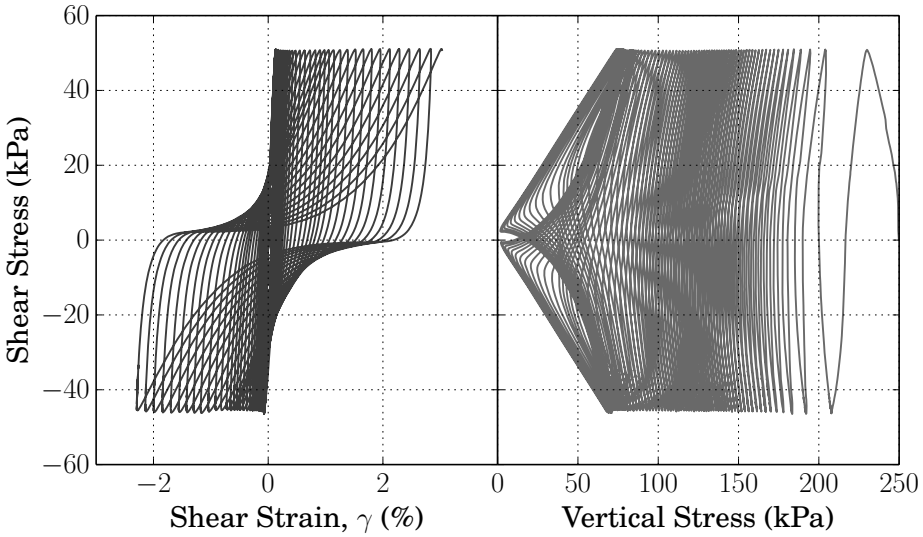


Figure 6.1: Results of a constant-volume cyclic simple shear test: Left: Shear stress versus shear strain hysteresis loops, Right: Shear stress versus vertical stress.

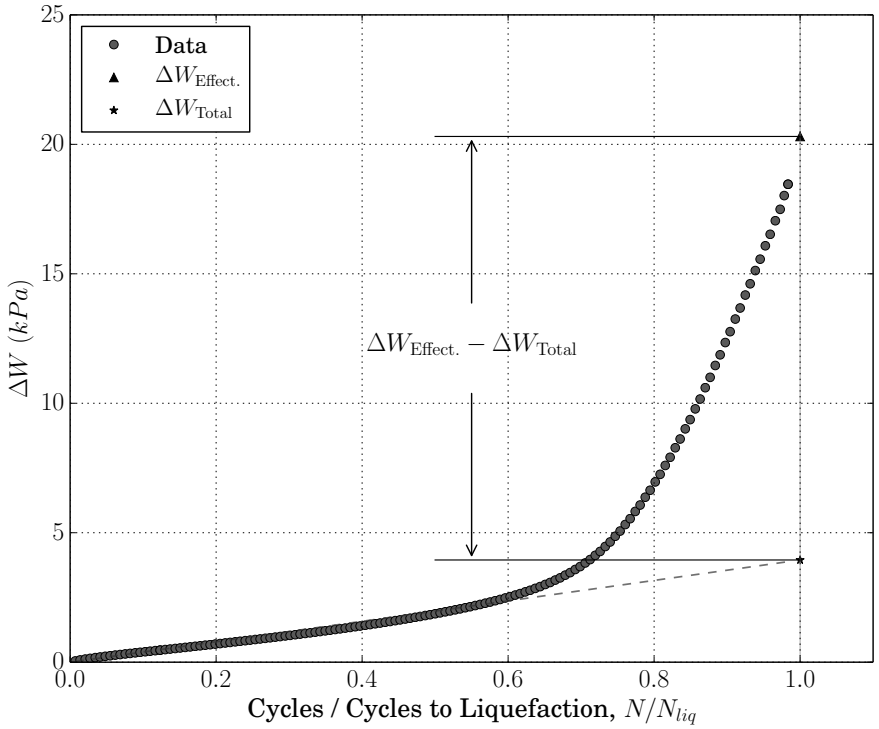


Figure 6.2: Results of a constant-volume cyclic simple shear test: Dissipated energy increasing to liquefaction. A best fit line over the linear portion of the dissipated energy curve ($\Delta W_{\text{Effect.}}$) is used to obtain the dissipated energy corresponding to total stress analyses (ΔW_{Total}). The difference between the two measures of dissipated energy is illustrated here and can be attributed to the softening of the soil with increasing pore water pressure.

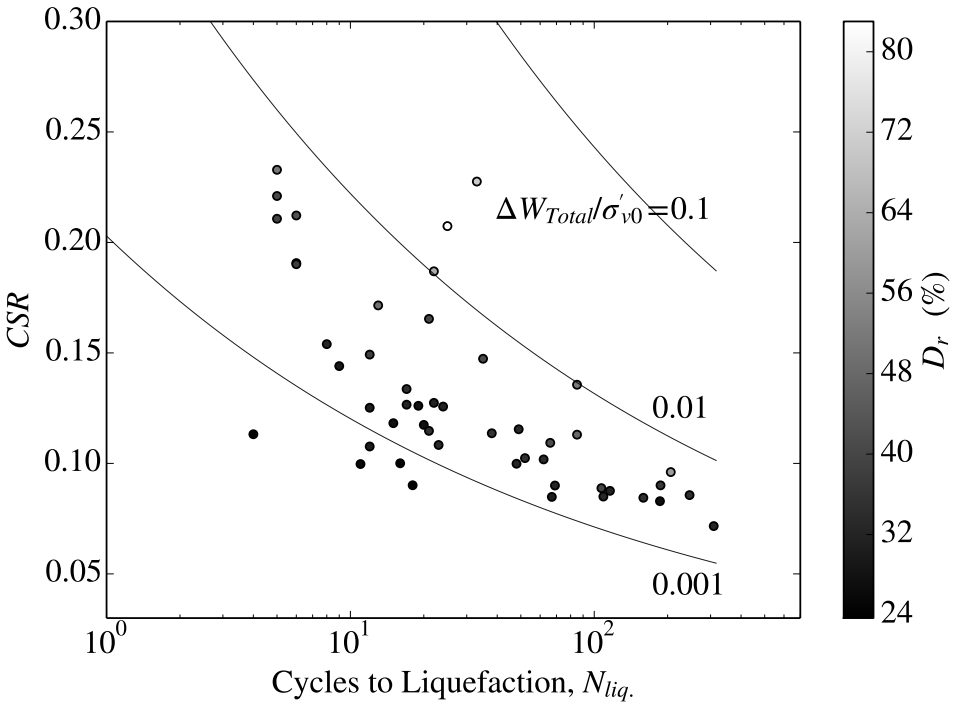


Figure 6.3: Results of cyclic simple shear tests with contours of equal dissipated energy to liquefaction.

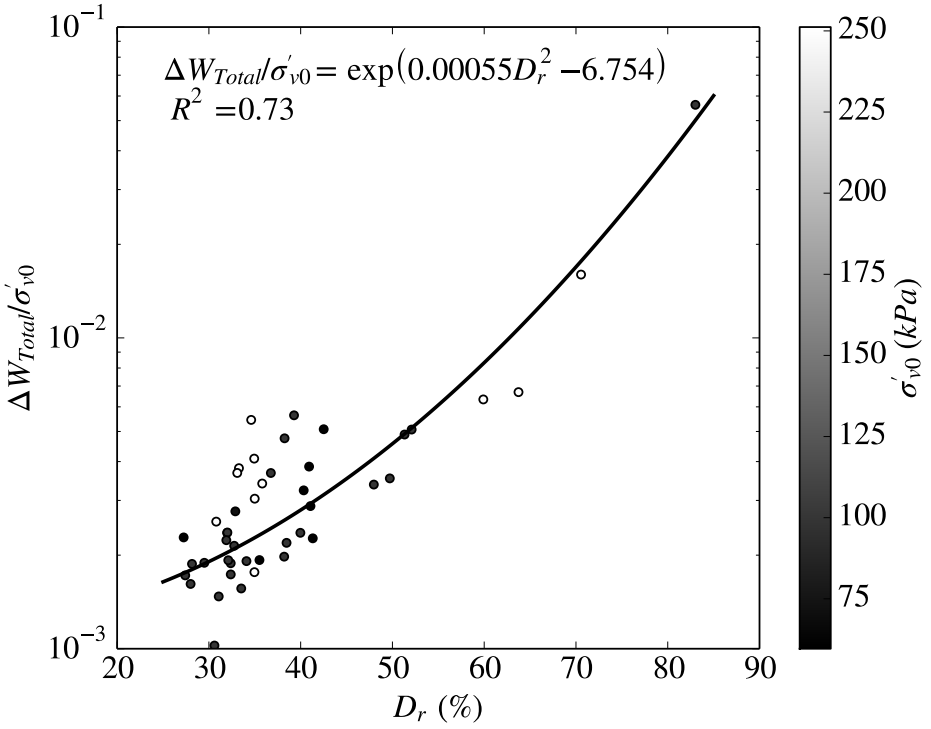


Figure 6.4: Results of cyclic simple shear tests plotted with normalized dissipated energy to liquefaction (corrected to remove softening effects) versus relative density.

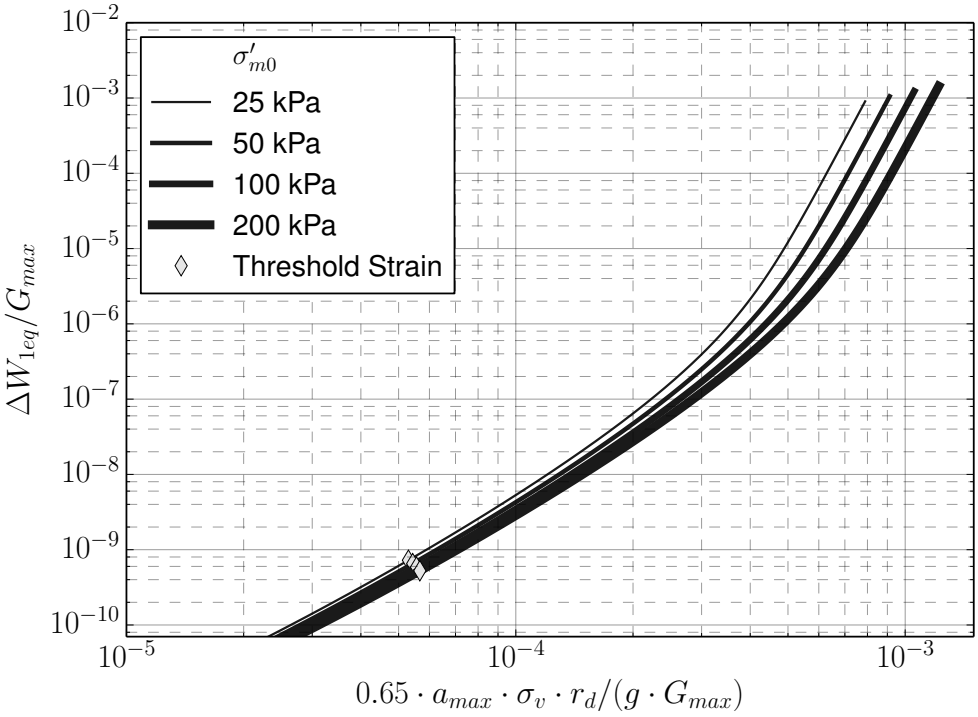


Figure 6.5: Direct Method to estimate $\Delta W_{1eq}/G_{max}$ (after Tokimatsu and Seed 1987) using the Darendeli and Stokoe (2001) degradation curves. Lines correspond to a clean sand ($FC = 0\%$, $PI = 0$). To use the figure, calculate the value on the x-axis, read up to the appropriate curve, and obtain the corresponding value from the y-axis.

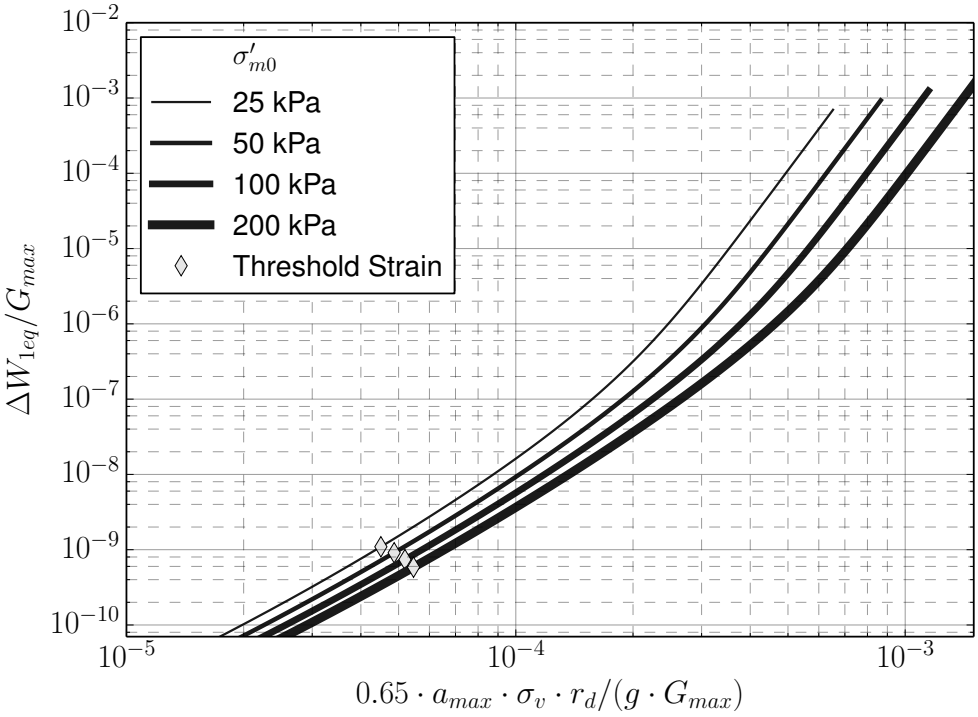


Figure 6.6: Direct Method to estimate $\Delta W_{1eq}/G_{max}$ (after Tokimatsu and Seed 1987) using the Darendeli and Stokoe (2001) degradation curves. Lines correspond to "sands with high fines content." To use the figure, calculate the value on the x-axis, read up to the appropriate curve, and obtain the corresponding value from the y-axis.

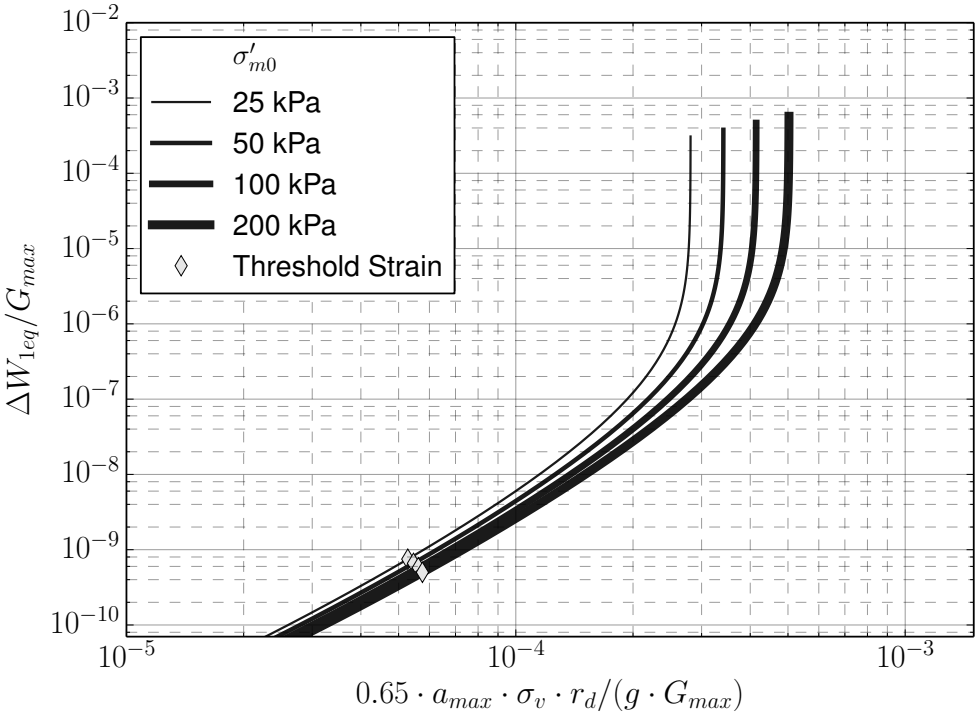


Figure 6.7: Direct Method to estimate $\Delta W_{1eq}/G_{max}$ (after Tokimatsu and Seed 1987) using the Darendeli and Stokoe (2001) degradation curves. Lines correspond to a low-plasticity silt. To use the figure, calculate the value on the x-axis, read up to the appropriate curve, and obtain the corresponding value from the y-axis.

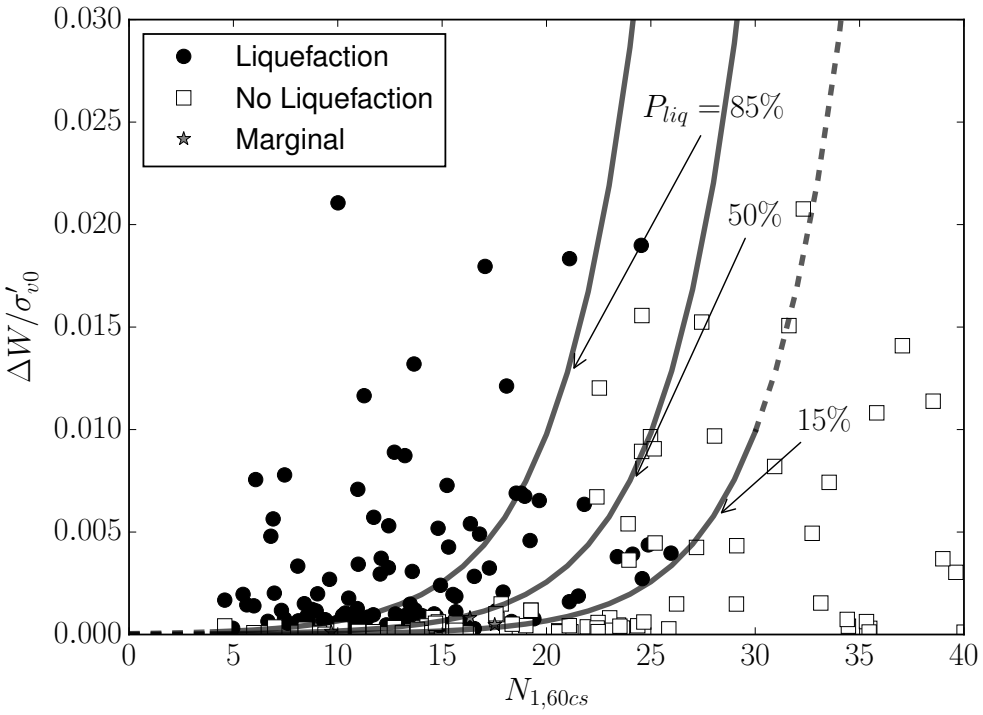


Figure 6.8: Proposed limit state curve (scenario #1) with case history data.

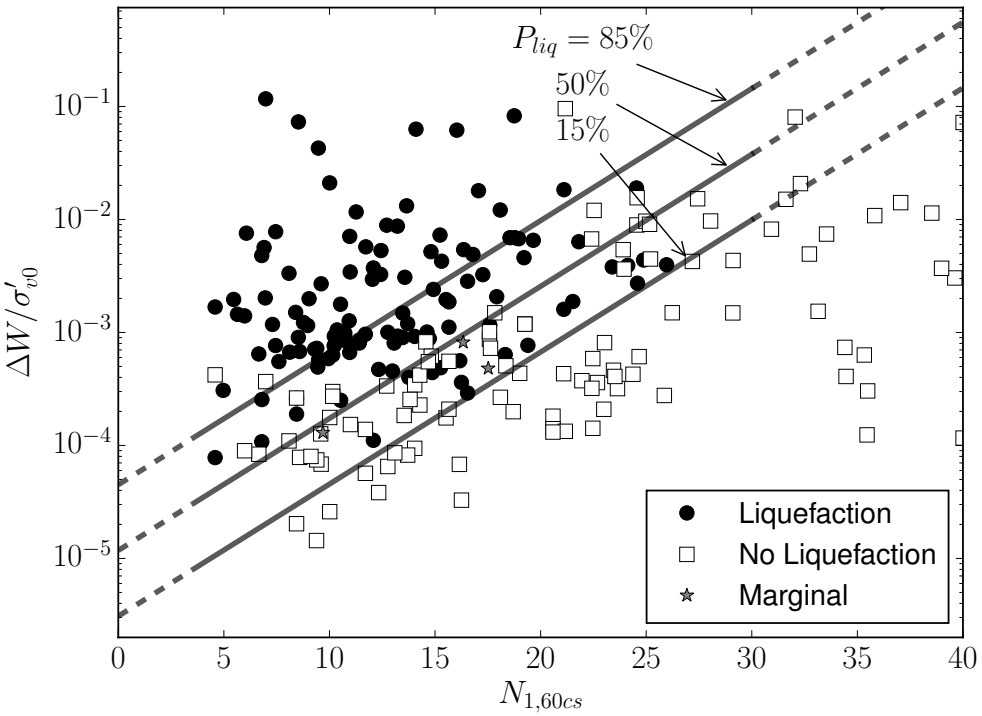


Figure 6.9: Proposed limit state curve (scenario #1) with case history data and normalized dissipated energy on a log axis.

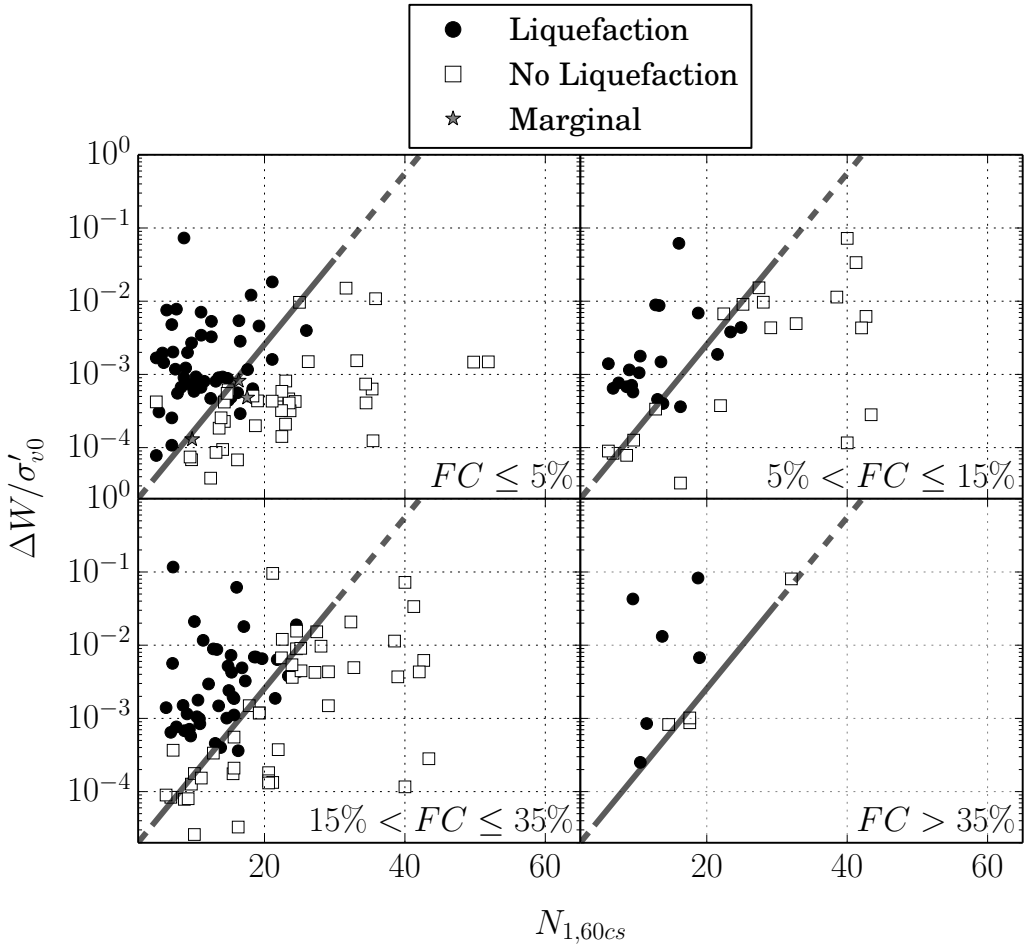


Figure 6.10: Proposed limit state curve (scenario #1, $P = 50\%$) with case history data for various ranges of fines content.

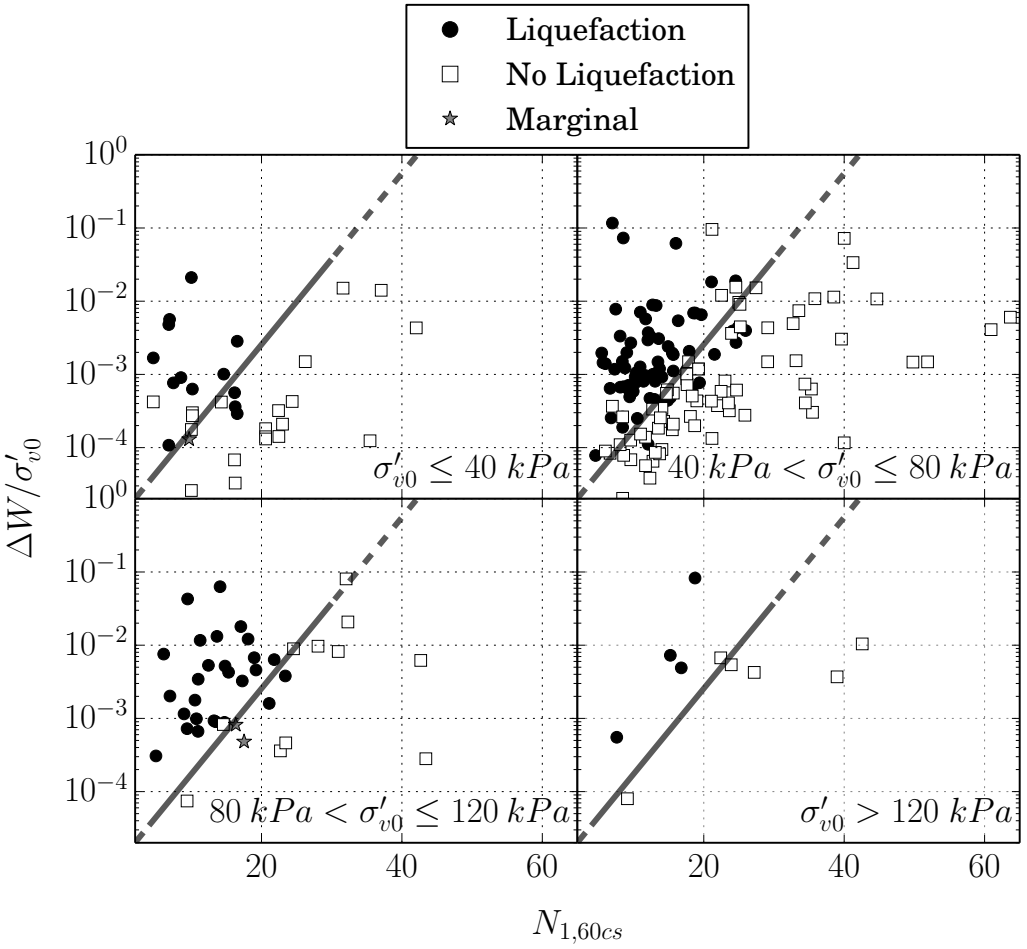


Figure 6.11: Proposed limit state curve (scenario #1, $P = 50\%$) with case history data for various ranges of initial effective overburden stress.

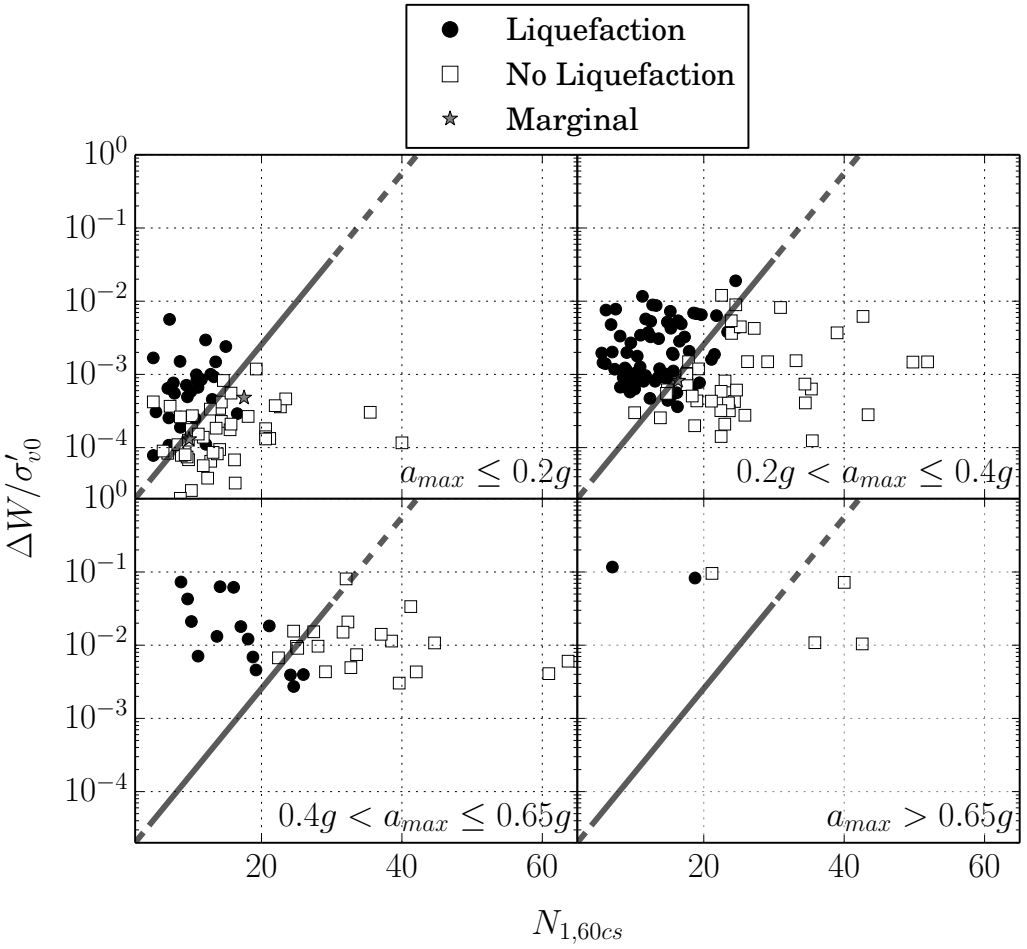


Figure 6.12: Proposed limit state curve (scenario #1, $P = 50\%$) with case history data for various ranges of peak ground acceleration.

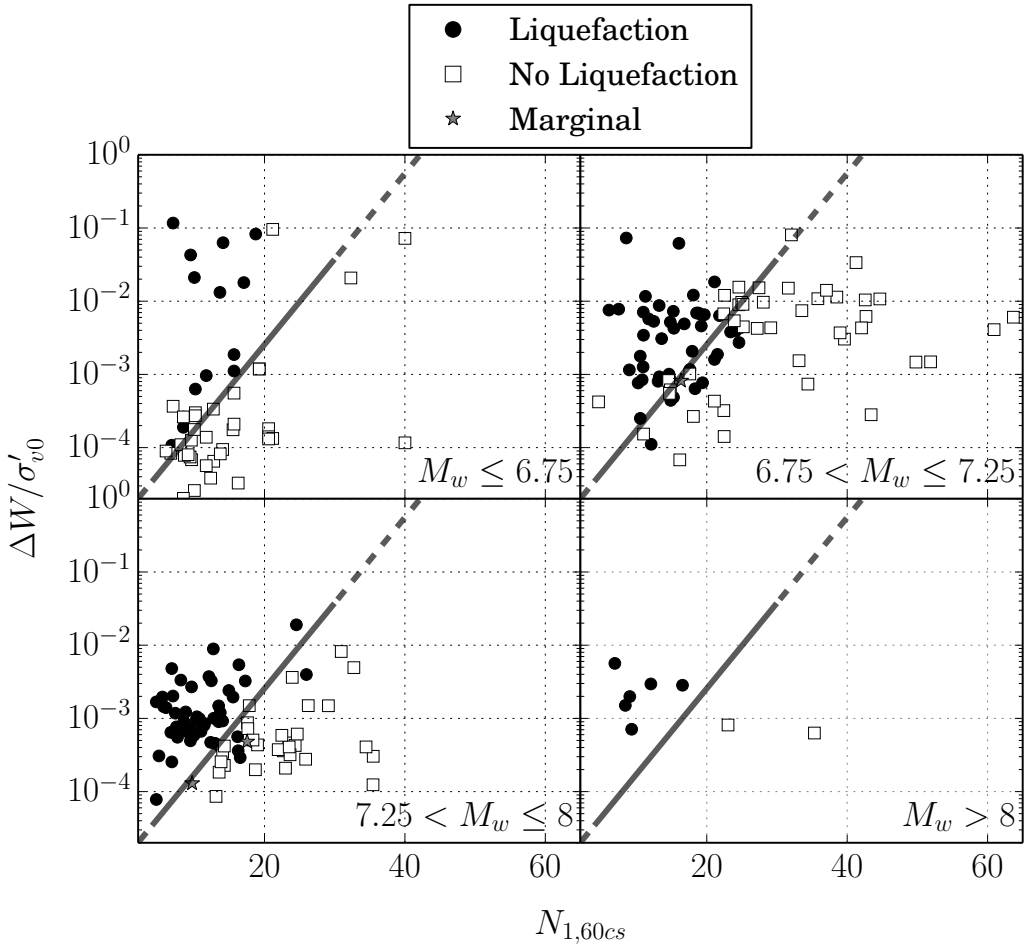


Figure 6.13: Proposed limit state curve (scenario #1, $P = 50\%$) with case history data for various ranges of moment magnitude.

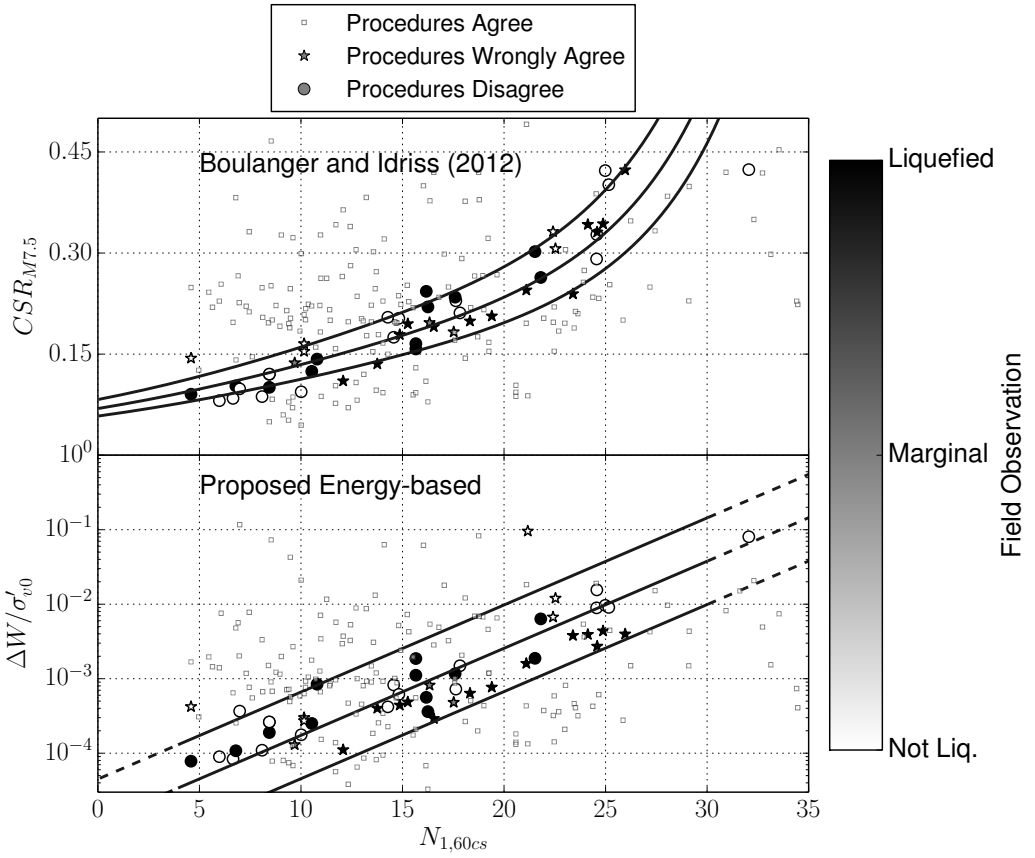


Figure 6.14: Comparison of proposed procedure and the Boulanger and Idriss (2012) procedure. Solid lines indicate the 15, 50, and 85% probabilities of liquefaction. Large dots signify case history data points for which the models do not agree; the color of the point indicates whether the case history site liquefied. White squares indicate points on which the two procedures agree correctly, and the stars indicate points where the two procedures agree incorrectly.

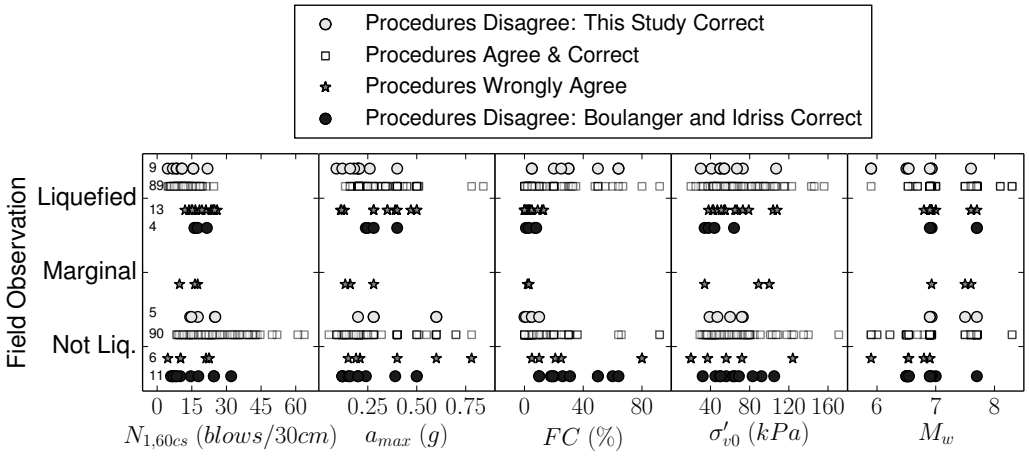


Figure 6.15: Comparison of proposed procedure and Boulanger and Idriss (2012). Points are separated on the y-axis by whether the case history liquefied or not. The shape of the points indicate which of the procedures predicted the case history correctly. The numbers in the left-most pane indicate the number of case histories in each row.

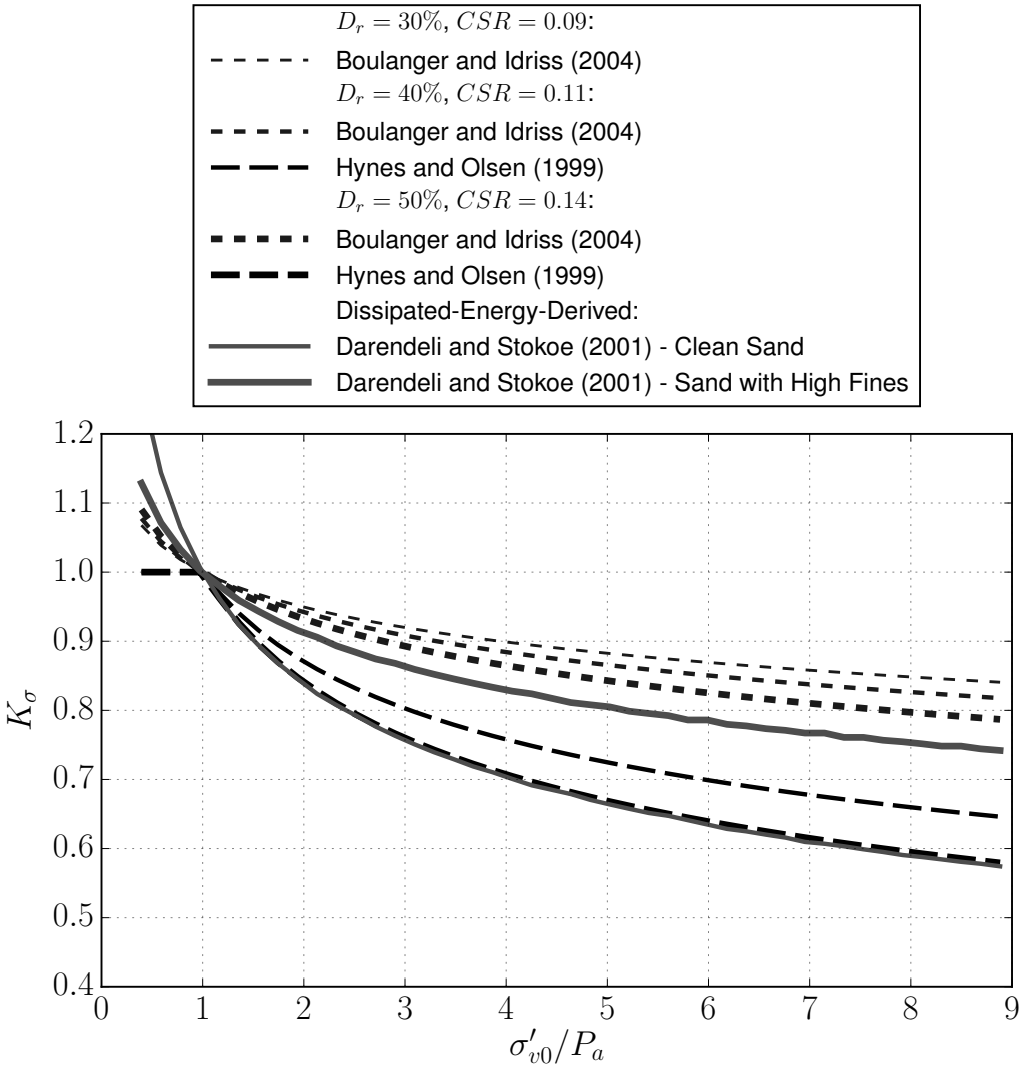


Figure 6.16: Overburden correction factor developed for stress-based methods assuming $\Delta W_{liq}/\sigma'_{v0}$ is insensitive to extreme overburden stresses.

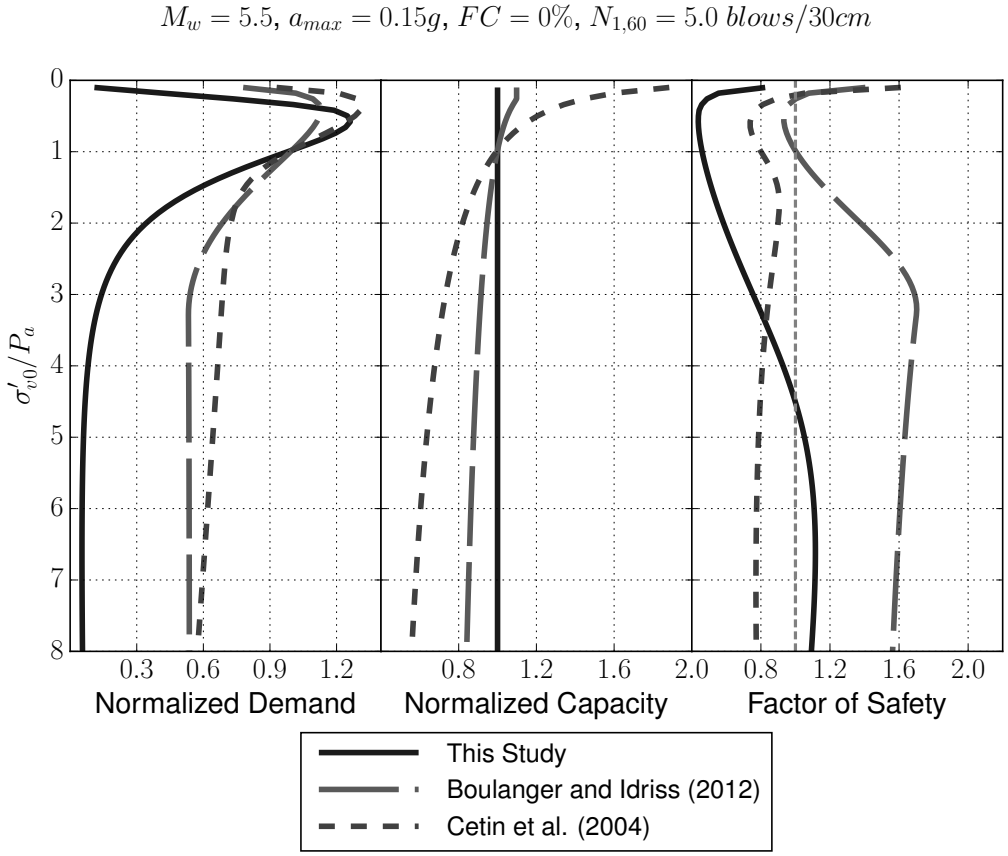


Figure 6.17: Demand, capacity, and factor of safety as a function of initial effective overburden stress for a $M_w 5.5$, $a_{max} = 0.15g$ event. The ground water table is located at a shallow depth (0.5 m) for all initial effective overburden stresses.

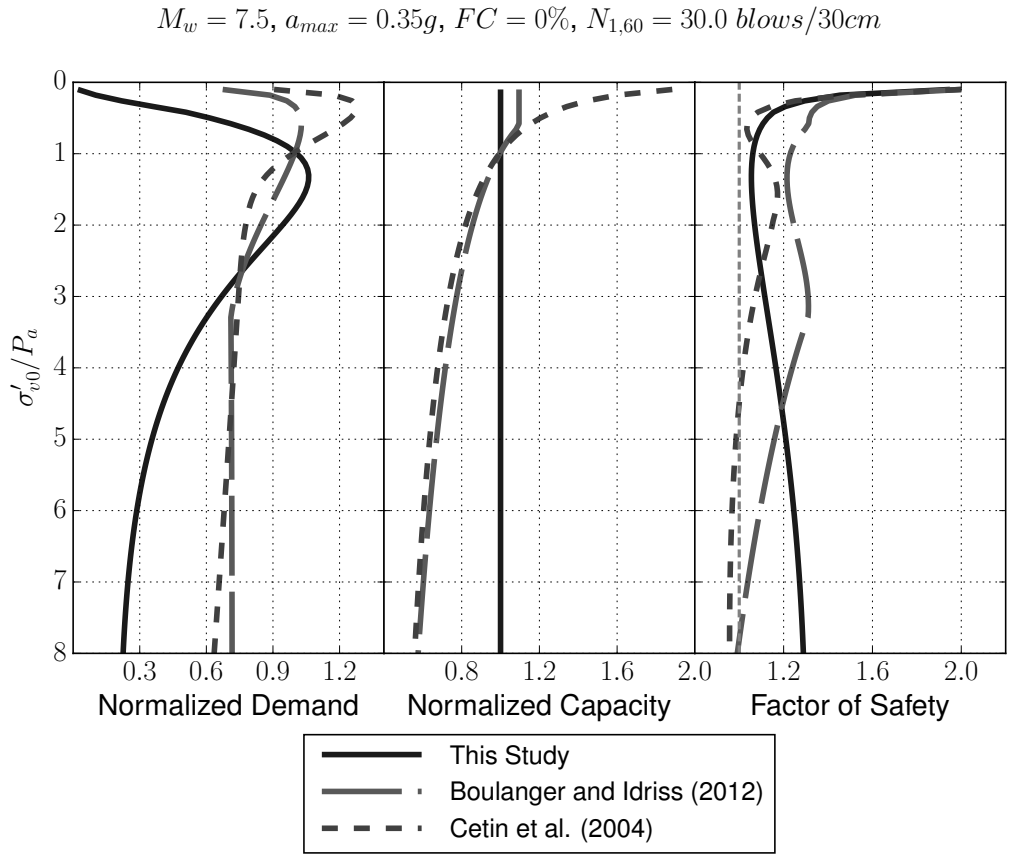


Figure 6.18: Demand, capacity, and factor of safety as a function of initial effective overburden stress for a M_w7.5, $a_{max} = 0.35g$ event. The ground water table is located at a shallow depth (0.5 m) for all initial effective overburden stresses.

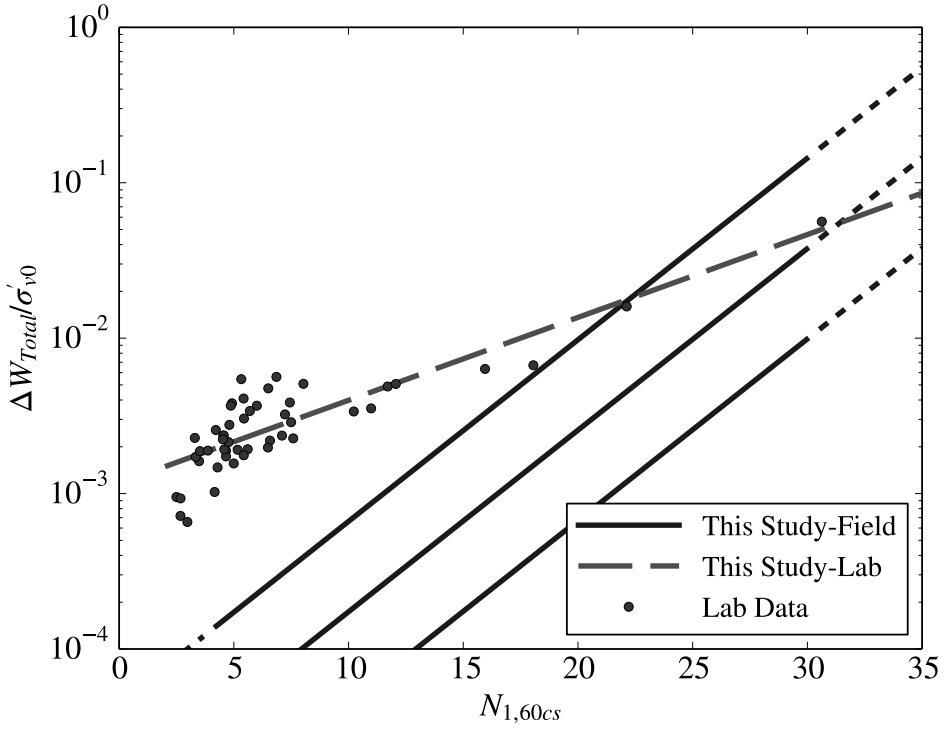


Figure 6.19: Proposed limit state curve (scenario #1, P = 15, 50, & 85%) compared with the laboratory data.

6.13 Electronic Supplement

The SPT case history database compiled by Boulanger et al. (2012)[1] was used in this research to develop a liquefaction limit state curve as a function of dissipated energy and SPT blow count. Table 6.10 summarizes the data provided by their data files. Table 6.11 provides the data calculated as part of this research.

Table 6.10: SPT Case History Database- Earthquake and Site Info

No.	**Earthquake**/Site	Liq?	M_w	PGA	$N_{1,60cs}$	z	σ_v	σ'_v
				(g)		(m)	(kPa)	(kPa)
	1944 Tohnankai							
1	Komei	Yes	8.1	0.2	9.3	5.2	98.0	68.0
2	Ienaga	Yes	8.1	0.2	8.4	4.3	80.0	61.0
3	Meiko	Yes	8.1	0.2	6.9	3.7	69.0	39.0
	1948 Fukui							
4	Shonenji Temple	Yes	7.0	0.4	11.7	4.0	75.0	48.0
5	Takaya 45	Yes	7.0	0.35	21.1	7.5	141.0	104.0
	1964 Niigata							
6	Arayamotomachi	Yes	7.6	0.09	4.6	3.3	63.0	41.0
7	Cc17-1	Yes	7.6	0.16	10.0	7.0	132.0	72.0
8	Cc17-2	Yes	7.6	0.16	13.0	5.3	85.0	43.0
9	Kawagishi-cho	Yes	7.6	0.16	6.8	3.8	71.0	53.0
10	Old Town -1	No	7.6	0.18	22.7	7.0	132.0	81.0
11	Old Town -2	No	7.6	0.18	23.5	10.1	190.0	109.0
12	Rail Road-1	Yes	7.6	0.16	11.0	10.1	190.0	100.0
13	Rail Road-2	Marginal	7.6	0.16	17.5	10.1	190.0	100.0
14	River Site	Yes	7.6	0.16	9.4	4.6	86.0	47.0
15	Road Site	No	7.6	0.18	14.0	6.1	115.0	79.0
16	Showa Br 2	Yes	7.6	0.16	7.4	4.3	80.0	39.0

[1]Boulanger, R., D. Wilson, and I. Idriss. "Examination and Reevalaution of SPT-Based Liquefaction Triggering Case Histories." *Journal of Geotechnical and Geoenvironmental Engineering*, 138, no. 8 (2012): 898-909.

No.	**Earthquake**/Site	Liq?	M_w	PGA	$N_{1,60cs}$	z	σ_v	σ'_v
17	Showa Br 4	No	7.6	0.18	35.5	6.1	115.0	67.0
	1968 earthquake-April 1							
18	Hososhima	No	7.5	0.24	17.6	2.9	53.0	45.0
	1968 Tokachi-Oki							
19	Aomori Station	Yes	8.3	0.21	16.5	5.7	95.0	38.0
20	Hachinohe -2	No	8.3	0.23	35.3	6.1	115.0	76.0
21	Hachinohe -4	No	8.3	0.23	23.0	4.0	75.0	45.0
22	Hachinohe-6	Yes	8.3	0.23	9.0	4.0	75.0	42.0
23	Nanaehama1-2-3	Yes	8.3	0.2	12.0	4.0	75.0	45.0
	1971 San Fernando							
24	Juvenile Hall	Yes	6.6	0.45	9.5	6.1	112.0	96.0
25	Van Norman	Yes	6.6	0.45	13.7	6.1	112.0	96.0
	1975 Haicheng							
26	Panjin Chemical Fertilizer Plant	Yes	7.0	0.2	13.2	8.2	155.0	89.0
27	Shuang Tai Zi River Sluice Gate	No	7.0	0.2	14.6	8.2	158.0	92.0
28	Ying Kou Glass Fibre Plant	Yes	7.0	0.3	19.0	7.8	147.0	85.0
29	Ying Kou Paper Plant	Yes	7.0	0.3	11.0	8.2	158.0	92.0
	1976 Guatemala							
30	Amatitlan B-1	Yes	7.5	0.14	5.0	10.4	139.0	86.0
31	Amatitlan B-2	Marginal	7.5	0.14	9.7	4.6	55.0	34.0
32	Amatitlan B-3&4	No	7.5	0.14	14.3	10.7	137.0	71.0
	1976 Tangshan							
33	Coastal Region	Yes	7.6	0.13	13.8	4.5	87.0	54.0

No.	**Earthquake**/Site	Liq?	M_w	PGA	$N_{1,60cs}$	z	σ_v	σ'_v
34	Le Ting L8-14	Yes	7.6	0.2	13.5	4.4	81.0	53.0
35	Luan Nan-L1	No	7.6	0.22	24.4	3.5	62.0	38.0
36	Luan Nan-L2	Yes	7.6	0.22	8.5	3.5	56.0	32.0
37	Qing Jia Ying	Yes	7.6	0.35	24.5	5.3	102.0	59.0
38	Tangshan City	No	7.6	0.5	32.7	5.3	98.0	75.0
39	Yao Yuan Village	Yes	7.6	0.2	14.9	6.1	118.0	67.0
	1977 Argentina							
40	San Juan B-1	Yes	7.5	0.2	10.7	8.2	142.0	106.0
41	San Juan B-3	Yes	7.5	0.2	7.6	11.1	199.0	156.0
42	San Juan B-4	No	7.5	0.2	14.3	3.7	63.0	39.0
43	San Juan B-5	No	7.5	0.2	13.5	3.1	53.0	44.0
44	San Juan B-6	Yes	7.5	0.2	11.4	5.2	90.0	56.0
	1978 Miyagiken-Oki							
45	Arahama (A-9)	No	6.5	0.1	12.8	6.4	121.0	67.0
46	Hiyori-18 (site C)	No	6.5	0.14	15.5	5.2	98.0	71.0
47	Ishinomaki-2	No	6.5	0.12	6.7	3.5	66.0	45.0
48	Kitawabuchi-2	No	6.5	0.14	12.3	3.4	62.0	59.0
49	Nakajima-18 (site A)	No	6.5	0.14	14.0	6.1	115.0	79.0
50	Nakamura Dyke N-4	Yes	6.5	0.12	6.8	2.8	53.0	30.0
51	Nakamura Dyke N-5	No	6.5	0.12	9.6	3.4	63.0	42.0
52	Oiiri-1	No	6.5	0.14	9.4	6.4	106.0	85.0
53	Shiomi-6 (Site D)	No	6.5	0.14	8.6	4.0	75.0	60.0
54	Yuriage Br-1	No	6.5	0.12	6.0	4.3	80.0	56.0
55	Yuriage Br-2	No	6.5	0.12	16.3	2.5	46.0	34.0
56	Yuriage Br-3	No	6.5	0.12	12.7	4.3	80.0	42.0
57	Yuriagekami-1	No	6.5	0.12	8.1	5.5	99.0	63.0
58	Yuriagekami-2	No	6.5	0.12	13.7	4.3	80.0	47.0
	1978 Miyagiken-Oki							
59	Arahama (A-9)	Yes	7.7	0.2	12.8	6.4	121.0	67.0
60	Hiyori-18 (site C)	Yes	7.7	0.24	15.5	5.2	98.0	71.0
61	Ishinomaki-2	Yes	7.7	0.2	6.7	3.5	66.0	45.0

No.	**Earthquake**/Site	Liq?	M_w	PGA	$N_{1,60cs}$	z	σ_v	σ'_v
62	Ishinomaki-4	No	7.7	0.2	22.0	4.5	87.0	57.0
63	Kitawabuchi-2	Yes	7.7	0.28	12.3	3.4	62.0	59.0
64	Kitawabuchi-3	No	7.7	0.28	17.6	4.8	90.0	73.0
65	Nakajima 2	No	7.7	0.24	17.8	4.6	86.0	65.0
66	Nakajima-18 (site A)	Yes	7.7	0.24	14.0	6.1	115.0	79.0
67	Nakamura Dyke N-1	No	7.7	0.32	26.2	3.4	63.0	39.0
68	Nakamura Dyke N-4	Yes	7.7	0.32	6.8	2.8	53.0	30.0
69	Nakamura Dyke N-5	Yes	7.7	0.32	9.6	3.4	63.0	42.0
70	Oiiri-1	Yes	7.7	0.24	9.4	6.4	106.0	85.0
71	Shiomi-6 (Site D)	Yes	7.7	0.24	8.6	4.0	75.0	60.0
72	Yuriage Br-1	Yes	7.7	0.24	6.0	4.3	80.0	56.0
73	Yuriage Br-2	Yes	7.7	0.24	16.3	2.5	46.0	34.0
74	Yuriage Br-3	Yes	7.7	0.24	12.7	4.3	80.0	42.0
75	Yuriage Br-5	No	7.7	0.24	24.0	7.3	138.0	78.0
76	Yuriagekami-1	Yes	7.7	0.24	8.1	5.5	99.0	63.0
77	Yuriagekami-2	Yes	7.7	0.24	13.7	4.3	80.0	47.0
78	Yuriagekami-3	No	7.7	0.24	24.7	5.5	103.0	70.0
	1979 Imperial Valley							
79	Heber Road A1	No	6.5	0.78	40.0	2.9	53.0	42.0
80	Heber Road A2	Yes	6.5	0.78	7.0	3.7	68.0	50.0
81	Heber Road A3	No	6.5	0.78	21.2	4.0	79.0	56.0
82	Kornbloom B	No	6.5	0.13	11.7	4.3	77.0	62.0
83	McKim Ranch A	Yes	6.5	0.51	10.0	2.1	38.0	32.0
84	Radio Tower B1	Yes	6.5	0.2	8.4	3.4	62.0	50.0
85	Radio Tower B2	No	6.5	0.2	20.6	2.3	40.0	38.0
86	River Park A	Yes	6.5	0.24	10.2	1.8	35.0	20.0
87	Wildlife B	No	6.5	0.17	15.7	4.6	87.0	54.0
	1980 Mid-Chiba							
88	Owi-1	No	6.0	0.1	9.6	6.1	108.0	57.0
89	Owi-2	No	6.0	0.1	9.1	14.3	254.0	123.0
	1981 WestMorland							

No.	**Earthquake**/Site	Liq?	M_w	PGA	$N_{1,60cs}$	z	σ_v	σ'_v
90	Kornbloom B	Yes	5.9	0.32	11.7	4.3	77.0	62.0
91	McKim Ranch A	No	5.9	0.09	10.0	2.1	38.0	32.0
92	Radio Tower B1	Yes	5.9	0.2	8.4	3.4	62.0	50.0
93	Radio Tower B2	No	5.9	0.2	20.6	2.3	40.0	38.0
94	River Park A	No	5.9	0.21	10.2	1.8	35.0	20.0
95	River Park C	No	5.9	0.21	19.3	4.3	83.0	45.0
96	Wildlife B	Yes	5.9	0.26	15.7	4.6	87.0	54.0
	1982 Urakawa-Oki							
97	Tokachi	No	6.9	0.17	16.2	2.4	42.0	35.0
	1983 Nihonkai-Chubu							
98	Arayamotomachi	No	6.8	0.15	4.6	4.3	69.0	37.0
99	Arayamotomachi Coarse Sand	No	6.8	0.15	18.1	9.2	158.0	77.0
100	Takeda Elementary School	Yes	6.8	0.11	12.1	4.3	81.0	42.0
	1983 Nihonkai-Chubu							
101	Aomori Station	Yes	7.7	0.12	16.5	5.7	95.0	38.0
102	Arayamotomachi	Yes	7.7	0.2	4.6	4.3	69.0	37.0
103	Gaiko Wharf B-2	Yes	7.7	0.23	12.4	7.5	123.0	53.0
104	Noshiro Section N-7	Yes	7.7	0.25	16.2	3.5	55.0	38.0
105	Takeda Elementary School	Yes	7.7	0.28	12.1	4.3	81.0	42.0
106	Akita station (1)	No	7.7	0.2	18.7	2.9	52.0	41.0
107	Akita station (2)	No	7.7	0.2	13.8	2.9	53.0	41.0
108	Aomori Port	No	7.7	0.12	13.1	3.3	63.0	41.0
109	Gaiko 1	Yes	7.7	0.2	8.8	6.9	132.0	79.0
110	Gaiko 2	Yes	7.7	0.2	7.0	9.8	189.0	107.0
111	Hakodate	No	7.7	0.05	9.4	4.3	81.0	54.0
112	Nakajima No. 1 (5)	Yes	7.7	0.2	10.4	6.5	124.0	74.0
113	Nakajima No. 2 (1)	Yes	7.7	0.2	13.5	7.1	136.0	81.0

No.	**Earthquake**/Site	Liq?	M_w	PGA	$N_{1,60cs}$	z	σ_v	σ'_v
114	Nakajima No. 2 (2)	Yes	7.7	0.2	9.5	3.8	71.0	48.0
115	Nakajima No. 3(3)	Yes	7.7	0.2	10.2	6.0	115.0	71.0
116	Nakajima No. 3(4)	Yes	7.7	0.2	11.4	5.7	109.0	68.0
117	Ohama No. 1(1)	No	7.7	0.2	19.0	3.9	74.0	47.0
118	Ohama No. 1(2)	No	7.7	0.2	23.6	3.4	64.0	42.0
119	Ohama No. 1(3)	No	7.7	0.2	23.0	2.6	48.0	34.0
120	Ohama No. 1(4)	No	7.7	0.2	34.5	5.2	99.0	60.0
121	Ohama No. 1(5)	No	7.7	0.2	35.5	2.2	41.0	31.0
122	Ohama No. 1(58- 22)	No	7.7	0.2	18.4	4.5	85.0	53.0
123	Ohama No. 2 (2)	Yes	7.7	0.2	5.5	5.2	100.0	56.0
124	Ohama No. 3 (1)	Yes	7.7	0.2	7.3	5.4	103.0	63.0
125	Ohama No. 3 (3)	Yes	7.7	0.2	5.6	5.5	104.0	64.0
126	Ohama No. 3 (4)	Yes	7.7	0.2	8.1	3.9	73.0	49.0
127	Ohama No. Rvt (1)	No	7.7	0.2	23.5	4.5	86.0	55.0
128	Ohama No. Rvt (2)	No	7.7	0.2	22.5	6.7	127.0	77.0
129	Ohama No. Rvt (3)	No	7.7	0.2	25.9	3.6	67.0	46.0
	1984							
130	Hososhima	No	6.9	0.27	17.6	2.9	53.0	45.0
	1987 Superstition Hills							
131	Radio Tower B1	No	6.2	0.09	8.4	3.4	62.0	50.0
132	Wildlife B	No	6.2	0.13	15.7	4.6	87.0	54.0
133	Heber Road A1	No	6.5	0.16	40.0	2.9	53.0	42.0
134	Heber Road A2	No	6.5	0.15	7.0	3.7	68.0	50.0
135	Heber Road A3	No	6.5	0.13	21.2	4.0	79.0	56.0
136	Kornbloom B	No	6.5	0.17	11.7	4.3	77.0	62.0
137	McKim Ranch A	No	6.5	0.16	10.0	2.1	38.0	32.0
138	Radio Tower B1	No	6.5	0.2	8.4	3.4	62.0	50.0
139	Radio Tower B2	No	6.5	0.18	20.6	2.3	40.0	38.0
140	River Park A	No	6.5	0.19	10.2	1.8	35.0	20.0
141	River Park C	No	6.5	0.19	19.3	4.3	83.0	45.0
142	Wildlife B	Yes	6.5	0.21	15.7	4.6	87.0	54.0

No.	**Earthquake**/Site	Liq?	M_w	PGA	$N_{1,60cs}$	z	σ_v	σ'_v
	1989 Loma Prieta							
143	Alameda Bay Farm Dike	No	6.9	0.24	43.4	6.5	125.0	91.0
144	Farris Farm	Yes	6.9	0.37	10.5	6.0	106.0	92.0
145	General Fish	No	6.9	0.28	22.4	2.5	45.0	35.0
146	Hall Avenue	No	6.9	0.14	11.0	4.6	75.0	64.0
147	Marine Laboratory B1	Yes	6.9	0.28	13.1	4.6	87.0	65.0
148	Marine Laboratory B2	Yes	6.9	0.28	14.9	3.5	65.0	55.0
149	Marine Laboratory UCB-6-12 & F1- F6	Yes	6.9	0.28	17.6	5.3	102.0	64.0
150	MBARI No. 3: EB-1	No	6.9	0.28	22.5	2.0	35.0	35.0
151	MBARI No. 3: EB-5	No	6.9	0.28	14.8	3.4	63.0	47.0
152	MBARI No. 4 (B4/B5/EB2/EB3)	No	6.9	0.28	21.1	3.4	62.0	48.0
153	MBARI Technology	No	6.9	0.28	14.7	3.4	62.0	48.0
154	Miller Farm CMF 3	Yes	6.9	0.39	15.3	6.2	114.0	101.0
155	Miller Farm CMF 5	Yes	6.9	0.39	23.4	7.0	130.0	108.0
156	Miller Farm CMF 8	Yes	6.9	0.39	14.8	6.0	111.0	95.0
157	Miller Farm CMF10	No	6.9	0.39	24.6	8.4	158.0	105.0
158	POO7-2	Yes	6.9	0.28	14.7	6.3	121.0	89.0
159	POO7-3	Marginal	6.9	0.28	16.3	6.3	121.0	89.0
160	POR-2&3&4	Yes	6.9	0.18	10.5	5.9	97.0	73.0
161	Sandholdt UC-B10	Yes	6.9	0.28	15.3	3.0	55.0	43.0
162	Sandholdt UC-B10	No	6.9	0.28	34.4	6.1	115.0	73.0
163	SFOBB-1&2	Yes	6.9	0.27	9.0	6.3	118.0	86.0
164	State Beach UC-B1	Yes	6.9	0.28	10.2	3.4	61.0	46.0
165	State Beach UC-B2	Yes	6.9	0.28	18.3	4.9	90.0	67.0
166	Treasure Island	Yes	6.9	0.16	10.8	6.5	116.0	67.0
167	WoodMarine UC-B4	Yes	6.9	0.28	14.6	1.8	32.0	25.0
	1990 Luzon							
168	Cereenan St. B-12	No	7.7	0.25	29.1	5.0	94.0	68.0
169	Perez B1v. B-11	Yes	7.7	0.25	17.3	7.2	139.0	90.0
	1993 Kushiro-							

No.	**Earthquake**/Site	Liq?	M_w	PGA	$N_{1,60cs}$	z	σ_v	σ'_v
	Oki							
170	Kushiro Port Quay Wall Site A	Yes	7.6	0.4	16.4	5.2	100.0	68.0
171	Kushiro Port Quay Wall Site D	No	7.6	0.4	30.9	10.8	208.0	118.0
172	Kushiro Port Seismo St.	Yes	7.6	0.47	26.0	3.8	65.0	47.0
	1994 Northridge							
173	Balboa B1v. Unit C	Yes	6.7	0.84	18.8	8.5	156.0	143.0
174	Malden Street Unit D	No	6.7	0.51	32.3	9.3	154.0	101.0
175	Potrero Canyon C1	Yes	6.7	0.43	14.1	7.1	139.0	88.0
176	Wynne Ave. Unit C1	Yes	6.7	0.51	17.1	6.7	129.0	105.0
	1995 Hyogoken-Nambu (Kobe)							
177	1	No	6.9	0.4	51.9	5.8	113.0	80.0
178	2	No	6.9	0.4	42.7	8.0	152.0	103.0
179	3	No	6.9	0.4	49.8	5.8	109.0	77.0
180	4	No	6.9	0.4	33.1	4.3	76.0	54.0
181	5	Yes	6.9	0.35	6.1	8.9	173.0	116.0
182	6	No	6.9	0.4	22.5	5.9	107.0	72.0
183	7	Yes	6.9	0.4	10.9	3.3	62.0	60.0
184	8	Yes	6.9	0.5	24.1	5.0	85.0	65.0
185	9	Yes	6.9	0.5	11.0	4.3	79.0	64.0
186	10	No	6.9	0.6	28.0	7.5	137.0	107.0
187	11	Yes	6.9	0.5	8.5	6.8	114.0	62.0
188	12	No	6.9	0.5	27.4	5.3	92.0	72.0
189	13	Yes	6.9	0.5	16.0	6.5	116.0	74.0
190	14	No	6.9	0.5	24.6	4.8	86.0	69.0
191	15	Yes	6.9	0.5	19.2	5.7	102.0	82.0
192	16	No	6.9	0.6	25.0	4.5	80.0	60.0
193	17	Yes	6.9	0.5	21.1	4.5	80.0	43.0
194	18	No	6.9	0.7	42.6	10.5	199.0	171.0
195	19	No	6.9	0.6	22.4	7.5	137.0	124.0

No.	**Earthquake**/Site	Liq?	M_w	PGA	$N_{1,60cs}$	z	σ_v	σ'_v
196	20	No	6.9	0.55	63.7	6.0	114.0	75.0
197	21	No	6.9	0.6	33.5	3.5	62.0	44.0
198	22	No	6.9	0.6	38.5	6.0	114.0	79.0
199	23	No	6.9	0.6	25.2	5.0	92.0	72.0
200	24	Yes	6.9	0.5	24.6	3.5	63.0	51.0
201	25	No	6.9	0.7	35.8	3.5	64.0	50.0
202	26	No	6.9	0.6	37.1	3.5	63.0	37.0
203	27	No	6.9	0.6	42.0	2.5	43.0	29.0
204	28	Yes	6.9	0.4	21.5	3.5	62.0	44.0
205	29	Yes	6.9	0.4	17.9	3.8	67.0	49.0
206	30	No	6.9	0.6	41.3	8.5	146.0	78.0
207	31	No	6.9	0.6	44.7	4.0	73.0	46.0
208	32	No	6.9	0.5	29.1	3.5	61.0	41.0
209	33	No	6.9	0.5	32.1	8.0	142.0	83.0
210	34	Yes	6.9	0.4	24.9	7.0	124.0	73.0
211	35	Yes	6.9	0.5	18.8	4.5	79.0	55.0
212	36	No	6.9	0.6	31.6	3.5	61.0	36.0
213	37	Yes	6.9	0.35	19.4	5.0	89.0	79.0
214	38	Yes	6.9	0.5	18.1	8.0	143.0	94.0
215	39	No	6.9	0.6	60.9	4.5	84.0	66.0
216	40	No	6.9	0.6	39.6	3.5	66.0	59.0
217	41	Yes	6.9	0.4	13.6	4.1	71.0	50.0
218	42	Yes	6.9	0.4	13.2	5.0	84.0	46.0
219	43	Yes	6.9	0.35	19.7	4.7	80.0	55.0
220	44	Yes	6.9	0.4	7.5	4.0	67.0	43.0
221	Ashiyama A (Mntn Sand 1)	No	6.9	0.4	25.2	5.2	97.0	80.0
222	Ashiyama C-D-E (Marine Sand)	Yes	6.9	0.4	12.4	8.8	166.0	115.0
223	Port Island Borehole Array Station	Yes	6.9	0.34	11.3	7.8	149.0	96.0
224	Port Island Improved Site (Ikegaya)	No	6.9	0.4	27.2	8.5	159.0	125.0
225	Port Island Improved Site (Tanahashi)	No	6.9	0.4	23.9	10.0	189.0	140.0
226	Port Island Improved	No	6.9	0.4	39.0	9.5	179.0	135.0

No.	**Earthquake**/Site	Liq?	M_w	PGA	$N_{1,60cs}$	z	σ_v	σ'_v
	Site (Watanabe)							
227	Port Island Site I	Yes	6.9	0.34	15.2	10.0	192.0	123.0
228	Rokko Island	Yes	6.9	0.4	21.8	7.5	141.0	107.0
	Building D							
229	Rokko Island Site G	Yes	6.9	0.34	16.8	11.5	219.0	146.0
230	Torishima Dike	Yes	6.9	0.25	18.5	4.7	93.0	46.0

Table 6.11: SPT Case History Database-Calculations

No.	n_{eq}	r_d	τ_{avg}	ΔW	$\frac{\Delta W}{\sigma'_v}$	G_{max}	$\frac{G_\gamma}{G_{max}}$	D_γ	$P_{liq.}$
			kPa	kPa		MPa			
1	21.3	0.91	11.57	0.048	0.00071	61.8	0.52	0.09	0.89
2	21.3	0.92	9.6	0.092	0.0015	56.8	0.31	0.13	0.98
3	21.3	0.93	8.37	0.22	0.0056	42.9	0.17	0.17	1.00
4	12.1	0.89	17.36	0.27	0.0057	55.4	0.24	0.16	0.99
5	12.8	0.81	25.89	0.17	0.0016	95.8	0.39	0.12	0.19
6	26.0	0.93	3.42	0.0032	7.8e-05	38.9	0.72	0.05	0.73
7	20.5	0.85	11.72	0.042	0.00059	64.8	0.53	0.08	0.82
8	20.5	0.89	7.84	0.02	0.00046	54.0	0.57	0.08	0.51
9	20.4	0.92	6.85	0.013	0.00025	49.8	0.61	0.07	0.84
10	19.6	0.85	13.19	0.029	0.00036	86.2	0.60	0.07	0.01
11	19.6	0.8	17.77	0.05	0.00046	100.8	0.57	0.07	0.01
12	20.5	0.8	15.79	0.067	0.00067	78.5	0.52	0.08	0.78
13	20.5	0.8	15.79	0.048	0.00048	89.4	0.57	0.08	0.16
14	20.5	0.9	8.05	0.023	0.00049	51.6	0.55	0.08	0.82
15	19.6	0.87	11.72	0.027	0.00034	74.8	0.59	0.07	0.33
16	20.5	0.91	7.54	0.03	0.00077	43.9	0.50	0.09	0.96
17	19.6	0.87	11.72	0.02	0.0003	87.7	0.62	0.07	0.00
18	16.9	0.93	7.78	0.039	0.00087	60.0	0.35	0.13	0.30
19	21.8	0.91	11.94	0.11	0.0028	54.3	0.39	0.12	0.73
20	21.1	0.9	15.5	0.048	0.00063	93.3	0.56	0.08	0.00
21	21.1	0.93	10.47	0.037	0.00081	64.5	0.52	0.08	0.03
22	21.1	0.93	10.47	0.083	0.002	48.2	0.40	0.11	0.98
23	22.4	0.93	9.1	0.13	0.003	54.1	0.24	0.15	0.95
24	10.4	0.81	26.65	4.1	0.043	73.8	0.03	0.23	1.00

No.	n_{eq}	r_d	τ_{avg}	ΔW	$\frac{\Delta W}{\sigma'_v}$	G_{max}	$\frac{G_\gamma}{G_{max}}$	D_γ	$P_{liq.}$
25	10.4	0.81	26.65	1.3	0.013	81.8	0.09	0.21	0.99
26	16.1	0.79	15.95	0.083	0.00093	78.1	0.42	0.11	0.70
27	16.1	0.79	16.26	0.076	0.00083	81.5	0.44	0.10	0.55
28	13.6	0.8	22.94	0.57	0.0067	84.2	0.16	0.17	0.77
29	13.6	0.79	24.39	0.32	0.0034	75.4	0.30	0.14	0.98
30	21.4	0.79	9.6	0.026	0.00031	57.8	0.58	0.07	0.94
31	21.4	0.9	4.33	0.0044	0.00013	44.2	0.69	0.05	0.43
32	21.4	0.78	9.4	0.016	0.00023	71.3	0.64	0.06	0.21
33	22.3	0.9	6.63	0.022	0.0004	61.5	0.46	0.10	0.40
34	18.7	0.9	9.52	0.079	0.0015	60.6	0.31	0.14	0.80
35	18.0	0.92	8.18	0.016	0.00043	60.1	0.58	0.07	0.00
36	18.0	0.92	7.39	0.029	0.0009	41.3	0.49	0.09	0.94
37	14.9	0.89	20.57	1.1	0.019	75.1	0.09	0.19	0.59
38	12.9	0.89	28.24	0.37	0.0049	91.0	0.28	0.15	0.00
39	18.7	0.87	13.36	0.16	0.0024	70.1	0.27	0.14	0.81
40	18.3	0.83	15.25	0.1	0.00099	80.4	0.37	0.12	0.87
41	18.3	0.78	20.05	0.086	0.00055	88.3	0.52	0.09	0.92
42	18.3	0.92	7.5	0.016	0.00042	52.8	0.57	0.08	0.37
43	18.3	0.93	6.4	0.0081	0.00018	55.3	0.64	0.06	0.21
44	18.3	0.88	10.35	0.048	0.00085	59.5	0.44	0.10	0.81
45	18.8	0.8	6.27	0.0044	6.5e-05	67.1	0.73	0.05	0.08
46	16.4	0.83	7.41	0.012	0.00018	72.9	0.53	0.09	0.10
47	17.4	0.88	4.54	0.0038	8.4e-05	45.6	0.69	0.05	0.57
48	16.4	0.88	4.99	0.0023	3.8e-05	62.3	0.76	0.04	0.04
49	16.4	0.81	8.43	0.0074	9.4e-05	74.8	0.69	0.05	0.08
50	17.4	0.9	3.74	0.0032	0.00011	37.4	0.68	0.05	0.63
51	17.4	0.88	4.35	0.0029	6.8e-05	49.0	0.72	0.05	0.25
52	16.4	0.8	7.69	0.0063	7.4e-05	69.3	0.70	0.05	0.29
53	16.4	0.87	5.91	0.0047	7.8e-05	56.7	0.70	0.05	0.37
54	17.4	0.86	5.35	0.005	9e-05	49.3	0.68	0.05	0.65
55	17.4	0.91	3.28	0.0011	3.3e-05	51.1	0.78	0.04	0.00
56	17.4	0.86	5.35	0.014	0.00034	53.1	0.45	0.11	0.44
57	17.4	0.82	6.35	0.0069	0.00011	57.1	0.66	0.06	0.52
58	17.4	0.86	5.35	0.0039	8.2e-05	57.3	0.71	0.05	0.07
59	19.2	0.87	13.69	0.067	0.001	67.1	0.48	0.10	0.75
60	17.8	0.89	13.65	0.14	0.002	72.9	0.29	0.14	0.71

No.	n_{eq}	r_d	τ_{avg}	ΔW	$\frac{\Delta W}{\sigma'_v}$	G_{max}	$\frac{G_\gamma}{G_{max}}$	D_γ	$P_{liq.}$
61	19.2	0.93	7.94	0.029	0.00065	45.6	0.51	0.09	0.96
62	19.2	0.91	10.25	0.021	0.00037	71.7	0.59	0.07	0.01
63	16.7	0.93	10.47	0.028	0.00047	62.3	0.54	0.08	0.58
64	16.7	0.9	14.75	0.053	0.00073	76.5	0.50	0.09	0.25
65	17.8	0.9	12.13	0.097	0.0015	72.4	0.31	0.13	0.43
66	17.8	0.88	15.71	0.073	0.00092	74.8	0.48	0.09	0.63
67	15.8	0.93	12.16	0.058	0.0015	62.1	0.43	0.11	0.02
68	15.8	0.94	10.36	0.14	0.0048	37.4	0.28	0.14	1.00
69	15.8	0.93	12.16	0.11	0.0027	49.0	0.34	0.13	0.99
70	17.8	0.87	14.39	0.062	0.00072	69.3	0.50	0.09	0.89
71	17.8	0.92	10.72	0.041	0.00068	56.7	0.50	0.09	0.91
72	17.8	0.91	11.36	0.078	0.0014	49.3	0.41	0.11	0.99
73	17.8	0.95	6.79	0.012	0.00036	51.1	0.59	0.07	0.18
74	17.8	0.91	11.36	0.37	0.0089	53.1	0.13	0.18	0.99
75	17.8	0.85	18.39	0.28	0.0036	85.8	0.24	0.15	0.18
76	17.8	0.89	13.7	0.21	0.0033	57.1	0.26	0.15	1.00
77	17.8	0.91	11.36	0.056	0.0012	57.3	0.45	0.10	0.73
78	17.8	0.89	14.25	0.043	0.00061	81.9	0.54	0.08	0.01
79	8.1	0.9	24.24	3	0.072	71.5	0.03	0.21	0.01
80	8.1	0.88	30.24	5.8	0.12	48.7	0.03	0.21	1.00
81	8.1	0.87	34.77	5.4	0.096	70.4	0.03	0.21	0.99
82	17.0	0.86	5.59	0.0035	5.7e-05	63.0	0.72	0.05	0.10
83	9.7	0.93	11.69	0.67	0.021	43.3	0.06	0.21	1.00
84	14.2	0.89	7.14	0.013	0.00026	51.5	0.54	0.08	0.74
85	14.2	0.92	4.79	0.0069	0.00018	57.5	0.50	0.10	0.01
86	13.2	0.94	5.12	0.013	0.00063	34.4	0.47	0.09	0.82
87	15.2	0.85	8.17	0.03	0.00056	63.7	0.39	0.12	0.33
88	16.9	0.76	5.08	0.0072	0.00013	57.1	0.55	0.08	0.43
89	16.9	0.54	8.53	0.0098	8e-05	82.6	0.62	0.07	0.33
90	10.0	0.82	13.06	0.06	0.00096	63.0	0.35	0.12	0.82
91	16.8	0.9	2.01	0.00083	2.6e-05	43.3	0.69	0.06	0.07
92	12.1	0.85	6.86	0.0095	0.00019	51.5	0.56	0.08	0.65
93	12.1	0.9	4.66	0.0053	0.00014	57.5	0.51	0.09	0.00
94	11.9	0.92	4.38	0.006	0.0003	34.4	0.55	0.08	0.64
95	11.9	0.82	9.24	0.054	0.0012	61.5	0.28	0.15	0.25
96	10.9	0.8	11.83	0.1	0.0019	63.7	0.23	0.16	0.69

No.	n_{eq}	r_d	τ_{avg}	ΔW	$\frac{\Delta W}{\sigma'_v}$	G_{max}	$\frac{G_\gamma}{G_{max}}$	D_γ	$P_{liq.}$
97	16.8	0.93	4.26	0.0024	6.8e-05	51.7	0.73	0.05	0.01
98	17.2	0.87	5.87	0.016	0.00042	37.0	0.54	0.08	0.97
99	17.2	0.75	11.61	0.021	0.00027	79.1	0.61	0.07	0.06
100	19.4	0.87	5.1	0.0047	0.00011	52.3	0.69	0.05	0.20
101	24.0	0.88	6.33	0.011	0.00029	54.3	0.63	0.06	0.13
102	19.2	0.91	8.16	0.062	0.0017	37.0	0.40	0.11	1.00
103	18.2	0.85	15.44	0.17	0.0033	59.2	0.34	0.13	0.95
104	17.5	0.93	8.27	0.021	0.00056	53.9	0.54	0.08	0.29
105	16.7	0.91	13.56	0.16	0.0037	52.3	0.32	0.13	0.97
106	19.0	0.94	6.5	0.0082	0.0002	58.3	0.65	0.06	0.03
107	19.0	0.94	6.62	0.01	0.00026	53.7	0.62	0.07	0.27
108	24.0	0.93	4.41	0.0035	8.6e-05	52.9	0.73	0.05	0.10
109	19.0	0.86	15.14	0.097	0.0012	65.5	0.44	0.10	0.96
110	19.0	0.81	20.43	0.22	0.002	71.2	0.38	0.12	0.99
111	33.4	0.91	2.49	0.00078	1.4e-05	55.2	0.85	0.03	0.03
112	19.0	0.87	14.36	0.078	0.0011	66.5	0.46	0.10	0.90
113	19.0	0.86	15.53	0.073	0.0009	74.9	0.49	0.09	0.67
114	19.0	0.92	8.71	0.028	0.00057	52.2	0.53	0.08	0.84
115	19.0	0.88	13.44	0.066	0.00093	64.9	0.48	0.09	0.89
116	19.0	0.88	12.82	0.055	0.00081	65.5	0.50	0.09	0.80
117	19.0	0.92	9.05	0.02	0.00043	62.7	0.57	0.08	0.08
118	19.0	0.93	7.91	0.013	0.00032	62.7	0.61	0.07	0.00
119	19.0	0.94	6.04	0.0071	0.00021	56.0	0.65	0.06	0.00
120	19.0	0.89	11.78	0.024	0.00041	82.4	0.59	0.07	0.00
121	19.0	0.95	5.2	0.0038	0.00012	59.7	0.71	0.05	0.00
122	19.0	0.91	10.27	0.027	0.00051	65.9	0.56	0.08	0.13
123	19.0	0.89	11.89	0.11	0.002	48.0	0.38	0.12	1.00
124	19.0	0.89	12.2	0.074	0.0012	55.4	0.44	0.10	0.98
125	19.0	0.89	12.3	0.093	0.0014	51.8	0.41	0.11	1.00
126	19.0	0.92	8.93	0.033	0.00067	50.4	0.51	0.09	0.93
127	19.0	0.91	10.37	0.022	0.00041	71.7	0.58	0.07	0.01
128	19.0	0.87	14.64	0.045	0.00059	83.8	0.54	0.08	0.02
129	19.0	0.92	8.25	0.013	0.00028	67.2	0.62	0.07	0.00
130	13.9	0.92	8.45	0.046	0.001	60.0	0.31	0.14	0.34
131	18.3	0.87	3.16	0.001	2e-05	51.5	0.80	0.04	0.09
132	15.6	0.83	6.24	0.011	0.00021	63.7	0.50	0.09	0.12

No.	n_{eq}	r_d	τ_{avg}	ΔW	$\frac{\Delta W}{\sigma'_v}$	G_{max}	$\frac{G_\gamma}{G_{max}}$	D_γ	$P_{liq.}$
133	15.8	0.9	4.85	0.0049	0.00012	71.5	0.56	0.08	0.00
134	16.1	0.88	5.82	0.018	0.00037	48.7	0.42	0.11	0.89
135	17.0	0.87	5.8	0.0075	0.00013	70.4	0.56	0.08	0.00
136	15.1	0.86	7.49	0.0086	0.00014	63.0	0.63	0.06	0.27
137	15.7	0.93	3.67	0.0057	0.00018	43.3	0.51	0.09	0.49
138	14.3	0.89	7.15	0.013	0.00026	51.5	0.54	0.08	0.74
139	14.9	0.92	4.31	0.005	0.00013	57.5	0.54	0.09	0.00
140	14.6	0.94	4.05	0.0054	0.00027	34.4	0.58	0.07	0.61
141	14.6	0.86	8.81	0.053	0.0012	61.5	0.30	0.14	0.25
142	14.1	0.85	9.91	0.06	0.0011	63.7	0.31	0.14	0.54
143	14.6	0.83	16.09	0.026	0.00028	107.2	0.60	0.07	0.00
144	12.3	0.84	21.34	0.16	0.0018	74.4	0.36	0.12	0.95
145	13.7	0.93	7.6	0.011	0.00032	56.5	0.58	0.07	0.01
146	18.3	0.87	5.95	0.0098	0.00015	62.9	0.55	0.08	0.36
147	13.7	0.87	13.8	0.052	0.00081	66.5	0.47	0.10	0.67
148	13.7	0.9	10.65	0.024	0.00044	63.4	0.53	0.08	0.33
149	13.7	0.85	15.86	0.075	0.0012	71.6	0.43	0.11	0.38
150	13.7	0.94	6.0	0.005	0.00014	56.5	0.66	0.06	0.00
151	13.7	0.9	10.35	0.029	0.00062	58.6	0.50	0.09	0.43
152	13.7	0.9	10.19	0.021	0.00043	65.1	0.55	0.08	0.03
153	13.7	0.9	10.19	0.026	0.00055	59.0	0.51	0.09	0.41
154	12.0	0.83	24.05	0.43	0.0043	86.6	0.19	0.16	0.89
155	12.0	0.81	26.81	0.41	0.0038	100.3	0.21	0.16	0.23
156	12.0	0.84	23.56	0.49	0.0052	83.2	0.17	0.17	0.93
157	12.0	0.78	31.32	0.94	0.0089	100.2	0.14	0.18	0.36
158	13.7	0.83	18.28	0.079	0.00088	80.5	0.46	0.10	0.55
159	13.7	0.83	18.28	0.073	0.00082	82.8	0.47	0.10	0.38
160	16.5	0.84	9.53	0.018	0.00025	66.3	0.57	0.07	0.55
161	13.7	0.91	9.15	0.021	0.00049	56.5	0.53	0.09	0.33
162	13.7	0.83	17.47	0.054	0.00074	90.9	0.50	0.09	0.00
163	14.0	0.83	17.19	0.099	0.0012	68.8	0.42	0.11	0.95
164	13.7	0.9	10.03	0.035	0.00076	52.2	0.47	0.10	0.86
165	13.7	0.86	14.16	0.043	0.00064	74.1	0.50	0.09	0.17
166	17.3	0.83	9.96	0.057	0.00084	64.0	0.36	0.12	0.84
167	13.7	0.95	5.52	0.025	0.001	42.6	0.33	0.13	0.60
168	17.5	0.9	13.69	0.1	0.0015	84.2	0.32	0.13	0.00

No.	n_{eq}	r_d	τ_{avg}	ΔW	$\frac{\Delta W}{\sigma'_v}$	G_{max}	$\frac{G_\gamma}{G_{max}}$	D_γ	$P_{liq.}$
169	17.5	0.86	19.33	0.29	0.0032	84.5	0.25	0.15	0.71
170	14.1	0.89	23.1	0.37	0.0054	72.4	0.26	0.15	0.88
171	14.1	0.79	42.6	0.97	0.0082	112.6	0.23	0.16	0.03
172	13.2	0.92	18.2	0.19	0.004	68.0	0.30	0.14	0.10
173	8.2	0.76	64.71	12	0.083	108.9	0.04	0.23	1.00
174	10.1	0.74	37.84	2.1	0.021	105.3	0.08	0.20	0.07
175	10.8	0.79	30.83	5.5	0.063	79.0	0.03	0.23	1.00
176	10.1	0.8	34.37	1.9	0.018	91.0	0.08	0.19	0.97
177	11.8	0.84	24.68	0.12	0.0015	104.7	0.41	0.11	0.00
178	11.8	0.79	31.16	0.64	0.0062	113.6	0.17	0.17	0.00
179	11.8	0.84	23.81	0.11	0.0015	101.8	0.41	0.11	0.00
180	11.8	0.88	17.35	0.083	0.0015	77.4	0.40	0.12	0.00
181	12.4	0.77	30.25	0.88	0.0076	71.2	0.19	0.17	1.00
182	11.8	0.84	23.31	0.87	0.012	81.1	0.11	0.19	0.64
183	11.8	0.9	14.58	0.076	0.0013	60.8	0.40	0.12	0.90
184	10.8	0.86	23.76	0.25	0.0039	78.4	0.28	0.15	0.19
185	10.8	0.88	22.54	0.45	0.0071	62.8	0.20	0.17	1.00
186	10.0	0.8	42.74	1	0.0097	104.6	0.18	0.17	0.14
187	10.8	0.82	30.24	4.5	0.073	57.5	0.05	0.22	1.00
188	10.8	0.85	25.49	1.1	0.015	85.3	0.09	0.19	0.27
189	10.8	0.82	31.04	4.6	0.062	75.1	0.04	0.21	1.00
190	10.8	0.87	24.18	1.1	0.016	81.2	0.09	0.19	0.53
191	10.8	0.84	27.93	0.38	0.0046	83.0	0.26	0.15	0.64
192	10.0	0.87	27.23	0.58	0.0097	76.0	0.18	0.17	0.35
193	10.8	0.87	22.69	0.79	0.018	61.6	0.14	0.19	0.84
194	9.4	0.73	66.55	1.8	0.01	146.3	0.18	0.18	0.00
195	10.0	0.8	42.74	0.83	0.0067	106.3	0.21	0.16	0.47
196	10.3	0.84	34.04	0.45	0.006	106.1	0.24	0.16	0.00
197	10.0	0.9	21.74	0.33	0.0074	70.1	0.21	0.17	0.01
198	10.0	0.84	37.14	0.9	0.011	97.1	0.17	0.18	0.00
199	10.0	0.86	30.86	0.65	0.0091	83.5	0.19	0.17	0.32
200	10.8	0.9	18.41	0.14	0.0027	69.8	0.32	0.14	0.10
201	9.4	0.9	26.19	0.54	0.011	75.9	0.17	0.18	0.00
202	10.0	0.9	22.09	0.52	0.014	65.9	0.16	0.18	0.00
203	10.0	0.93	15.54	0.13	0.0043	60.1	0.29	0.14	0.00
204	11.8	0.9	14.5	0.083	0.0019	62.6	0.37	0.12	0.20

No.	n_{eq}	r_d	τ_{avg}	ΔW	$\frac{\Delta W}{\sigma'_v}$	G_{max}	$\frac{G_\gamma}{G_{max}}$	D_γ	$P_{liq.}$
205	11.8	0.89	15.52	0.1	0.0021	63.0	0.35	0.13	0.52
206	10.0	0.78	44.26	2.6	0.034	98.1	0.10	0.20	0.00
207	10.0	0.89	25.22	0.49	0.011	76.7	0.18	0.17	0.00
208	10.8	0.9	17.83	0.18	0.0043	65.4	0.27	0.15	0.02
209	10.8	0.79	36.39	6.7	0.08	95.2	0.03	0.23	0.35
210	11.8	0.81	26.16	0.32	0.0044	83.8	0.28	0.15	0.16
211	10.8	0.87	22.41	0.38	0.0069	67.6	0.22	0.16	0.78
212	10.0	0.9	21.39	0.54	0.015	62.5	0.16	0.18	0.06
213	12.4	0.86	17.41	0.061	0.00077	81.7	0.47	0.10	0.15
214	10.8	0.79	36.65	1.1	0.012	87.4	0.16	0.18	0.92
215	10.0	0.87	28.6	0.27	0.0041	98.6	0.28	0.15	0.00
216	10.0	0.9	23.15	0.18	0.003	84.5	0.31	0.14	0.00
217	11.8	0.88	16.31	0.15	0.0031	59.0	0.30	0.14	0.91
218	11.8	0.86	18.78	0.4	0.0087	56.1	0.20	0.17	0.99
219	12.4	0.87	15.79	0.36	0.0065	68.4	0.14	0.18	0.71
220	11.8	0.89	15.43	0.33	0.0078	46.1	0.19	0.17	1.00
221	11.8	0.86	21.56	0.36	0.0045	88.0	0.18	0.17	0.15
222	11.8	0.77	33.27	0.61	0.0053	87.3	0.24	0.16	0.98
223	12.6	0.79	26.11	1.1	0.012	77.5	0.11	0.18	1.00
224	11.8	0.78	32.14	0.53	0.0043	112.2	0.21	0.16	0.06
225	11.8	0.75	36.62	0.76	0.0054	114.9	0.19	0.16	0.28
226	11.8	0.76	35.17	0.5	0.0037	127.4	0.22	0.16	0.00
227	12.6	0.75	31.62	0.9	0.0073	95.5	0.16	0.17	0.95
228	11.8	0.8	29.32	0.68	0.0064	98.0	0.16	0.17	0.51
229	12.6	0.71	34.6	0.72	0.0049	106.8	0.20	0.16	0.84
230	14.3	0.87	13.11	0.32	0.0069	61.6	0.14	0.18	0.80

6.13.1 Sample Calculations

This section provides sample calculations for one of the case histories in the database above. The calculations will follow case history number 8: site *Cc17-2* of the 1964 M=7.6 Niigata earthquake - June 16. In this example, liquefaction did occur with a 7.6 magnitude earthquake producing a peak ground acceleration of 0.16g. The layer of interest was at a depth of 5.3m and had total and effective vertical overburden stresses of 85.0 kPa and 43.0

kPa, respectively.

Calculate Small-Strain Shear Modulus

In the absence of shear wave velocity measurements, the small-strain shear modulus can be estimated using the SPT blowcount (Seed et al. 1986[2]):

$$G_{max} = 440 \cdot N_{1,60cs}^{\frac{1}{3}} \cdot P_a \cdot \sqrt{\frac{\sigma'_{m0}}{P_a}} \tag{6.28}$$

where σ'_{m0} is :

$$\begin{aligned} \sigma'_{m0} &= (1 + 2K_o)\sigma'_{v0}/3 \\ &= (1 + 2 \cdot 0.44)43.0/3 \\ &= 26.9 \end{aligned} \tag{6.29}$$

and P_a is the atmospheric pressure (101.325 kPa). K_o is estimated from $N_{1,60cs}$ via ϕ' (Sabatini et al. 2002[3] after Hatanaka and Uchida 1996[4]):

$$\begin{aligned} \phi' &= \sqrt{15.4 \cdot N_{1,60cs}} + 20 \\ &= \sqrt{15.4 \cdot 13.0} + 20 \\ &= 34.1 \end{aligned} \tag{6.30}$$

[2]Seed, H. B., Wong, R. T., Idriss, I. M., and Tokimatsu, K. (1986). "Moduli and Damping Factors for Dynamic Analyses of Cohesionless Soils." Journal of Geotechnical Engineering, 112(11), 1016–1032.

[3]Sabatini, P. J., Bachus, R. C., Mayne, P. W., Schneider, J. A., and Zettler, T. E. (2002). Geotechnical engineering circular no. 5: Evaluation of Soil and Rock Properties. Federal Highway Administration, Washington, D.C., 385.

[4]Hatanaka, M., and Uchida, A. (1996). "A simple method for the determination of Ko-value in sandy soils." *Soils and Foundations*, 36(2), 93–99.

and (Jaky 1944[5], 1948[6]):

$$\begin{aligned} K_o &= 1 - \sin(\phi') \\ &= 1 - \sin(34.1) \\ &= 0.44 \end{aligned} \tag{6.31}$$

Thus:

$$\begin{aligned} G_{max} &= 440 \cdot 13.0^{\frac{1}{3}} \cdot 101.325 \cdot \sqrt{\frac{26.9}{101.325}} \\ &= 54002\text{kPa} \end{aligned} \tag{6.32}$$

Calculate Stress-Reduction Coefficient

The stress reduction coefficient is used to estimate the shear stress at depth using the peak ground acceleration at the surface of the profile. A stress reduction coefficient was developed as part of this research (see the Technical Note in Chapter 5):

$$r_d = (1 - \alpha) \exp\left(\frac{-z}{\beta}\right) + \alpha \tag{6.33}$$

where z is depth from the surface in meters, and

$$\alpha = \exp(a_1 + a_2 \cdot M_w) \tag{6.34}$$

$$\beta = a_3 + a_4 \cdot M_w \tag{6.35}$$

[5]Jaky, J. (1944). "The coefficient of earth pressure at rest." *Journal of the Society of Hungarian Architects and Engineers*, 78(22), 355–358.

[6]Jaky, J. (1948). "Influence of ground water level oscillation on subsidence of structures." *Proc. II. Int. Conf. Soil Mech.*, Rotterdam.

where $a_1 = -4.031$, $a_2 = 0.364$, $a_3 = -28.47$, and $a_4 = 7.798$. Thus:

$$\begin{aligned} \alpha &= \exp(-4.03 + 0.36 \cdot 7.6) \\ &= 0.282 \\ \beta &= -28.47 + 7.80 \cdot 7.6 \\ &= 30.795 \end{aligned} \tag{6.36}$$

$$\begin{aligned} r_d &= (1 - 0.28) \exp\left(\frac{-5.3}{30.79}\right) + 0.28 \\ &= 0.89 \end{aligned} \tag{6.37}$$

Calculate the Dissipated Energy in One Equivalent Cycle

For a liquefaction evaluation utilizing site response analyses, dissipated energy can be calculated directly using the stress and strain time histories from the depth of interest. For the simplified procedure, this dissipated energy must be estimated by multiplying the number of equivalent cycles by the dissipated energy in one equivalent cycle:

$$\Delta W = \Delta W_{1,eq} \cdot n_{eq} \tag{6.38}$$

The quantity $\Delta W_{1,eq}/G_{max}$ can be estimated graphically using Figure 6.20. This figure was created using Darendeli and Stokoe's (2001[7]) shear modulus and damping degradation curves and Tokimatus and Seed's (1987[8]) direct approach. We enter the figure from the x-axis using the following value:

$$\begin{aligned} 0.65 \cdot a_{max} \cdot \sigma_v \cdot r_d/(g \cdot G_{max}) &= 0.65 \cdot 0.2 \cdot 85.0 \cdot 0.89/(1 \cdot 54002) \\ &= 0.0001451 \end{aligned} \tag{6.39}$$

[7]Darendeli, M. B., and Stokoe, K. H. (2001). Development of a new family of normalized modulus reduction and material damping curves. Geotechnical Engineering Report GD01-1, University of Texas at Austin.

[8]Tokimatsu, K., and Seed, H. B. (1987). "Evaluation of settlements in sands due to earthquake shaking." Journal of Geotechnical Engineering, 113(8), 861–878.

From this point on the x-axis, we read up to the curve corresponding to our mean effective overburden pressure(σ'_{m0} = 26.9 kPa). From there, we read left to the y-axis to read off the value of $\Delta W_{1,eq}/G_{max}$ which is equal to 1.768e-08. Multiplying this value by G_{max}, we obtain our value of $\Delta W_{1,eq}$ which is 0.000955 kPa. The diamonds in the figure correspond to a threshold shear strain of 10^{-2} %. If we had plotted below the diamonds, we could assume that no liquefaction would occur. Note that a fifth-order fit to these curves is presented in the body of the paper.

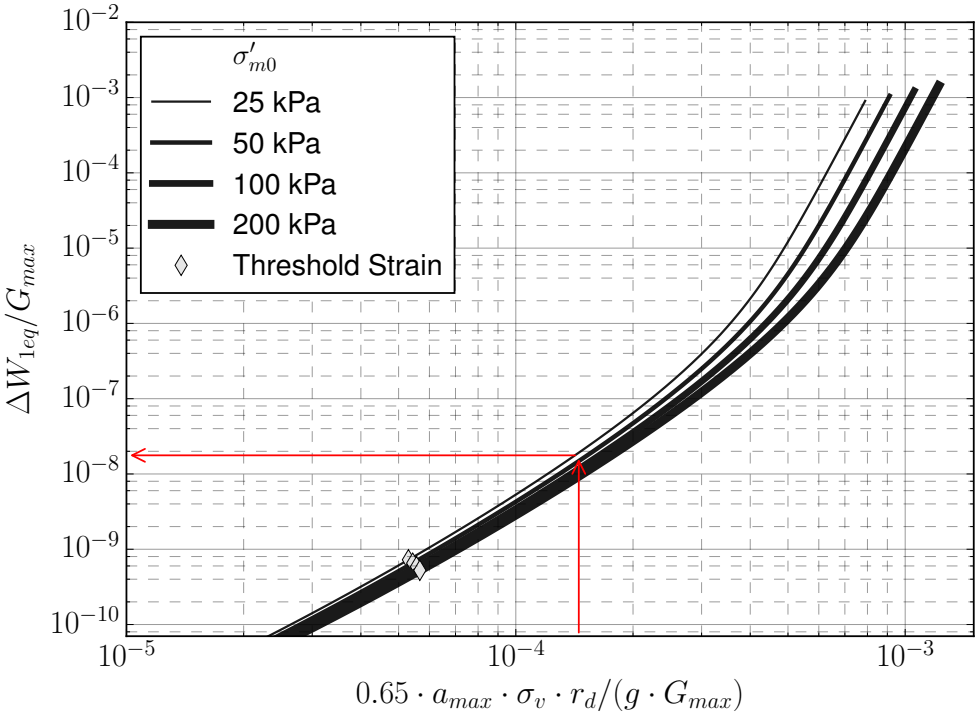


Figure 6.20: Graphical method to obtain $\Delta W_{1,eq}$ using Darendeli and Stokoe's shear modulus and damping degradation curves.

Calculate Number of Equivalent Cycles

The number of equivalent cycles can be calculated according to the following equation:

$$n_{eq} = 2 \cdot \exp\left(b_1 + b_2 \ln(a_{max}) + b_3 M_w\right) \tag{6.40}$$

where $b_1 = -0.3643$, $b_2 = -0.4105$, and $b_3 = 0.2553$. Thus:

$$\begin{aligned} n_{eq} &= 2 \cdot \exp\left(-0.3643 + -0.4105 \ln(0.16) + 0.2553 \cdot 7.6\right) \\ &= 20.5 \end{aligned} \tag{6.41}$$

Calculate Total Dissipated Energy

With the dissipated energy in one cycle and the number of equivalent cycles, we are now able to calculate the total dissipated energy, ΔW:

$$\begin{aligned} \Delta W &= \Delta W_{1,eq} \cdot n_{eq} \\ &= 0.000955 \cdot 20.5 \\ &= 0.0196 \text{ kPa} \end{aligned} \tag{6.42}$$

The normalized dissipated energy is easily computed:

$$\begin{aligned} \frac{\Delta W}{\sigma'_v} &= \frac{0.0196}{43.0} \\ &= 0.000456 \end{aligned} \tag{6.43}$$

Calculate the Limit-State Value, g

The limit-state function is defined as follows:

$$g = c_1 N_{1,60cs} - \ln\left(\frac{\Delta W}{\sigma'_{v0}}\right) + c_2 + \varepsilon \tag{6.44}$$

where $c_1 = 0.299$, $c_2 = -11.610$, and the standard deviation of ε, σ_ε, is 1.3. Thus:

$$\begin{aligned} g &= 0.30 \cdot 13.0 - \ln(0.000456) + -11.61 \\ &= -0.0329 \end{aligned} \tag{6.45}$$

Since g is less than zero, we expect liquefaction to occur.

Calculate Probability of Liquefaction

With a values of g and σ_ε, we can calculate the probability of liquefaction:

$$\begin{aligned} P[g \leq 0] &= 1 - \Phi\left[\frac{g}{\sigma_\varepsilon}\right] \\ &= 1 - \Phi\left[\frac{-0.0329}{1.29}\right] \\ &= 51\% \end{aligned} \tag{6.46}$$

Relatedly, we can define a normalized dissipated energy curve for a given probability of liquefaction:

$$\begin{aligned} \left(\frac{\Delta W}{\sigma'_{v0}}\right)_P &= \exp\left(c_1 N_{1,60cs} + c_2 + \Phi^{-1}[P] \cdot \sigma_\varepsilon\right) \\ &= \exp\left(0.299 \cdot N_{1,60cs} + -11.610 + \Phi^{-1}[P] \cdot 1.3\right) \end{aligned} \tag{6.47}$$

Calculate the Dissipated Energy to Liquefaction

For deterministic analyses, we can calculate the normalized dissipated energy required to cause liquefaction, $\Delta W_{liq}/\sigma'_{v0}$. We define the deterministic curve

as the probabilistic curve at a probability of 15%:

$$\begin{aligned}\left(\frac{\Delta W_{liq}}{\sigma'_{v0}}\right) &= \exp\left(c_1 N_{1,60cs} + c_2 - 1.036 \cdot \sigma_\varepsilon\right) \\ &= \exp\left(0.30 \cdot 13.0 + -11.61 - 1.036 \cdot 1.29\right) \\ &= 1.1561e-04\end{aligned} \tag{6.48}$$

Calculate the Factor of Safety

Finally, the factor of safety against liquefaction is calculated as:

$$\begin{aligned}FS_{liq} &= \frac{\ln\left(\frac{\Delta W_{liq}}{\sigma'_{v0}}\right) - c_{FS}}{\ln\left(\frac{\Delta W_{calculated}}{\sigma'_{v0}}\right) - c_{FS}} \\ &= \frac{\ln\left(0.000116\right) - \left(1.2 \cdot -11.61\right)}{\ln\left(0.000456\right) - \left(1.2 \cdot -11.61\right)} \\ &= 0.8\end{aligned} \tag{6.49}$$

because c_{FS} is defined as:

$$c_{FS} = 1.2 \cdot c_2 \tag{6.50}$$

6.13.2 Procedure to Evaluate Liquefaction Potential

This section gives the step-by-step procedure to evaluate liquefaction potential using the energy-based procedure proposed in the attached paper. Two variations are presented: a "simplified" procedure, and a "non-simplified" procedure. All units used in these calculation should be SI units, unless otherwise specified.

Simplified Procedure

The "simplified" procedure does not require site response analyses. Instead, it relies on the expected peak ground acceleration and a number of equivalent cycles (n_{eq}) correlation.

1. Obtain parameters to characterize the capacity of the liquefiable layer and the demand of the earthquake motion. These are:

- SPT blow count (N_m)
- Fines Content (FC)
- Depth (z)
- Total overburden stress (σ_v)
- Initial effective overburden stress (σ'_{v0})
- Various SPT details:
  - Energy Ratio (ER)
  - Borehole diameter
  - Rod length
  - Sampler information
- Peak ground acceleration, a_{max}
- Moment magnitude, M_w

2. Correct the measured SPT blow count (N_m) using the procedure of Idriss and Boulanger (2008):

   $N_{60} = C_E \cdot C_B \cdot C_R \cdot C_S \cdot N_m$

   where:

- C_E is the energy ratio correction factor: $C_E = \frac{ER}{60}$
  The following ranges have been given (Seed et al. 1984; Skempton 1986; NCEER 1997):
  - Doughnut Hammer: $0.5 \leq C_E \leq 1.0$
  - Safety Hammer: $0.7 \leq C_E \leq 1.2$
  - Automatic Triphammer: $0.8 \leq C_E \leq 1.3$
- C_B is the borehole diameter correction factor (Skempton 1986):
  - 65-115 mm: $C_B = 1.0$
  - 150 mm: $C_B = 1.05$
  - 200 mm: $C_B = 1.15$
- C_R is the rod length correction factor (Youd et al. 2001). Assume a rod stick-up length of 1.5 m so that rod length $= 1.5 + z$:
  - Rod length < 3 m: $C_R = 0.75$
  - Rod length 3-4 m: $C_R = 0.80$
  - Rod length 4-6 m: $C_R = 0.85$
  - Rod length 6-10 m: $C_R = 0.95$
  - Rod length 10-30 m: $C_R = 1.00$
- C_S is the sampler correction factor (Seed et al. 1984; Seed et al. 2001):
  - Standard split spoon without room for liners: $C_S = 1.0$
  - Split spoon with room for liners but without the liner present: $C_S = 1 + \frac{N_{1,60}}{100}; 1.1 \leq C_S \leq 1.3$

$N_{1,60} = C_N \cdot N_{60}$

where (Boulanger 2003):

$C_N = \left(\frac{P_a}{\sigma'_{v0}}\right)^{\alpha} \leq 1.7$

P_a is standard atmospheric pressure (1 atm = 101.325 kPa), and α is defined as:

$\alpha = 0.784 - 0.0768\sqrt{N_{1,60}}$

Finally, the SPT blow count with fines content corrections is:

$N_{1,60cs} = N_{1,60} + \Delta N_{1,60}$

where:

$\Delta N_{1,60} = \exp\left(1.63 + \frac{9.7}{FC+0.01} - \left[\frac{15.7}{FC+0.01}\right]^2\right)$

You will notice that both C_S and C_N are a function of $N_{1,60}$. Thus, you have to assume a value of $N_{1,60}$ to start and iterate until convergence is reached. If the corrected SPT blow count is greater than 30, liquefaction is not expected to occur.

3. Calculate the mean initial effective stress, σ'_{m0}:

   - Estimate the effective friction angle, ϕ' (Hatanaka and Uchida 1996; Sabatini et al. 2002):
     $\phi' = \sqrt{15.4 N_{1,60cs}} + 20$
   - Estimate the at-rest earth pressure coefficient, K_o (Jaky 1944; 1948):
     $K_o = 1 - \sin\phi'$
   - Calculate the mean initial effective stress:
     $\sigma'_{m0} = \frac{1+2\cdot K_o}{3} \cdot \sigma'_{v0}$
     σ'_{v0} should be limited to values above 40 kPa.

4. Estimate the small strain shear modulus, G_{max} (Seed et al. 1986; Ohta and Goto 1976):

   $G_{max} = 440 \cdot N_{1,60cs}^{\frac{1}{3}} \cdot P_a \cdot \sqrt{\frac{\sigma'_{m0}}{P_a}}$

   where P_a is standard atmospheric pressure (101.325 kPa).

5. Calculate the number of equivalent stress cycles, n_{eq} (Lasley et al. 2015a):

   $n_{eq} = \exp\left(b_1 + b_2 \ln(a_{max}) + b_3 M_w\right)$

   Table 6.12: Regression Coefficients for n_{eq} (Lasley et al. 2015a)

EQ Regime	b_1	b_2	b_3	b_4
Active Shallow-Crustal	0.806	-0.298	-8.98e-4	0.254
Stable Continental	0.557	-0.390	-1.80e-4	0.315

6. Calculate the depth reduction coefficient, r_d (Lasley et al. 2015b):

   $r_d = (1 - \alpha) \exp\left(\frac{-z}{\beta}\right) + \alpha$

   where z is in meters, and

   $\alpha = \exp(a_1 + a_2 \cdot M_w)$

   $\beta = a_3 + a_4 \cdot M_w$

   Table 6.13: Regression Coefficients for r_d (Lasley et al. 2015b)

EQ Regime	a_1	a_2	a_3	a_4	σ
Active Shallow-Crustal	-4.031	0.364	-28.47	7.798	0.1397
Stable Continental	-4.369	0.439	-15.38	4.922	0.1586

7. Estimate the dissipated energy in one equivalent cycle:

   - Calculate Γ:
     $\Gamma = \log_{10}\left(0.65 \cdot a_{max} \cdot \sigma_v \cdot r_d/(g \cdot G_{max})\right)$; $\sim -5 \leq \Gamma \leq \sim -3$
   - Calculate $\Delta W_{1,eq}/G_{max}$:
     $\frac{\Delta W_{1,eq}}{G_{max}} = 10^{\left(d_1\Gamma^5 + d_2\Gamma^4 + d_3\Gamma^3 + d_4\Gamma^2 + d_5\Gamma + d_6\right)}$

   Table 6.14: Regression Coefficients for the Fit of the $\Delta W_{1,eq}/G_{max}$ Curves (Clean Sand)

σ'_{m0} (kPa)	d_1	d_2	d_3	d_4	d_5	d_6
25	0.242	5.848	56.101	267.91	640.3145	608.05276
50	0.232	5.545	52.685	249.053	589.1456	553.24074
100	0.22	5.219	49.129	229.977	538.6714	500.38834
200	0.208	4.881	45.541	211.152	489.8841	450.27458

Table 6.15: Regression Coefficients for the Fit of the $\Delta W_{1,eq}/G_{max}$ Curves (Sand with High Fines Content)

σ'_{m0} (kPa)	d_1	d_2	d_3	d_4	d_5	d_6
25	-0.0249	-0.4056	-1.7385	3.74134	45.391	80.934471
50	0.015	0.4993	6.2671	38.1141	116.1858	135.54471
100	0.0401	1.043	10.814	56.2139	149.4765	156.46374
200	0.0544	1.33	12.98	63.487	158.8015	156.90162

Alternatively, values of $\Delta W_{1,eq}/G_{max}$ can be obtained from Figures 6.21 through 6.23. In either case, use the figures to verify that the threshold shear strain is exceeded. (i.e. Do the values plot above the diamonds? As a rule of thumb, Γ should be greater than approximately -4.5.) If the threshold shear strain is not exceeded, liquefaction is not likely for the given peak ground acceleration.

8. Calculate the total normalized dissipated energy for the given conditions. This is your demand term:
   $\frac{\Delta W}{\sigma'_{v0}} = \left(\frac{\Delta W_{1,eq}}{G_{max}}\right) \cdot \frac{G_{max}}{\sigma'_{v0}} \cdot (2 \cdot n_{eq})$

9. Calculate the factor of safety (FS) against liquefaction:
   $FS = \frac{0.30N_{1,60cs}+0.98}{\ln\left(\frac{\Delta W}{\sigma'_{v0}}\right)+13.93}$

10. Alternatively, calculate the probability of liquefaction, P:
    $P = 1 - \Phi\left[\frac{0.30N_{1,60cs} - \ln\left(\frac{\Delta W}{\sigma'_{v0}}\right) - 11.61}{1.29}\right]$

Non-Simplified Procedure

The "non-simplified" procedure requires a well-characterized soil profile, representative earthquake ground motions, and equivalent-linear site response analyses.

1. Obtain information about the soil profile. For every layer down to the bedrock halfspace:

- Shear wave velocity (V_S)
- Thickness of layer
- Unit weight
- Fines content (FC) and plasticity index (PI) for shear modulus reduction and damping curves

For layers of interest:

- SPT blow count (N_m)
- Total overburden stress (σ_v) via the unit weights
- Initial effective overburden stress (σ'_{v0}) via the depth of the ground water table
- Various SPT details:
  - Energy Ratio (ER)
  - Borehole diameter
  - Rod length
  - Sampler information

2. Correct your measured SPT blow counts using step 2 of the Simplified Procedure (above). If the corrected SPT blow count of a layer is greater than 30, it is unlikely to liquefy.

3. Obtain representative earthquake motions.

4. Perform equivalent-linear site response analyses for the profile using each of your motions. You should use Darendeli and Stokoe's (2001) shear modulus reduction and damping curves. From the response strain time histories, verify that the threshold shear strain is exceeded for each layer of interest. If it is not exceeded, liquefaction in that layer is unlikely.

5. For each layer of interest:

   a. Calculate the dissipated energy. This can be done using the stress and strain time histories and the trapezoidal rule:
   $\Delta W = \frac{1}{2} \sum_{i=1}^{j-1} \left((\tau_{i+1} + \tau_i) \cdot (\gamma_{i+1} - \gamma_i) \right)$

where τ and γ are the response stress and strain time histories, respectively, of the layer of interest. j is the total number of data points in the input motion.

b. Normalize the dissipated energy by the initial effective overburden stress (σ'_{v0}). This is your demand term ($\Delta W/\sigma'_{v0}$).

c. Calculate the factor of safety (FS) against liquefaction for each layer of interest:
$FS = \frac{0.30N_{1,60cs}+0.98}{\ln\left(\frac{\Delta W}{\sigma'_{v0}}\right)+13.93}$

d. Alternatively, calculate the probability of liquefaction, P:
$P = 1 - \Phi\left[\frac{0.30N_{1,60cs}-\ln\left(\frac{\Delta W}{\sigma'_{v0}}\right)-11.61}{1.29}\right]$

References

Boulanger, R. W., (2003). "High overburden stress effects in liquefaction analyses," *J. Geotechnical and Geoenvironmental Eng.*, ASCE 129(12), 1071–082.

Darendeli, M. B., and Stokoe, K. H. (2001). Development of a new family of normalized modulus reduction and material damping curves. Geotechnical Engineering Report GD01-1, University of Texas at Austin.

Hatanaka, M., and Uchida, A. (1996). "A simple method for the determination of Ko-value in sandy soils." *Soils and Foundations*, 36(2), 93–99.

Idriss, I. M., and Boulanger, R. W. (2008). *Soil Liquefaction During Earthquakes*. Earthquake Engineering Research Institute, Oakland, California.

Jaky, J. (1944). "The coefficient of earth pressure at rest." *Journal of the Society of Hungarian Architects and Engineers*, 78(22), 355–358.

Jaky, J. (1948). "Influence of ground water level oscillation on subsidence of structures." *Proc. II. Int. Conf. Soil Mech.*, Rotterdam.

Lasley, S.J., Green, R.A., and Rodriguez-Marek, A. (2015a) "Development of a number of equivalent cycles correlation based on dissipated energy." *In preparation.*

Lasley, S.J., Green, R.A., and Rodriguez-Marek, A. (2015b) "A new stress reduction coefficient relationship for liquefaction triggering analyses." *In preparation.*

National Center for Earthquake Engineering Research (NCEER), 1997. *Proceedings of the NCEER Workshop on Evaluation of Liquefaction Resistance of Soils*, T. L. Youd and I. M. Idriss, editors, Technical Report NCEER-97-022, 41-88.

Ohta, Y. and Goto, N. (1976). "Estimation of S-wave velocity in terms of characteristic indices of soil." *Butsuri-Tanko*, 29(4), 34-41.

Sabatini, P. J., Bachus, R. C., Mayne, P. W., Schneider, J. A., and Zettler, T. E. (2002). "Geotechnical engineering circular no. 5: Evaluation of Soil and Rock Properties." Federal Highway Administration, Washington, D.C., 385.

Seed, H. B., Tokimatsu, K., Harder, L. F. Jr., and Chung, R., (1984). "The influence of SPT procedures on soil liquefaction resistance evaluations," Report No. UCB/EERC-84/15, Earthquake Engineering Research Center, University of California at Berkeley.

Seed, H. B., Wong, R. T., Idriss, I. M., and Tokimatsu, K. (1986). "Moduli and Damping Factors for Dynamic Analyses of Cohesionless Soils." *Journal of Geotechnical Engineering*, 112(11), 1016–1032.

Seed, R. B., Cetin, K. O., Moss, R. E. S., Kammerer, A., Wu, J., Pestana, J., Riemer, M., (2001). "Recent advances in soil liquefaction engineering and seismic site response evaluation," *Proceedings, 4th International Conference and Symposium on Recent Advances in Geotechnical Earthquake Engineering and Soil Dynamics*, University of Missouri, Rolla, MO, Paper SPL-2.

Skempton, A. W., 1986. "Standard penetration test procedures and the effects in sands of overburden pressure, relative density, particle size, aging and overconsolidation," *Geotechnique*, 36(3), 425–47.

Tokimatsu, K., and Seed, H. B. (1987). "Evaluation of settlements in sands due to earthquake shaking." *Journal of Geotechnical Engineering*, 113(8), 861–878.

Figures

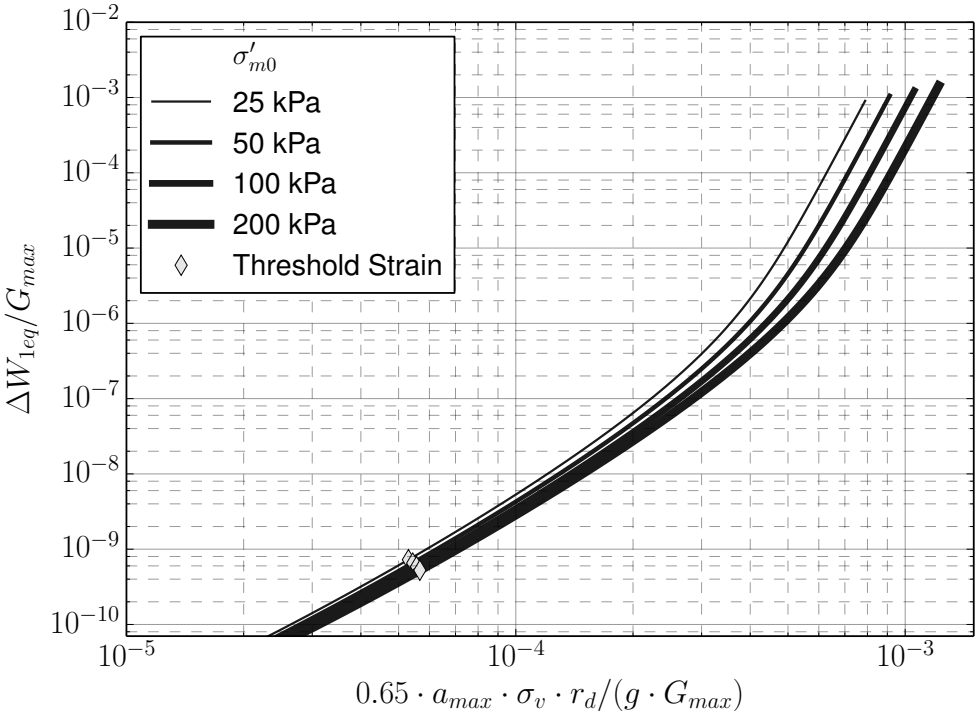


Figure 6.21: Direct Method to estimate $\Delta W_{1eq}/G_{max}$ (after Tokimatsu and Seed 1987) using the Darendeli and Stokoe (2001) degradation curves. Lines correspond to a clean sand ($FC = 0\%$, $PI = 0$). To use the figure, calculate the value on the x-axis, read up to the appropriate curve, and obtain the corresponding value from the y-axis.

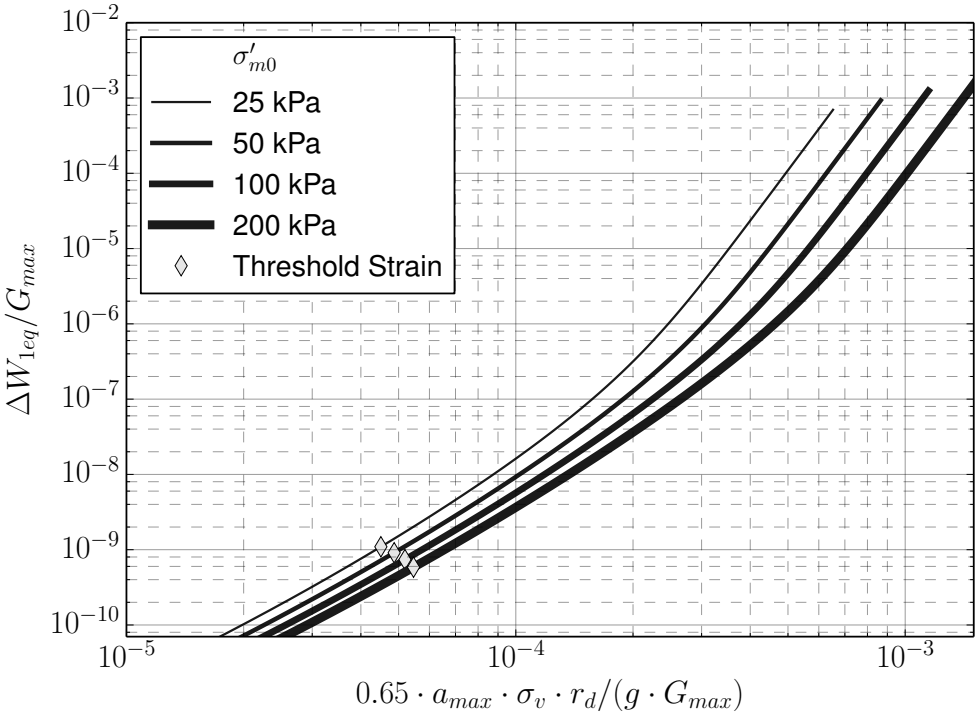


Figure 6.22: Direct Method to estimate $\Delta W_{1eq}/G_{max}$ (after Tokimatsu and Seed 1987) using the Darendeli and Stokoe (2001) degradation curves. Lines correspond to "sands with high fines content." To use the figure, calculate the value on the x-axis, read up to the appropriate curve, and obtain the corresponding value from the y-axis.

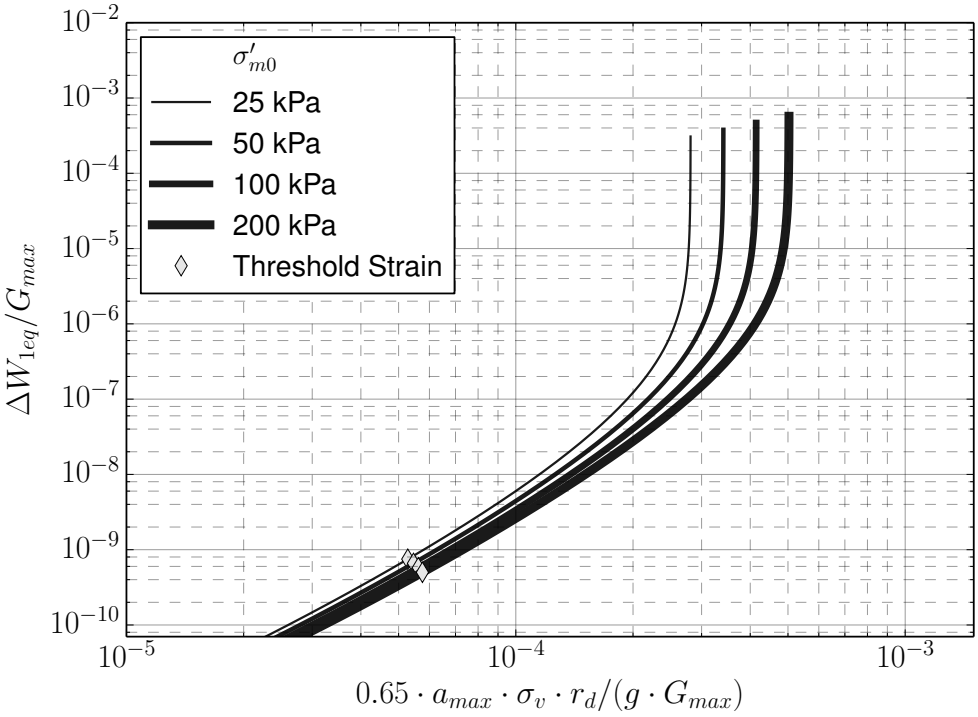


Figure 6.23: Direct Method to estimate $\Delta W_{1eq}/G_{max}$ (after Tokimatsu and Seed 1987) using the Darendeli and Stokoe (2001) degradation curves. Lines correspond to a low-plasticity silt. To use the figure, calculate the value on the x-axis, read up to the appropriate curve, and obtain the corresponding value from the y-axis.

Chapter 7

Conclusions

7.1 Summary of Research

The purpose of the research presented herein is the application of fatigue theories to two important geotechnical earthquake engineering phenomena: 1. The development of a "non-simplified" seismic compression evaluation method, and 2. The development of an energy-based liquefaction potential evaluation procedure that will be familiar to users of the stress-based simplified procedure.

In pursuit of these research objectives, several sub-tasks and related research was performed. This section summarizes the results of the entire body of research.

7.1.1 Seismic Compression Method

Proposed in Chapter 3 is a "non-simplified" method for predicting the severity of seismic compression. It is based on a modified version of the Richart-Newmark (R-N; Richart and Newmark 1948) cumulative damage hypothesis and uses volumetric strain as the damage metric. The proposed method estimates volumetric strain from any shear strain time history. It was calibrated using the results of 460 cyclic simple shear tests performed on dry quartz sand specimens at Virginia Tech. The relative density of the test specimens ranged from 30 to 80%. At each relative density, specimens were subjected

to strain-controlled sinusoidal loading with shear strains ranging from 0.2 to 1%. The overburden pressures of the tests were 50, 100, and 250 kPa.

In addition to the 460 cyclic simple shear tests performed with sinusoidal loading, 23 tests were performed with variable-amplitude sinusoidal loading, and 16 tests were performed with earthquake strain time histories as the loading functions. These variable-amplitude and earthquake loading tests were used to validate the R-N-based seismic compression method. The proposed method accurately predicted volumetric strains, especially compared to the Palmgren-Miner (Palmgren 1924; Miner 1945) method. Additionally, a case history was presented in which the proposed R-N-based seismic compression method predicted reasonable volumetric strains using the strain time histories of equivalent-linear site response analyses.

In addition to accurately predicting seismic compression in a non-simplified approach, the proposed seismic compression method can be used to compute number of equivalent cycles for use in simplified procedures. Compared to the number of equivalent cycle correlations commonly used in other simplified seismic compression procedures, the number of equivalent cycles computed using the proposed method uses a consistent damage metric (volumetric strain) to equate earthquake motions to an equivalently-damaging sinusoidal motion with a specified number of cycles.

7.1.2 Number of Equivalent Stress Cycles Correlation

The number of equivalent stress cycles (n_{eq}) concept plays an important role in geotechnical earthquake engineering and, particularly, in liquefaction evaluations. It allows for the conversion of an earthquake motion into an equivalently-damaging number of sinusoidal cycles of equal amplitude. Most of the current applications of n_{eq} in liquefaction evaluations assume high cycle fatigue conditions (many cycles to failure; loading remains in the elastic range) in conjunction with the Palmgren-Miner (Palmgren 1924; Miner 1945) cumulative damage hypothesis. Soil liquefaction, however, is associated with low-cycle fatigue (few cycles to failure; significant damage in each cycle) conditions. Proposed in Chapter 4 is a low-cycle alternative application of the Palmgren-Miner cumulative damage hypothesis using dissipated energy as the damage metric.

The n_{eq} correlation proposed in Chapter 4 was developed using the results

of equivalent-linear site response analyses. These analyses were performed using 50 soil profiles from liquefaction sites (Cetin 2000) and 228 pairs of rock motions representing shallow-crustal and stable continental tectonic regimes. The n_{eq} correlation was regressed using two functional forms and incorporated the random effects of earthquake event and profile group. One of these correlations is a function of peak ground acceleration and earthquake magnitude, and the other correlation is a function of magnitude and site-to-source distance. Calibration coefficients are provided for both active shallow-crustal and stable continental tectonic regimes.

Because the n_{eq} correlations proposed herein are not a function of magnitude alone, they demonstrate the need for magnitude scaling factors ($MSFs$) that are likewise. Magnitude scaling factors play an important role in stress-based simplified liquefaction evaluation procedures and are based on n_{eq} correlations; most of the magnitude scaling factors currently in use are a function of magnitude alone.

Also demonstrated in the development of the n_{eq} correlation is that low-cycle n_{eq} values are more strongly correlated with earthquake intensity measures (i.e. magnitude, bracketed duration, significant duration, rms acceleration, and a_{max}) than traditional high-cycle n_{eq} values.

7.1.3 Stress Reduction Coefficient Correlation

Based on the results of the equivalent-linear site response analyses discussed in conjunction with the proposed n_{eq} correlation, a new stress reduction coefficient relationship (Chapter 5) was also developed. The stress reduction coefficient allows for the computation of the seismically induced stresses at depth in a soil profile without the need to perform site response analyses. As such, it is an important part of simplified liquefaction evaluation procedures. A simple functional form was developed with two variants. One variant is a function of earthquake magnitude, the average shear wave velocity in the upper 12 m of a profile (V_{S12}), and depth; the other variant is a function of magnitude and depth. Both variants yield r_d values with less bias and uncertainty than other common r_d relationships (i.e. Liao and Whitman 1986, Idriss 1999, Cetin 2000). Calibration coefficients for both variants are provided for both active shallow-crustal and stable continental tectonic regimes.

7.1.4 Dissipated-Energy-Based Liquefaction Evaluation

A simplified liquefaction evaluation procedure was developed based on dissipated energy and an SPT liquefaction case history database (Chapter 6). It was developed using the aforementioned n_{eq} correlation and r_d relationship and the Darendeli and Stokoe (2001) shear modulus and damping degradation curves. The normalized dissipated energy ($\Delta W/\sigma'_{v0}$) was calculated for each case history in the database, and a limit state capacity curve was regressed as a function of corrected SPT blow count. The regression was performed in two different ways: by assuming the input values are known exactly and by accounting for uncertainties in each of the input parameters.

The resulting limit state capacity curve has a simple functional form and can be used both deterministically (returning a factor of safety against liquefaction) or probabilistically (returning a probability of liquefaction). Compared to two other common stress-based simplified procedures (i.e. Cetin et al. 2004; Boulanger and Idriss 2012), the energy-based procedure proposed herein has a similar rate of correct prediction of the occurrence/non-occurrence of liquefaction for the case histories of the case history database.

Overburden correction factors (K_σ) were derived for stress-based liquefaction procedures using dissipated energy as the damage metric. The derived correction factor curves are influenced by the shear modulus and damping degradation curves (Darendeli and Stokoe 2001, in this case) and compared favorably with two other sets of K_σ curves (i.e. Hynes and Olsen 1999; Boulanger and Idriss 2004).

7.2 Recommendations for Future Work

Considerable progress has been made in the application of fatigue theories to seismic compression estimation and to the evaluation of liquefaction potential. In the case of seismic compression, a modified version of the Richart-Newmark cumulative damage hypothesis accurately predicts seismic compression using shear strain time histories. However, as mentioned previously, a number-of-equivalent-strain-cycles correlation still needs to be developed in order to develop a simplified seismic compression method based on the

same principles.

In the case of liquefaction evaluation, an alternative, low-cycle application of the Palmgren-Miner hypothesis has been used to develop a dissipated-energy-based number-of-equivalent-cycles correlation. This correlation, in turn, was used to successfully develop an SPT-based liquefaction evaluation procedure. Using the same general framework, CPT- and shear wave velocity-based procedures could be developed to cover most use cases.

Finally, the procedures developed herein have been suggested for use in tectonic regimes such as that of the central-eastern United States and with non-earthquake sources of loading. A validation of these procedures in those use cases is warranted.

7.3 References

Boulanger, R. W. and Idriss, I. M., (2004). "State normalization of penetration resistances and the effect of overburden stress on liquefaction resistance," *Proceedings, 11th International Conference on Soil Dynamics and Earthquake Engineering, and 3rd International Conference on Earthquake Geotechnical Engineering*, D. Doolin et al., eds., Stallion Press, Vol. 2, pp. 484–91.

Boulanger, R., and Idriss, I. (2012). "Probabilistic Standard Penetration Test–Based Liquefaction–Triggering Procedure." *Journal of Geotechnical and Geoenvironmental Engineering*, 138(10), 1185–1195.

Cetin, K. O. (2000). "Reliability-based assessment of seismic soil liquefaction initiation hazard." University of California, Berkeley, Berkeley, CA.

Cetin, K. O., Seed, R. B., Der Kiureghian, A., Tokimatsu, K., Harder, L. F., Kayen, R. E., and Moss, R. E. S. (2004). "Standard Penetration Test-Based Probabilistic and Deterministic Assessment of Seismic Soil Liquefaction Potential." *Journal of Geotechnical and Geoenvironmental Engineering*, 130(12), 1314–1340.

Darendeli, M. B., and Stokoe, K. H. (2001). "Development of a new family of normalized modulus reduction and material damping curves." Geotechnical Engineering Report GD01-1, University of Texas at Austin.

Hynes, M. E., and Olsen, R. S. (1999). "Influence of confining stress on liquefaction resistance." *Proc., Int. Workshop on Phys. and Mech. of Soil Liquefaction*, Balkema, Rotterdam, The Netherlands, 145–152.

Idriss I. M. (1999) "An update to the Seed-Idriss simplified procedure for evaluating liquefaction potential.", *Proc., TRB Workshop on New Approaches to Liquefaction*, Publication No. FHWA-RD-99- 165, Federal Highway Administration.

Liao, S. S. C., and Whitman, R. V. (1986). "Catalogue of liquefaction and non-liquefaction occurrences during earthquakes." Res. Rep., Dept. of Civ. Engrg., Massachusetts Institute of Technology, Cambridge, Mass.

Miner, M. A. (1945). "Cumulative Damage in Fatigue." *Journal of Applied Mechanics*, 12(3), A–159-A–164.

Palmgren, A. (1924). "Die lebensdauer von kugellagern (Life length of roller bearings, in German)." *Zeitschrift des Vereins Deutscher Ingenieure*, 68(14), 339-341.

Richart, F.E. and Newmark, N.M. (1948). "An hypothesis for determination of cumulative damage in fatigue," *ASTM Proceedings*, 48: 767-800.

Appendix A

Contents of Appendices

A Contents of Appendices

B Seismic Compression Lab Test Results

C Cycle Counting Methods for Seismic Compression

D Cyclic Simple Shear Lab Test Results

E Soil Profiles of the Equivalent-Linear Analyses

F Earthquake Motions

G Cyclic Simple Shear Testing Manual

H Cyclic Triaxial Testing Manual

I Conference Paper: Verification of ShakeVT2

J Code

Appendix B

Seismic Compression Lab Test Results

This section contains the results of the strain-controlled, cyclic simple shear tests performed for seismic compression. The tests were performed on dry quartz (pool filter) sand. The overburden pressure was held constant during cyclic loading allowing seismic compression (volumetric strain) to occur. See Chapter 3 for more details. The first section contains the results of the constant-amplitude sinusoidal loading tests. The second section contains the results of the irregular loading tests.

B.1 Constant-Amplitude Results

Tables B.1 and B.2 give the results of the constant-amplitude sinusoidal tests. The variable γ is the single-amplitude shear strain; $N_{\varepsilon_v=1\%}$ is the number of cycles to one percent volumetric strain; r_0 is the calibration coefficient for the Richart-Newmark cumulative damage hypothesis (asymptotic r); and C_1 and C_2 are the calibration coefficients for the Byrne method. The path of the file corresponds to the overburden pressure and relative density bins (e.g. ./Sinusoids/50Pa/Dr30/121.csv).

Table B.1: Details and Calibration Coefficients of All Constant-Amplitude CSS Tests

No.	$D_{r,nom.}$	$D_{r,est.}$	γ (SA)	σ_v	$N_{\varepsilon_v=1\%}$	r_0	C_1	C_2	File
1	30.0	55.2	0.002	100	860	0.085	0.69	1.39	121.csv
2	30.0	55.2	0.0039	100	82	0.218	0.49	1.71	125.csv
3	30.0	55.2	0.0089	100	7	0.492	0.51	1.45	129.csv
4	30.0	55.2	0.0024	100	258	0.112	0.84	1.4	182.csv

No.	$D_{r,nom.}$	$D_{r,est.}$	γ (SA)	σ_v	$N_{\varepsilon_v=1\%}$	r_0	C_1	C_2	File
5	30.0	55.2	0.0024	100	471	0.138	0.55	1.65	186.csv
6	30.0	55.2	0.002	100	521	0.152	0.58	1.33	122.csv
7	30.0	55.2	0.0019	100	1275	0.097	0.75	1.56	124.csv
8	30.0	55.2	0.004	100	30	0.297	0.73	1.32	126.csv
9	30.0	55.2	0.0039	100	41	0.266	0.63	1.41	128.csv
10	30.0	55.2	0.0089	100	10	0.461	0.31	1.33	130.csv
11	30.0	55.2	0.0089	100	8	0.299	0.45	1.54	132.csv
12	30.0	55.2	0.0024	100	326	0.148	0.5	1.38	183.csv
13	30.0	55.2	0.0024	100	96	0.189	0.87	1.13	185.csv
14	30.0	55.2	0.0026	100	73	0.224	0.74	1.1	187.csv
15	30.0	55.2	0.0026	100	71	0.215	0.7	1.04	189.csv
16	30.0	55.2	0.003	100	115	0.181	0.68	1.49	192.csv
17	30.0	55.2	0.003	100	114	0.188	0.48	1.32	194.csv
18	30.0	55.2	0.0051	100	20	0.328	0.48	1.24	196.csv
19	30.0	55.2	0.0051	100	24	0.405	0.4	1.25	198.csv
20	30.0	55.2	0.0061	100	14	0.347	0.51	1.3	200.csv
21	30.0	55.2	0.0061	100	19	0.275	0.51	1.65	202.csv
22	30.0	55.2	0.002	100	306	0.148	0.6	1.16	333.csv
23	30.0	55.2	0.0024	100	117	0.167	0.99	1.23	343.csv
24	30.0	55.2	0.0026	100	183	0.16	0.7	1.42	354.csv
25	30.0	88.9	0.003	100	222	0.154	0.45	1.59	361.csv
26	30.0	55.2	0.004	100	33	0.34	0.54	1.2	368.csv
27	30.0	55.2	0.0061	100	15	0.325	0.66	1.6	380.csv
28	30.0	55.2	0.005	100	40	0.292	0.29	1.4	54.csv
29	30.0	55.2	0.0069	100	39	0.376	0.16	1.59	57.csv
30	30.0	55.2	0.0079	100	20	0.453	0.29	1.81	61.csv
31	30.0	55.2	0.0024	100	476	0.101	1.0	1.75	63.csv
32	30.0	55.2	0.0039	100	59	0.202	0.62	1.6	65.csv
33	30.0	55.2	0.002	100	1688	0.094	1.03	1.87	67.csv
34	30.0	55.2	0.0025	100	303	0.111	1.01	1.6	71.csv
35	30.0	55.2	0.0023	100	1485	0.109	1.73	2.89	75.csv
36	30.0	55.2	0.0025	100	1075	0.109	1.74	3.0	77.csv
37	30.0	55.2	0.006	100	22	0.282	0.38	1.51	79.csv
38	30.0	55.2	0.0074	100	10	0.299	0.47	1.48	81.csv
39	30.0	55.2	0.0074	100	13	0.285	0.43	1.66	85.csv
40	30.0	55.2	0.0026	100	476	0.144	0.53	1.76	87.csv
41	30.0	55.2	0.0023	100	615	0.101	0.71	1.63	89.csv
42	30.0	55.2	0.0074	100	20	0.317	0.31	1.68	91.csv
43	30.0	55.2	0.0079	100	8	0.413	0.6	1.45	94.csv
44	30.0	55.2	0.0026	100	232	0.161	0.61	1.48	190.csv
45	30.0	55.2	0.003	100	102	0.202	0.44	1.23	195.csv
46	30.0	55.2	0.0051	100	21	0.294	0.6	1.41	199.csv
47	30.0	55.2	0.0061	100	21	0.287	0.37	1.47	203.csv
48	30.0	55.2	0.0061	100	14	0.299	0.58	1.45	205.csv

No.	$D_{r,nom.}$	$D_{r,est.}$	γ (SA)	σ_v	$N_{\varepsilon_v=1\%}$	r_0	C_1	C_2	File
49	30.0	55.2	0.002	100	697	0.117	0.71	1.49	335.csv
50	30.0	55.2	0.0024	100	168	0.145	1.19	1.47	342.csv
51	30.0	55.2	0.0024	100	272	0.123	1.1	1.63	344.csv
52	30.0	55.2	0.004	100	64	0.219	0.47	1.55	367.csv
53	30.0	55.2	0.0061	100	18	0.333	0.46	1.44	381.csv
54	30.0	55.2	0.0089	100	13	0.408	0.26	1.52	58.csv
55	30.0	55.2	0.0099	100	14	0.328	0.27	1.94	60.csv
56	30.0	55.2	0.0059	100	37	0.265	0.27	1.6	62.csv
57	30.0	55.2	0.0049	100	24	0.255	0.58	1.42	66.csv
58	30.0	55.2	0.0027	100	210	0.123	0.91	1.61	74.csv
59	30.0	55.2	0.0059	100	45	0.255	0.24	1.73	78.csv
60	30.0	55.2	0.0074	100	25	0.328	0.21	1.58	82.csv
61	30.0	55.2	0.0074	100	25	0.288	0.24	1.73	83.csv
62	30.0	55.2	0.0025	100	216	0.149	0.72	1.47	86.csv
63	30.0	55.2	0.0023	100	299	0.134	0.62	1.37	90.csv
64	30.0	55.2	0.0079	100	13	0.395	0.39	1.6	95.csv
65	30.0	55.2	0.0022	100	850	0.107	0.78	1.69	96.csv
66	30.0	55.2	0.002	100	521	0.131	0.69	1.36	123.csv
67	30.0	55.2	0.004	100	26	0.317	0.63	1.13	127.csv
68	30.0	55.2	0.0089	100	7	0.271	0.44	1.38	131.csv
69	30.0	55.2	0.0024	100	287	0.154	0.45	1.3	184.csv
70	30.0	55.2	0.0026	100	101	0.195	0.77	1.24	188.csv
71	30.0	55.2	0.003	100	73	0.232	0.48	1.12	193.csv
72	30.0	55.2	0.0051	100	21	0.322	0.43	1.18	197.csv
73	30.0	55.2	0.0061	100	18	0.338	0.46	1.48	201.csv
74	30.0	55.2	0.0026	100	99	0.193	0.56	1.08	355.csv
75	30.0	55.2	0.0061	100	16	0.341	0.52	1.47	379.csv
76	30.0	55.2	0.002	100	1436	0.103	0.54	1.55	56.csv
77	30.0	55.2	0.0029	100	163	0.148	0.77	1.61	64.csv
78	30.0	55.2	0.0099	100	17	0.463	0.29	2.23	68.csv
79	30.0	55.2	0.0025	100	525	0.114	0.86	1.86	76.csv
80	30.0	55.2	0.0059	100	27	0.272	0.33	1.53	80.csv
81	30.0	55.2	0.0025	100	500	0.114	0.66	1.71	88.csv
82	30.0	55.2	0.0064	100	20	0.315	0.37	1.53	93.csv
83	45.0	69.3	0.002	100	395	0.14	0.58	1.2	206.csv
84	45.0	69.3	0.002	100	618	0.094	0.67	1.31	210.csv
85	45.0	69.3	0.0024	100	62	0.236	0.82	0.94	214.csv
86	45.0	69.3	0.0026	100	117	0.171	0.8	1.28	218.csv
87	45.0	69.3	0.0031	100	33	0.286	0.83	1.0	222.csv
88	45.0	69.3	0.002	100	1241	0.116	1.81	2.33	207.csv
89	45.0	69.3	0.002	100	387	0.118	0.62	1.13	209.csv
90	45.0	69.3	0.0024	100	83	0.2	0.75	1.03	211.csv
91	45.0	69.3	0.0024	100	127	0.205	0.55	1.07	213.csv
92	45.0	69.3	0.0026	100	610	0.087	0.81	1.89	215.csv

No.	$D_{r,nom.}$	$D_{r,est.}$	γ (SA)	σ_v	$N_{\varepsilon_v=1\%}$	r_0	C_1	C_2	File
93	45.0	69.3	0.0027	100	68	0.251	0.46	0.89	217.csv
94	45.0	69.3	0.0026	100	89	0.189	0.98	1.25	219.csv
95	45.0	69.3	0.003	100	68	0.229	0.72	1.26	221.csv
96	45.0	69.3	0.003	100	120	0.171	0.91	1.62	223.csv
97	45.0	69.3	0.003	100	75	0.208	0.81	1.37	225.csv
98	45.0	69.3	0.004	100	55	0.239	0.7	1.73	227.csv
99	45.0	69.3	0.004	100	26	0.292	0.87	1.38	229.csv
100	45.0	69.3	0.005	100	26	0.297	0.58	1.56	231.csv
101	45.0	69.3	0.005	100	42	0.257	0.42	1.71	233.csv
102	45.0	53.2	0.0061	100	19	0.311	0.54	1.69	235.csv
103	45.0	69.3	0.006	100	32	1.0	0.41	1.91	237.csv
104	45.0	69.3	0.0091	100	10	0.446	0.43	1.74	239.csv
105	45.0	69.3	0.0091	100	10	0.428	0.46	1.88	241.csv
106	45.0	69.3	0.0091	100	13	0.36	0.45	2.26	243.csv
107	45.0	69.3	0.002	100	1063	0.086	0.91	1.6	337.csv
108	45.0	69.3	0.0024	100	151	0.146	1.13	1.37	345.csv
109	45.0	69.3	0.0024	100	849	0.083	1.18	2.05	347.csv
110	45.0	69.3	0.0024	100	616	0.093	1.16	1.94	349.csv
111	45.0	69.3	0.0024	100	1133	0.075	1.29	2.14	351.csv
112	45.0	69.3	0.0026	100	944	0.075	0.78	2.04	357.csv
113	45.0	69.3	0.0026	100	771	0.082	0.93	2.03	359.csv
114	45.0	69.3	0.003	100	157	0.168	0.5	1.48	363.csv
115	45.0	69.3	0.003	100	168	0.144	0.92	1.78	365.csv
116	45.0	69.3	0.004	100	49	0.22	0.9	1.76	369.csv
117	45.0	69.3	0.004	100	136	0.17	0.55	2.13	371.csv
118	45.0	69.3	0.0041	100	41	0.228	0.58	1.41	373.csv
119	45.0	69.3	0.0061	100	22	0.294	0.54	1.8	383.csv
120	45.0	69.3	0.0041	100	16	0.359	0.93	1.11	226.csv
121	45.0	69.3	0.004	100	18	0.337	0.86	1.1	230.csv
122	45.0	69.3	0.0051	100	25	0.278	0.55	1.49	234.csv
123	45.0	69.3	0.0061	100	18	0.311	0.55	1.63	238.csv
124	45.0	69.3	0.0092	100	14	0.564	0.34	2.01	242.csv
125	45.0	69.3	0.002	100	1209	0.088	1.01	1.68	338.csv
126	45.0	69.3	0.0024	100	1132	0.069	0.97	2.05	348.csv
127	45.0	55.0	0.0026	100	325	0.117	0.74	1.65	356.csv
128	45.0	69.3	0.003	100	415	0.116	0.72	2.08	362.csv
129	45.0	69.3	0.003	100	104	0.191	0.6	1.39	366.csv
130	45.0	69.3	0.004	100	100	0.181	0.58	1.96	372.csv
131	45.0	69.3	0.0061	100	30	0.358	0.36	1.76	384.csv
132	45.0	69.3	0.0091	100	11	0.46	0.4	1.85	394.csv
133	45.0	69.3	0.002	100	1938	0.073	0.85	1.67	208.csv
134	45.0	69.3	0.0024	100	131	0.171	0.82	1.23	212.csv
135	45.0	69.3	0.0026	100	94	0.188	0.84	1.21	216.csv
136	45.0	69.3	0.003	100	43	0.23	1.17	1.26	220.csv

No.	$D_{r,nom.}$	$D_{r,est.}$	γ (SA)	σ_v	$N_{\varepsilon_v=1\%}$	r_0	C_1	C_2	File
137	45.0	55.0	0.003	100	241	0.134	1.0	2.0	224.csv
138	45.0	69.3	0.004	100	23	0.311	0.79	1.22	228.csv
139	45.0	69.3	0.005	100	36	0.251	0.53	1.74	232.csv
140	45.0	69.3	0.0061	100	20	0.307	0.49	1.61	236.csv
141	45.0	69.3	0.0091	100	15	0.312	0.44	2.34	240.csv
142	45.0	69.3	0.002	100	1826	0.11	1.07	2.05	336.csv
143	45.0	69.3	0.0024	100	388	0.094	1.02	1.65	346.csv
144	45.0	69.3	0.0024	100	489	0.101	0.7	1.5	350.csv
145	45.0	69.3	0.0026	100	240	0.111	1.36	1.75	358.csv
146	45.0	69.3	0.003	100	363	0.118	0.82	2.14	364.csv
147	45.0	69.3	0.004	100	173	0.165	0.56	2.37	370.csv
148	45.0	69.3	0.006	100	31	0.273	0.34	1.74	382.csv
149	65.0	88.6	0.002	100	255	0.113	1.06	1.19	246.csv
150	65.0	88.6	0.0024	100	4186	0.09	0.86	2.58	252.csv
151	65.0	88.7	0.0024	100	1221	0.07	0.93	1.93	254.csv
152	65.0	88.6	0.0026	100	3470	0.084	0.92	2.88	255.csv
153	65.0	88.6	0.0026	100	1037	0.086	0.9	2.17	258.csv
154	65.0	88.6	0.003	100	669	0.095	0.6	2.07	262.csv
155	65.0	88.6	0.002	100	686	0.073	1.22	1.45	248.csv
156	65.0	88.6	0.0024	100	1127	0.065	1.16	1.97	253.csv
157	65.0	88.6	0.0026	100	532	0.087	0.77	1.75	257.csv
158	65.0	88.6	0.0026	100	775	0.083	0.74	1.9	259.csv
159	65.0	88.6	0.003	100	718	0.102	0.54	2.18	261.csv
160	65.0	88.6	0.003	100	536	0.106	0.65	2.13	263.csv
161	65.0	88.6	0.004	100	57	0.209	0.67	1.68	265.csv
162	65.0	88.6	0.004	100	96	0.17	0.67	1.99	267.csv
163	65.0	88.7	0.004	100	233	0.139	0.6	2.5	269.csv
164	65.0	88.6	0.005	100	65	0.195	0.92	2.62	271.csv
165	65.0	88.6	0.005	100	73	1.0	0.5	2.19	273.csv
166	65.0	88.6	0.006	100	35	0.214	0.98	2.74	275.csv
167	65.0	88.7	0.0061	100	36	0.272	1.0	2.99	277.csv
168	65.0	88.6	0.006	100	41	0.267	1.07	3.17	279.csv
169	65.0	88.6	0.0091	100	18	0.3	0.51	2.84	281.csv
170	65.0	88.6	0.0091	100	29	0.274	0.28	2.8	283.csv
171	65.0	88.6	0.002	100	1273	0.044	1.18	1.54	339.csv
172	65.0	88.6	0.0024	100	1717	0.071	1.15	2.24	352.csv
173	65.0	88.6	0.0026	100	515	0.079	1.51	2.04	360.csv
174	65.0	88.6	0.004	100	89	0.173	0.77	2.06	375.csv
175	65.0	88.6	0.004	100	89	0.175	0.69	1.99	377.csv
176	65.0	88.6	0.0061	100	56	0.231	0.47	2.69	386.csv
177	65.0	88.6	0.0091	100	18	0.276	0.62	3.08	389.csv
178	65.0	88.7	0.004	100	548	0.116	0.45	2.86	266.csv
179	65.0	88.6	0.004	100	592	0.124	0.67	3.52	270.csv
180	65.0	88.6	0.005	100	75	0.191	0.65	2.42	274.csv

No.	$D_{r,nom.}$	$D_{r,est.}$	γ (SA)	σ_v	$N_{\varepsilon_v=1\%}$	r_0	C_1	C_2	File
181	65.0	88.6	0.0061	100	16	0.301	0.85	1.92	278.csv
182	65.0	88.6	0.009	100	15	0.289	0.59	2.72	282.csv
183	65.0	88.6	0.002	100	1025	0.072	1.15	1.57	340.csv
184	65.0	88.6	0.004	100	258	0.138	0.63	2.68	374.csv
185	65.0	88.7	0.0061	100	33	0.242	0.69	2.38	385.csv
186	65.0	88.6	0.0091	100	18	0.32	0.5	2.88	390.csv
187	65.0	88.6	0.0091	100	25	0.291	0.4	3.01	391.csv
188	65.0	88.6	0.003	100	781	0.092	0.58	2.23	260.csv
189	65.0	88.6	0.004	100	74	0.18	0.65	1.81	264.csv
190	65.0	88.6	0.004	100	183	1.0	0.59	2.38	268.csv
191	65.0	88.6	0.005	100	96	0.175	0.57	2.54	272.csv
192	65.0	88.6	0.006	100	53	0.242	0.79	3.11	276.csv
193	65.0	88.6	0.0091	100	21	0.296	0.4	2.72	280.csv
194	65.0	88.6	0.0091	100	24	1.0	0.29	2.57	284.csv
195	65.0	69.2	0.0024	100	1631	0.059	1.14	2.16	353.csv
196	65.0	88.6	0.004	100	146	0.181	0.55	2.28	376.csv
197	65.0	88.6	0.0061	100	26	0.246	0.98	2.46	387.csv
198	80.0	103.3	0.006	100	19	0.289	1.51	2.47	325.csv
199	80.0	103.3	0.006	100	33	0.249	0.93	2.59	326.csv
200	80.0	103.2	0.004	100	375	0.128	0.72	2.72	308.csv
201	80.0	103.2	0.005	100	141	0.163	0.51	2.74	312.csv
202	80.0	103.2	0.005	100	124	0.148	0.94	3.11	317.csv
203	80.0	103.3	0.004	100	273	0.131	0.79	2.77	306.csv
204	80.0	103.2	0.0039	100	292	0.118	0.62	2.56	307.csv
205	80.0	103.2	0.004	100	362	0.097	1.36	3.08	309.csv
206	80.0	103.2	0.0049	100	805	0.129	0.46	4.4	311.csv
207	80.0	103.2	0.0049	100	204	0.2	0.2	2.31	313.csv
208	80.0	103.2	0.0049	100	196	0.122	0.56	2.92	315.csv
209	80.0	103.2	0.005	100	28	0.239	1.06	2.0	318.csv
210	80.0	103.2	0.005	100	60	0.193	0.93	2.46	320.csv
211	80.0	103.2	0.006	100	42	0.235	0.94	2.8	322.csv
212	80.0	103.2	0.0089	100	11	0.414	1.35	3.19	327.csv
213	80.0	103.2	0.0089	100	37	0.245	0.42	3.49	329.csv
214	80.0	103.2	0.0089	100	16	0.36	0.72	3.04	331.csv
215	80.0	103.2	0.0039	100	242	0.115	0.87	2.62	310.csv
216	80.0	103.2	0.0049	100	68	0.19	1.0	2.61	314.csv
217	80.0	103.2	0.005	100	85	0.178	1.23	3.0	319.csv
218	80.0	103.2	0.0059	100	42	0.25	0.93	2.84	323.csv
219	80.0	103.2	0.0059	100	62	0.19	1.05	3.27	324.csv
220	80.0	103.2	0.0089	100	19	0.31	0.87	3.46	328.csv
221	80.0	103.2	0.0089	100	17	0.306	0.69	3.08	332.csv
222	80.0	103.2	0.005	100	141	0.156	0.61	2.94	378.csv
223	80.0	103.2	0.0091	100	16	0.286	0.66	3.07	392.csv
224	80.0	103.2	0.0091	100	14	0.321	0.73	2.99	393.csv

No.	$D_{r,nom.}$	$D_{r,est.}$	γ (SA)	σ_v	$N_{\varepsilon_v=1\%}$	r_0	C_1	C_2	File
225	80.0	103.2	0.006	100	28	0.261	0.94	2.45	321.csv
226	80.0	103.2	0.0089	100	20	0.295	0.75	3.39	330.csv
227	30.0	63.2	0.003	250	138	0.145	0.92	1.64	10.csv
228	30.0	63.2	0.005	250	28	0.28	0.7	1.75	14.csv
229	30.0	63.2	0.0061	250	16	0.322	0.62	1.61	18.csv
230	30.0	63.2	0.006	250	25	0.272	0.47	1.78	19.csv
231	30.0	63.2	0.0091	250	9	0.414	0.44	1.69	22.csv
232	30.0	63.2	0.0026	250	745	0.059	0.98	1.85	26.csv
233	30.0	63.2	0.003	250	202	0.13	0.86	1.79	13.csv
234	30.0	63.2	0.005	250	29	0.254	0.57	1.62	15.csv
235	30.0	63.2	0.005	250	35	0.24	0.5	1.66	17.csv
236	30.0	63.2	0.006	250	17	0.294	0.56	1.6	21.csv
237	30.0	63.2	0.0091	250	8	0.401	0.52	1.58	23.csv
238	30.0	63.2	0.0091	250	8	0.42	0.52	1.73	25.csv
239	30.0	63.2	0.0026	250	638	0.059	1.04	1.83	27.csv
240	30.0	63.2	0.0026	250	2072	0.023	1.68	2.33	29.csv
241	30.0	63.2	0.0024	250	770	0.058	1.31	1.85	56.csv
242	30.0	63.2	0.004	250	42	0.245	0.71	1.53	7.csv
243	30.0	63.2	0.002	250	5579	0.075	1.07	2.02	3.csv
244	30.0	63.2	0.0024	250	1625	0.028	1.14	1.87	57.csv
245	30.0	63.2	0.0024	250	2673	0.047	1.41	2.23	58.csv
246	30.0	63.2	0.004	250	73	0.189	0.71	1.84	6.csv
247	30.0	63.2	0.004	250	56	0.209	0.74	1.73	8.csv
248	30.0	63.2	0.004	250	56	0.2	0.73	1.68	9.csv
249	30.0	63.2	0.003	250	161	0.122	0.9	1.67	11.csv
250	30.0	63.2	0.003	250	316	0.097	1.04	2.02	12.csv
251	30.0	63.2	0.006	250	15	0.285	0.48	1.38	20.csv
252	30.0	63.2	0.002	250	2147	0.046	1.34	1.76	54.csv
253	30.0	63.2	0.0025	250	389	0.121	0.48	1.43	55.csv
254	30.0	63.2	0.005	250	30	0.253	0.68	1.78	16.csv
255	30.0	63.2	0.0091	250	9	0.408	0.42	1.65	24.csv
256	30.0	63.2	0.0026	250	792	0.052	1.08	1.88	28.csv
257	45.0	77.3	0.003	250	218	0.118	0.94	1.83	30.csv
258	45.0	76.9	0.004	250	139	0.145	0.96	2.39	31.csv
259	45.0	76.9	0.003	250	534	0.068	0.91	2.1	35.csv
260	45.0	76.9	0.003	250	648	0.072	1.02	2.22	36.csv
261	45.0	76.9	0.0026	250	840	0.057	1.25	2.02	41.csv
262	45.0	76.9	0.0051	250	27	0.271	0.74	1.79	46.csv
263	45.0	76.9	0.002	250	908	0.075	1.19	1.62	50.csv
264	45.0	76.9	0.004	250	183	0.124	0.95	2.55	34.csv
265	45.0	76.9	0.0026	250	1150	0.06	1.13	2.07	39.csv
266	45.0	76.9	0.0024	250	1059	0.065	1.27	2.02	45.csv
267	45.0	76.9	0.0051	250	33	0.245	0.85	2.02	47.csv
268	45.0	76.9	0.0051	250	26	0.272	0.81	1.82	49.csv

No.	$D_{r,nom.}$	$D_{r,est.}$	γ (SA)	σ_v	$N_{\varepsilon_v=1\%}$	r_0	C_1	C_2	File
269	45.0	76.9	0.006	250	23	0.25	0.53	1.83	59.csv
270	45.0	76.9	0.006	250	25	0.338	0.77	2.23	61.csv
271	45.0	76.9	0.009	250	13	0.48	0.87	2.97	63.csv
272	45.0	76.9	0.006	250	29	0.347	0.46	1.97	60.csv
273	45.0	76.9	0.009	250	12	0.501	0.65	2.49	64.csv
274	45.0	76.9	0.009	250	12	0.534	0.48	2.13	65.csv
275	45.0	76.9	0.009	250	12	0.51	0.55	2.38	66.csv
276	45.0	76.9	0.004	250	126	0.145	0.74	2.17	32.csv
277	45.0	76.9	0.004	250	119	0.145	0.76	2.17	33.csv
278	45.0	76.9	0.003	250	238	0.114	0.91	1.82	37.csv
279	45.0	76.9	0.0024	250	2257	0.074	1.76	2.4	43.csv
280	45.0	76.9	0.0051	250	36	0.258	0.68	1.95	48.csv
281	45.0	76.9	0.006	250	28	0.316	0.86	2.45	62.csv
282	65.0	76.7	0.003	250	551	0.065	0.93	2.09	69.csv
283	65.0	96.0	0.004	250	282	0.104	1.41	3.08	74.csv
284	65.0	96.0	0.005	250	113	0.163	0.91	3.09	78.csv
285	65.0	96.0	0.006	250	40	0.212	0.53	2.29	82.csv
286	65.0	62.6	0.003	250	335	0.098	0.75	1.88	68.csv
287	65.0	96.0	0.004	250	184	0.126	0.63	2.28	73.csv
288	65.0	96.0	0.005	250	76	0.172	1.69	3.14	75.csv
289	65.0	96.0	0.005	250	116	0.139	0.72	2.79	77.csv
290	65.0	96.0	0.006	250	33	0.287	1.0	2.71	79.csv
291	65.0	96.0	0.006	250	51	0.191	0.65	2.7	81.csv
292	65.0	96.0	0.009	250	17	0.261	0.47	2.71	83.csv
293	65.0	96.0	0.009	250	14	0.498	0.53	2.52	85.csv
294	65.0	96.0	0.009	250	14	0.283	0.7	2.79	86.csv
295	65.0	96.0	0.0026	250	5061	0.025	1.18	2.33	87.csv
296	65.0	96.0	0.004	250	242	0.097	1.06	2.7	71.csv
297	65.0	96.0	0.004	250	213	0.113	0.72	2.37	72.csv
298	65.0	96.0	0.005	250	90	0.149	0.83	2.71	76.csv
299	65.0	96.0	0.006	250	39	0.214	0.8	2.63	80.csv
300	65.0	96.0	0.009	250	12	0.285	0.58	2.46	84.csv
301	80.0	110.5	0.006	250	77	0.167	1.25	3.68	100.csv
302	80.0	110.5	0.009	250	21	0.27	0.85	3.59	104.csv
303	80.0	110.5	0.003	250	1031	0.067	1.25	2.59	108.csv
304	80.0	110.5	0.003	250	1469	0.058	1.15	2.58	109.csv
305	80.0	110.5	0.004	250	378	0.089	1.27	3.02	93.csv
306	80.0	110.5	0.006	250	84	0.185	0.69	3.18	103.csv
307	80.0	110.5	0.009	250	15	0.28	0.8	3.15	105.csv
308	80.0	110.5	0.009	250	25	0.332	0.44	3.07	107.csv
309	80.0	110.5	0.003	250	1120	0.045	0.96	2.32	110.csv
310	80.0	110.5	0.004	250	213	0.101	1.51	2.87	92.csv
311	80.0	110.5	0.004	250	652	0.065	1.18	3.19	94.csv
312	80.0	110.5	0.005	250	83	0.168	0.84	2.73	96.csv

No.	$D_{r,nom.}$	$D_{r,est.}$	γ (SA)	σ_v	$N_{\varepsilon_v=1\%}$	r_0	C_1	C_2	File
313	80.0	110.5	0.005	250	191	0.111	0.73	3.11	97.csv
314	80.0	110.5	0.005	250	99	0.146	0.91	2.88	98.csv
315	80.0	110.5	0.005	250	255	0.099	0.84	3.36	99.csv
316	80.0	110.5	0.0026	250	2168	0.051	1.09	2.25	112.csv
317	80.0	110.5	0.003	250	1576	0.049	1.12	2.56	91.csv
318	80.0	110.5	0.006	250	60	1.0	0.47	2.49	101.csv
319	80.0	110.5	0.006	250	51	0.216	0.79	2.93	102.csv
320	80.0	110.5	0.009	250	22	0.37	0.48	3.0	106.csv
321	80.0	110.5	0.004	250	338	0.087	1.11	2.88	95.csv
322	30.0	51.6	0.002	50	271	0.18	0.62	1.17	1.csv
323	30.0	51.6	0.0026	50	92	0.23	0.6	1.12	10.csv
324	30.0	51.6	0.0026	50	73	1.0	0.82	1.14	11.csv
325	30.0	51.6	0.0026	50	183	0.216	0.39	1.29	13.csv
326	30.0	51.8	0.003	50	118	0.205	0.45	1.32	17.csv
327	30.0	51.8	0.003	50	55	0.257	0.45	0.94	18.csv
328	30.0	52.2	0.003	50	62	0.261	0.64	1.18	19.csv
329	30.0	51.9	0.003	50	84	0.25	0.43	1.16	20.csv
330	30.0	51.9	0.004	50	31	0.295	0.47	1.12	24.csv
331	30.0	51.6	0.0026	50	119	0.19	0.88	1.37	12.csv
332	30.0	51.6	0.0026	50	111	0.232	0.33	0.93	14.csv
333	30.0	51.9	0.003	50	139	0.202	0.43	1.41	16.csv
334	30.0	51.9	0.002	50	700	0.135	0.77	1.64	2.csv
335	30.0	52.1	0.004	50	61	0.25	0.43	1.48	21.csv
336	30.0	52.1	0.004	50	35	0.247	0.72	1.41	23.csv
337	30.0	52.2	0.004	50	27	0.289	0.77	1.28	25.csv
338	30.0	52.0	0.005	50	38	0.309	0.24	1.18	27.csv
339	30.0	54.1	0.005	50	28	0.241	0.25	1.08	30.csv
340	30.0	51.8	0.006	50	19	0.328	0.37	1.32	32.csv
341	30.0	52.2	0.006	50	16	0.371	0.35	1.07	34.csv
342	30.0	52.2	0.006	50	15	0.386	0.37	1.12	36.csv
343	30.0	52.6	0.009	50	8	0.5	0.34	1.19	38.csv
344	30.0	52.2	0.004	50	46	0.264	0.43	1.27	42.csv
345	30.0	51.6	0.0024	50	178	0.207	0.66	1.34	8.csv
346	30.0	52.1	0.005	50	34	0.309	0.3	1.26	28.csv
347	30.0	52.1	0.005	50	14	0.386	0.66	1.15	29.csv
348	30.0	51.9	0.002	50	152	0.194	0.78	1.03	3.csv
349	30.0	52.3	0.006	50	19	0.343	0.39	1.37	31.csv
350	30.0	52.3	0.006	50	14	0.389	0.45	1.17	35.csv
351	30.0	52.5	0.009	50	10	0.441	0.28	1.2	39.csv
352	30.0	51.9	0.002	50	219	0.176	0.79	1.18	4.csv
353	30.0	51.8	0.002	50	404	0.124	1.03	1.43	5.csv
354	30.0	51.6	0.0024	50	160	0.181	0.66	1.25	6.csv
355	30.0	51.6	0.0024	50	176	0.167	0.84	1.34	7.csv
356	30.0	51.6	0.0024	50	169	0.208	0.54	1.23	9.csv

No.	$D_{r,nom.}$	$D_{r,est.}$	γ (SA)	σ_v	$N_{\varepsilon_v=1\%}$	r_0	C_1	C_2	File
357	30.0	52.6	0.009	50	8	0.441	0.37	1.17	40.csv
358	30.0	52.7	0.009	50	9	0.494	0.33	1.22	41.csv
359	30.0	51.9	0.0026	50	228	0.182	0.5	1.48	15.csv
360	30.0	52.0	0.004	50	57	0.271	0.32	1.23	22.csv
361	30.0	51.8	0.005	50	30	0.274	0.39	1.39	26.csv
362	30.0	51.8	0.006	50	18	0.345	0.32	1.16	33.csv
363	30.0	52.2	0.009	50	8	0.444	0.33	1.11	37.csv
364	45.0	85.4	0.0026	50	140	0.163	0.92	1.42	109a.csv
365	45.0	66.1	0.002	50	497	0.145	0.57	1.4	46.csv
366	45.0	66.1	0.0024	50	197	0.192	0.54	1.29	50.csv
367	45.0	65.8	0.0026	50	148	0.216	0.46	1.21	55.csv
368	45.0	65.8	0.003	50	93	0.261	0.24	0.92	59.csv
369	45.0	66.1	0.002	50	476	0.159	0.59	1.44	43.csv
370	45.0	66.1	0.002	50	261	0.175	0.49	1.07	45.csv
371	45.0	66.1	0.002	50	603	0.127	0.68	1.48	47.csv
372	45.0	66.0	0.0024	50	254	0.177	0.45	1.31	49.csv
373	45.0	66.0	0.0024	50	374	0.151	0.47	1.47	51.csv
374	45.0	65.8	0.0026	50	152	0.203	0.42	1.18	54.csv
375	45.0	65.8	0.003	50	135	0.197	0.38	1.29	56.csv
376	45.0	65.9	0.003	50	136	0.223	0.32	1.24	58.csv
377	45.0	65.9	0.003	50	77	0.247	0.74	1.44	60.csv
378	45.0	65.8	0.004	50	69	0.263	0.26	1.24	62.csv
379	45.0	65.8	0.004	50	76	0.269	0.26	1.3	64.csv
380	45.0	65.8	0.005	50	29	0.323	0.29	1.1	66.csv
381	45.0	65.9	0.005	50	37	0.302	0.28	1.27	68.csv
382	45.0	65.9	0.0049	50	33	0.28	0.4	1.4	70.csv
383	45.0	65.8	0.006	50	26	0.295	0.28	1.39	72.csv
384	45.0	65.8	0.006	50	18	0.33	0.41	1.34	74.csv
385	45.0	65.9	0.009	50	10	0.414	0.32	1.35	76.csv
386	45.0	65.8	0.009	50	10	0.78	0.25	1.14	78.csv
387	45.0	65.8	0.004	50	52	0.27	0.38	1.3	63.csv
388	45.0	65.9	0.005	50	25	0.306	0.42	1.27	67.csv
389	45.0	65.8	0.006	50	21	0.353	0.32	1.32	71.csv
390	45.0	65.8	0.006	50	24	0.323	0.33	1.45	75.csv
391	45.0	65.8	0.009	50	8	0.456	0.43	1.46	79.csv
392	45.0	66.0	0.002	50	403	0.152	0.48	1.21	44.csv
393	45.0	66.2	0.0024	50	232	0.18	0.67	1.44	48.csv
394	45.0	66.2	0.0026	50	176	0.18	0.7	1.5	52.csv
395	45.0	65.8	0.003	50	82	0.234	0.35	1.03	57.csv
396	45.0	65.9	0.004	50	67	0.274	0.22	1.14	61.csv
397	45.0	65.9	0.004	50	59	0.284	0.54	1.73	65.csv
398	45.0	65.9	0.005	50	38	0.299	0.29	1.32	69.csv
399	45.0	65.9	0.006	50	18	0.33	0.43	1.4	73.csv
400	45.0	65.8	0.009	50	8	0.43	0.46	1.45	77.csv

No.	$D_{r,nom.}$	$D_{r,est.}$	γ (SA)	σ_v	$N_{\varepsilon_v=1\%}$	r_0	C_1	C_2	File
401	65.0	85.2	0.004	50	121	0.181	0.6	2.1	100.csv
402	65.0	85.2	0.005	50	48	0.266	0.33	1.63	104.csv
403	65.0	85.2	0.006	50	30	0.305	0.43	1.91	108.csv
404	65.0	85.2	0.009	50	15	0.399	0.31	1.98	112.csv
405	65.0	85.2	0.002	50	1091	0.118	0.8	1.83	81.csv
406	65.0	85.2	0.002	50	1919	0.095	0.61	1.78	83.csv
407	65.0	85.2	0.005	50	40	0.29	0.33	1.49	101.csv
408	65.0	85.2	0.005	50	46	0.259	0.41	1.75	103.csv
409	65.0	85.2	0.006	50	29	0.327	0.34	1.68	105.csv
410	65.0	85.3	0.006	50	24	0.33	0.35	1.48	107.csv
411	65.0	85.2	0.009	50	15	0.405	0.32	1.95	109.csv
412	65.0	85.2	0.009	50	15	0.411	0.34	2.03	111.csv
413	65.0	85.2	0.002	50	1043	0.113	0.48	1.48	82.csv
414	65.0	85.2	0.0024	50	222	0.172	0.54	1.32	84.csv
415	65.0	85.2	0.0024	50	807	0.11	0.59	1.86	86.csv
416	65.0	85.2	0.0026	50	434	0.141	0.79	1.97	91.csv
417	65.0	85.2	0.003	50	127	0.199	0.57	1.49	93.csv
418	65.0	85.2	0.003	50	229	0.174	0.63	1.92	95.csv
419	65.0	85.2	0.004	50	94	0.232	0.37	1.71	97.csv
420	65.0	85.2	0.0024	50	640	0.106	0.74	1.82	87.csv
421	65.0	85.2	0.0024	50	546	0.134	0.56	1.69	88.csv
422	65.0	85.2	0.0026	50	296	0.17	0.51	1.63	92.csv
423	65.0	85.3	0.003	50	119	0.202	0.52	1.39	96-.csv
424	65.0	85.2	0.0026	50	193	0.202	0.47	1.39	89.csv
425	65.0	85.2	0.0026	50	303	0.146	0.73	1.73	90.csv
426	65.0	85.2	0.005	50	46	0.265	0.44	1.82	102.csv
427	65.0	85.2	0.006	50	27	0.321	0.39	1.73	106.csv
428	65.0	85.2	0.0091	50	12	0.474	0.37	1.89	110.csv
429	65.0	85.2	0.0024	50	320	0.164	0.49	1.48	85.csv
430	65.0	85.2	0.003	50	159	0.171	0.62	1.64	94.csv
431	65.0	85.2	0.004	50	74	0.222	0.55	1.79	98.csv
432	65.0	85.2	0.004	50	122	0.21	0.44	2.05	99.csv
433	80.0	99.7	0.002	50	1538	0.103	0.88	1.98	114.csv
434	80.0	99.7	0.0024	50	1041	0.101	0.8	2.07	119.csv
435	80.0	99.7	0.0026	50	698	0.099	1.1	2.17	124.csv
436	80.0	99.7	0.003	50	294	0.137	0.66	1.82	128.csv
437	80.0	99.7	0.004	50	172	0.141	0.65	2.33	132.csv
438	80.0	99.7	0.0024	50	1777	0.064	1.18	2.23	118.csv
439	80.0	99.7	0.0026	50	1297	0.063	0.95	2.19	123.csv
440	80.0	99.7	0.003	50	317	0.119	0.56	1.73	125.csv
441	80.0	99.7	0.003	50	199	0.158	0.56	1.65	127.csv
442	80.0	99.7	0.004	50	126	0.158	0.8	2.27	129.csv
443	80.0	99.7	0.004	50	129	0.152	0.95	2.41	131.csv
444	80.0	99.7	0.005	50	90	0.169	0.81	2.73	133.csv

No.	$D_{r,nom.}$	$D_{r,est.}$	γ (SA)	σ_v	$N_{\varepsilon_v=1\%}$	r_0	C_1	C_2	File
445	80.0	99.7	0.005	50	51	0.218	0.62	2.13	135.csv
446	80.0	99.7	0.006	50	43	0.212	0.87	2.81	137.csv
447	80.0	99.7	0.0061	50	40	0.226	0.8	2.72	139.csv
448	80.0	99.7	0.0091	50	15	0.376	0.56	2.78	141.csv
449	80.0	99.7	0.009	50	14	0.388	0.47	2.38	143.csv
450	80.0	99.7	0.005	50	69	0.208	0.89	2.64	136.csv
451	80.0	99.7	0.006	50	37	0.231	0.61	2.39	140.csv
452	80.0	99.7	0.009	50	15	0.36	0.44	2.36	144.csv
453	80.0	99.7	0.0024	50	2159	0.066	1.46	2.42	117.csv
454	80.0	99.7	0.0026	50	870	0.08	1.09	2.18	121.csv
455	80.0	99.7	0.0026	50	901	0.099	1.04	2.33	122.csv
456	80.0	99.7	0.003	50	498	0.112	0.76	2.27	126.csv
457	80.0	99.7	0.004	50	145	0.167	0.61	2.23	130.csv
458	80.0	99.7	0.005	50	91	0.164	0.89	2.81	134.csv
459	80.0	99.7	0.006	50	47	0.224	0.58	2.57	138.csv
460	80.0	99.7	0.009	50	16	0.354	0.39	2.4	142.csv
461	30.0	55.2	0.0024	100	96	0.189	0.87	1.13	185.csv
462	30.0	55.2	0.0024	100	96	0.189	0.87	1.13	185.csv
463	30.0	55.2	0.0024	100	96	0.189	0.87	1.13	185.csv
464	30.0	55.2	0.0024	100	96	0.189	0.87	1.13	185.csv
465	30.0	55.2	0.0024	100	96	0.189	0.87	1.13	185.csv
466	30.0	55.2	0.0024	100	96	0.189	0.87	1.13	185.csv

Table B.2: Cycles to Various Levels of Volumetric Strain

No.	$N_{\varepsilon_v=0.1\%}$	$N_{0.2\%}$	$N_{0.3\%}$	$N_{0.4\%}$	$N_{0.5\%}$	$N_{0.6\%}$	$N_{0.7\%}$	$N_{0.8\%}$	$N_{0.9\%}$	$N_{1\%}$
1	2	4	10	20	39	73	140	263	485	862
2	1	2	5	8	11	21	27	40	59	82
3	1	1	2	2	3	4	5	5	6	8
4	1	3	5	10	17	29	50	88	148	262
5	2	4	10	20	37	65	111	186	302	482
6	2	5	11	21	43	79	137	237	362	526
7	2	4	11	24	50	105	208	417	775	1287
8	1	2	3	5	8	11	14	17	24	30
9	1	2	4	5	8	11	17	24	30	43
10	1	2	2	3	4	5	6	8	8	11
11	1	1	1	2	2	3	4	5	6	8
12	2	4	8	15	27	48	83	138	215	326
13	1	3	5	8	12	19	29	43	64	96
14	1	3	4	7	11	17	25	37	52	73
15	1	2	4	7	11	16	24	35	50	71
16	1	3	5	8	13	22	34	51	77	115

No.	$N_{\varepsilon_v=0.1\%}$	$N_{0.2\%}$	$N_{0.3\%}$	$N_{0.4\%}$	$N_{0.5\%}$	$N_{0.6\%}$	$N_{0.7\%}$	$N_{0.8\%}$	$N_{0.9\%}$	$N_{1\%}$
17	1	3	5	9	14	23	35	53	78	114
18	1	2	3	4	5	7	9	12	17	21
19	1	2	3	4	6	8	11	15	19	24
20	1	1	2	3	4	5	7	9	11	15
21	1	1	2	3	4	6	8	11	16	19
22	2	4	8	16	28	48	79	125	202	314
23	1	3	4	8	12	20	32	50	77	118
24	1	3	5	10	17	31	48	77	119	183
25	1	3	6	12	22	36	62	94	149	236
26	1	2	3	5	7	9	13	20	27	34
27	1	2	2	3	4	5	7	9	12	16
28	1	3	4	6	9	13	17	23	31	40
29	2	4	5	8	11	15	21	24	34	40
30	1	1	2	4	5	7	9	12	17	21
31	1	3	6	12	24	47	91	163	295	485
32	1	2	3	5	8	14	20	30	43	59
33	2	5	12	26	59	133	290	572	1023	1722
34	1	2	5	10	20	36	62	105	184	311
35	2	4	13	29	65	145	295	556	965	1532
36	1	4	10	23	52	108	208	398	675	1097
37	1	1	2	3	5	7	10	14	18	24
38	1	1	1	2	3	4	5	7	8	10
39	1	1	2	2	3	4	6	8	10	13
40	2	4	10	20	37	69	121	192	311	485
41	1	3	8	16	33	62	114	208	369	630
42	1	2	2	4	5	7	9	12	17	21
43	1	1	2	2	3	4	5	5	6	8
44	2	3	6	12	22	36	60	96	150	233
45	1	3	5	9	14	22	33	49	71	103
46	1	2	2	4	5	7	9	12	18	21
47	1	1	2	3	5	7	9	12	16	21
48	1	1	2	3	4	5	6	8	11	14
49	2	5	10	22	42	82	147	264	438	701
50	2	3	5	9	15	24	40	65	104	168
51	2	3	6	11	19	34	61	98	167	272
52	1	2	4	6	10	15	24	34	47	64
53	1	2	3	4	5	6	8	11	15	19
54	1	2	2	3	4	6	7	9	11	14
55	1	1	2	3	4	5	7	9	11	15
56	1	2	3	5	8	11	15	21	28	37
57	1	1	2	3	5	7	9	14	18	24
58	1	2	4	9	16	26	45	78	129	216
59	1	2	3	5	9	13	19	27	34	47
60	1	2	3	5	7	9	12	15	20	25

No.	$N_{\varepsilon_v=0.1\%}$	$N_{0.2\%}$	$N_{0.3\%}$	$N_{0.4\%}$	$N_{0.5\%}$	$N_{0.6\%}$	$N_{0.7\%}$	$N_{0.8\%}$	$N_{0.9\%}$	$N_{1\%}$
61	1	2	2	4	6	8	11	15	20	27
62	1	3	6	11	20	34	56	91	145	220
63	1	3	7	14	24	43	72	119	191	301
64	1	2	2	4	4	5	8	8	11	15
65	2	4	10	20	43	82	162	295	527	862
66	2	4	10	20	36	66	120	201	320	523
67	1	2	3	4	6	8	11	17	21	27
68	1	1	1	1	2	3	3	4	5	7
69	2	4	8	16	28	47	77	125	193	291
70	1	3	5	8	13	21	32	48	70	101
71	1	3	5	8	12	18	27	38	53	73
72	1	2	3	4	5	7	10	13	17	21
73	1	2	3	4	5	7	9	11	16	18
74	1	3	5	9	14	21	32	46	67	99
75	1	2	2	3	5	6	8	10	13	17
76	2	7	14	30	63	131	262	497	885	1451
77	1	2	4	8	14	24	40	63	104	166
78	1	1	2	3	4	6	8	11	14	18
79	1	3	7	15	29	58	108	192	335	541
80	1	1	2	4	6	8	12	16	21	28
81	1	3	8	16	29	55	100	177	316	525
82	1	2	2	4	5	7	10	14	17	21
83	2	4	10	18	33	58	95	156	250	395
84	2	4	8	17	32	61	110	203	353	618
85	1	3	4	7	10	15	23	32	45	62
86	2	3	5	8	13	21	33	49	76	119
87	1	2	4	5	7	10	14	19	30	34
88	2	4	10	32	71	147	268	469	788	1254
89	2	4	8	15	26	46	79	134	231	387
90	1	2	4	7	12	18	26	39	59	84
91	2	3	6	10	17	27	41	62	91	127
92	2	4	7	14	31	51	94	177	325	614
93	2	3	5	8	12	17	28	35	49	70
94	1	3	4	7	11	17	31	40	61	90
95	1	3	4	7	10	15	27	34	48	68
96	1	3	4	8	13	20	33	50	78	120
97	1	2	4	7	10	16	28	36	60	75
98	1	2	4	6	9	13	23	32	40	62
99	1	2	3	4	6	8	11	15	23	31
100	1	2	3	4	6	8	11	15	23	31
101	1	2	4	5	8	12	18	25	33	42
102	1	2	3	4	5	6	9	11	16	21
103	1	2	3	5	6	9	13	21	29	33
104	1	2	2	3	4	5	5	6	8	10

No.	$N_{\varepsilon_v=0.1\%}$	$N_{0.2\%}$	$N_{0.3\%}$	$N_{0.4\%}$	$N_{0.5\%}$	$N_{0.6\%}$	$N_{0.7\%}$	$N_{0.8\%}$	$N_{0.9\%}$	$N_{1\%}$
105	1	1	2	3	4	5	5	7	8	10
106	1	2	2	3	4	5	6	8	11	15
107	2	4	9	20	41	84	166	323	615	1081
108	2	3	5	8	14	22	37	61	98	152
109	2	4	8	16	33	65	127	246	470	849
110	2	3	7	14	31	54	101	190	353	616
111	2	4	7	16	34	69	149	324	615	1140
112	2	4	8	17	34	68	136	266	528	965
113	2	4	7	14	33	62	120	236	441	791
114	1	3	5	10	17	30	45	70	108	159
115	1	3	5	8	14	28	40	65	105	178
116	1	2	4	5	7	11	16	28	35	57
117	1	3	5	8	14	27	37	62	92	149
118	1	2	3	5	7	10	15	22	31	42
119	1	2	3	4	5	7	10	13	18	24
120	1	2	2	3	5	6	8	10	13	17
121	1	2	3	4	5	6	8	11	14	20
122	1	2	3	4	5	8	11	14	20	26
123	1	2	2	3	5	6	8	11	15	19
124	1	1	2	3	4	5	7	9	12	15
125	2	4	9	20	42	91	185	382	703	1227
126	2	4	9	18	37	73	149	295	589	1140
127	1	3	6	12	23	39	67	120	207	325
128	1	3	6	13	27	48	90	148	265	440
129	1	3	5	8	13	21	33	48	71	105
130	1	3	4	7	12	21	33	45	68	100
131	1	2	3	4	6	9	13	19	25	33
132	1	2	2	3	4	5	6	7	9	11
133	2	4	10	23	53	118	264	555	1080	1954
134	1	3	5	9	14	23	37	61	90	131
135	1	3	4	7	12	18	31	42	63	94
136	1	2	3	5	7	10	15	21	33	43
137	1	3	5	9	16	31	49	100	178	241
138	1	2	3	4	5	8	10	14	20	26
139	1	2	3	5	7	10	14	21	29	36
140	1	2	3	4	5	7	9	12	16	21
141	1	2	2	3	4	5	7	9	12	16
142	2	5	12	32	74	177	381	702	1168	1839
143	2	3	6	11	21	37	66	120	216	388
144	2	3	7	13	25	47	91	160	295	499
145	1	3	5	8	15	31	46	79	137	240
146	2	3	6	12	26	43	77	129	237	382
147	2	3	5	10	17	33	53	75	121	179
148	1	2	3	5	7	9	13	19	25	34

No.	$N_{\varepsilon_v=0.1\%}$	$N_{0.2\%}$	$N_{0.3\%}$	$N_{0.4\%}$	$N_{0.5\%}$	$N_{0.6\%}$	$N_{0.7\%}$	$N_{0.8\%}$	$N_{0.9\%}$	$N_{1\%}$
149	1	3	6	10	18	32	50	81	145	255
150	2	5	14	37	107	295	702	1460	2597	4200
151	1	4	7	16	34	78	172	353	673	1227
152	2	5	14	36	91	236	528	1140	2102	3500
153	2	4	8	17	37	80	163	324	587	1052
154	2	3	7	14	32	60	119	206	381	672
155	2	3	6	12	25	48	91	189	382	702
156	2	3	7	14	32	63	134	295	586	1140
157	2	3	6	13	26	46	91	159	295	557
158	1	4	7	15	33	62	120	236	440	790
159	2	4	8	18	35	68	128	258	439	730
160	1	3	7	14	30	57	99	178	324	557
161	1	2	3	5	8	13	20	29	43	62
162	1	2	4	6	11	18	30	42	63	96
163	1	3	5	10	20	34	59	93	150	235
164	1	2	4	5	8	12	22	33	43	65
165	1	2	4	6	9	14	23	34	53	78
166	1	2	3	4	5	8	11	19	30	35
167	1	1	2	4	6	9	13	19	27	37
168	1	1	2	4	6	10	15	22	32	47
169	1	2	3	4	5	5	7	10	15	20
170	1	2	3	4	6	9	13	18	24	33
171	2	4	7	14	27	54	113	258	585	1285
172	2	4	9	20	44	93	207	470	936	1723
173	1	3	5	10	19	37	72	149	267	528
174	1	2	4	6	10	16	28	38	62	91
175	1	2	4	6	10	16	28	39	62	91
176	1	2	4	5	8	13	22	33	39	62
177	1	2	2	3	5	5	7	10	14	20
178	1	3	7	16	33	62	120	208	354	586
179	1	4	8	18	34	69	137	237	382	616
180	1	2	4	5	9	14	26	34	59	88
181	1	2	2	3	4	5	7	9	12	18
182	1	1	2	3	4	5	6	9	11	16
183	2	4	8	16	32	62	127	269	556	1025
184	1	3	5	10	23	34	62	99	178	265
185	1	2	3	4	6	8	12	20	29	34
186	1	2	3	4	5	6	8	10	16	22
187	1	2	3	5	5	7	10	16	22	31
188	2	4	8	18	35	69	135	266	470	791
189	1	2	4	6	9	14	23	33	53	78
190	1	3	5	9	18	32	54	88	134	194
191	1	2	4	6	11	19	31	42	64	96
192	1	2	3	5	8	12	18	27	39	56

No.	$N_{\varepsilon_v=0.1\%}$	$N_{0.2\%}$	$N_{0.3\%}$	$N_{0.4\%}$	$N_{0.5\%}$	$N_{0.6\%}$	$N_{0.7\%}$	$N_{0.8\%}$	$N_{0.9\%}$	$N_{1\%}$
193	1	2	3	4	5	6	9	13	18	24
194	1	2	3	5	6	8	10	16	21	29
195	2	4	8	18	37	78	177	381	847	1634
196	1	3	5	9	16	31	44	67	101	149
197	1	2	3	4	5	7	10	13	23	33
198	1	2	2	4	5	5	8	11	18	21
199	1	2	3	4	6	8	11	19	24	34
200	1	3	5	11	21	37	69	124	217	375
201	1	2	5	8	14	24	37	63	92	143
202	1	2	4	6	11	21	34	50	79	124
203	1	3	5	11	19	34	59	104	172	278
204	1	2	5	10	19	34	63	108	179	295
205	1	3	4	8	14	27	53	104	198	362
206	1	3	9	21	50	98	195	324	530	820
207	3	5	8	14	24	37	59	88	137	204
208	1	2	4	7	13	24	43	76	124	211
209	1	2	2	4	5	8	11	14	21	33
210	1	2	3	5	8	11	18	27	40	63
211	1	2	4	5	6	8	18	21	34	50
212	1	2	2	2	4	5	6	8	8	11
213	1	2	3	5	6	8	15	21	31	37
214	1	2	2	4	5	6	8	11	15	19
215	1	2	4	8	16	29	52	91	155	262
216	1	2	4	5	8	11	21	30	43	69
217	1	2	3	5	8	14	21	34	53	85
218	1	2	4	5	8	11	18	21	34	50
219	1	2	4	5	8	11	19	27	40	63
220	1	2	2	4	5	6	8	11	18	21
221	1	2	2	4	5	5	8	11	15	21
222	1	3	4	7	13	25	36	63	92	150
223	1	1	2	3	4	5	7	9	13	18
224	1	2	2	3	4	5	6	8	12	16
225	1	2	2	4	5	8	11	17	21	31
226	1	2	2	4	5	6	8	11	18	21
227	1	2	4	8	14	21	34	56	88	143
228	1	2	3	4	6	8	11	18	21	31
229	1	2	2	3	4	5	8	11	14	17
230	1	2	2	4	5	8	11	15	21	27
231	1	1	2	2	3	4	5	6	8	9
232	1	3	6	12	23	43	86	185	379	772
233	1	2	5	8	15	27	44	73	124	205
234	1	2	2	4	5	8	11	17	21	32
235	1	2	3	4	7	10	14	20	27	36
236	1	1	2	3	4	6	8	11	14	18

No.	$N_{\varepsilon_v=0.1\%}$	$N_{0.2\%}$	$N_{0.3\%}$	$N_{0.4\%}$	$N_{0.5\%}$	$N_{0.6\%}$	$N_{0.7\%}$	$N_{0.8\%}$	$N_{0.9\%}$	$N_{1\%}$
237	1	1	2	2	3	4	5	5	6	8
238	1	1	2	2	3	4	5	6	8	8
239	1	3	6	11	21	40	76	153	321	656
240	1	2	5	11	26	56	133	329	870	2127
241	2	3	7	11	21	40	79	166	353	785
242	1	2	3	5	8	11	17	24	34	43
243	2	5	11	29	72	214	672	1822	3616	5610
244	1	3	6	13	26	52	113	268	696	1714
245	1	3	7	17	39	87	216	548	1311	2738
246	1	2	3	5	10	15	24	34	50	76
247	1	2	3	5	8	14	18	27	40	56
248	1	2	3	5	8	13	18	27	40	59
249	1	2	4	7	12	20	34	59	101	172
250	1	3	5	10	17	30	53	95	176	321
251	1	1	2	3	4	5	7	9	12	16
252	1	3	7	15	34	82	201	482	1078	2147
253	2	5	8	16	26	46	76	131	227	395
254	1	2	2	4	6	8	12	18	24	32
255	1	1	2	2	3	4	5	6	8	10
256	1	3	6	11	21	40	82	176	395	817
257	1	2	4	8	14	24	43	76	134	221
258	1	2	4	7	11	21	34	53	92	150
259	1	3	5	11	20	37	72	137	276	540
260	1	3	5	11	21	42	88	176	350	685
261	1	3	5	11	21	43	88	189	408	859
262	1	2	3	4	5	8	11	14	21	27
263	2	3	7	14	30	59	123	250	494	920
264	1	2	4	8	14	21	37	63	111	192
265	1	3	7	13	27	59	121	279	598	1210
266	1	3	6	13	27	57	124	266	556	1065
267	1	2	3	4	6	8	11	18	24	34
268	1	2	2	4	5	8	11	14	21	27
269	1	1	2	3	5	7	9	12	17	24
270	1	1	2	3	5	7	10	14	18	25
271	1	1	1	2	3	4	6	8	10	13
272	1	2	3	4	6	9	12	17	23	30
273	1	1	2	2	3	4	5	7	9	12
274	1	1	2	2	3	4	6	8	9	12
275	1	1	2	2	3	4	6	8	9	12
276	1	2	4	7	11	18	31	50	82	134
277	1	2	4	6	11	18	30	49	78	123
278	1	2	5	8	17	27	47	85	147	250
279	1	3	7	18	42	100	261	638	1266	2432
280	1	2	3	5	7	11	14	21	27	37

No.	$N_{\varepsilon_v=0.1\%}$	$N_{0.2\%}$	$N_{0.3\%}$	$N_{0.4\%}$	$N_{0.5\%}$	$N_{0.6\%}$	$N_{0.7\%}$	$N_{0.8\%}$	$N_{0.9\%}$	$N_{1\%}$
281	1	1	2	3	5	7	11	15	21	29
282	1	3	5	11	18	34	66	134	279	569
283	1	2	4	7	13	26	49	87	168	306
284	1	2	4	6	10	18	29	47	76	121
285	1	2	3	4	6	9	15	21	30	40
286	1	3	5	11	18	31	56	105	192	353
287	1	2	4	8	14	24	42	69	118	203
288	1	1	2	4	7	11	19	29	49	81
289	1	2	3	6	10	17	29	47	76	132
290	1	1	2	4	5	8	12	16	26	36
291	1	2	3	5	7	11	17	24	37	53
292	1	1	2	3	4	5	8	10	14	18
293	1	1	2	2	3	5	6	8	12	15
294	1	1	2	2	3	5	6	8	12	15
295	1	3	7	17	34	87	250	770	2271	-1
296	1	2	4	7	13	24	42	78	145	261
297	1	2	4	8	14	24	43	76	134	237
298	1	2	3	5	8	15	21	34	60	92
299	1	2	2	4	5	8	15	18	33	40
300	1	1	2	2	3	4	5	7	10	13
301	1	2	3	5	8	11	18	34	47	79
302	1	2	2	3	5	5	8	14	18	21
303	1	3	7	14	27	53	114	250	527	1065
304	1	3	7	14	27	60	143	337	775	1548
305	1	2	5	8	16	30	59	111	221	427
306	1	2	3	5	8	15	24	34	61	92
307	1	1	2	2	3	5	7	8	14	18
308	1	2	2	3	5	6	9	15	18	28
309	1	3	6	11	24	46	98	217	498	1123
310	1	2	4	7	11	21	37	66	121	221
311	1	2	4	10	20	36	74	158	342	680
312	1	2	3	5	8	16	21	34	53	89
313	1	2	4	7	13	21	39	66	121	216
314	1	2	3	5	8	16	24	37	60	105
315	1	2	4	7	14	24	47	82	150	279
316	2	3	7	17	34	76	192	469	1094	-1
317	1	3	7	14	30	63	147	337	804	-1
318	1	2	3	5	8	11	18	31	40	63
319	1	2	3	5	8	11	18	24	34	53
320	1	1	2	3	5	5	8	14	18	27
321	1	2	4	8	14	27	50	95	192	369
322	2	4	10	17	30	52	81	123	184	271
323	2	3	5	9	14	22	33	47	67	93
324	1	2	4	7	11	17	25	36	52	73

No.	$N_{\varepsilon_v=0.1\%}$	$N_{0.2\%}$	$N_{0.3\%}$	$N_{0.4\%}$	$N_{0.5\%}$	$N_{0.6\%}$	$N_{0.7\%}$	$N_{0.8\%}$	$N_{0.9\%}$	$N_{1\%}$
325	2	4	8	15	26	42	63	92	131	183
326	1	3	6	10	16	28	39	62	89	120
327	1	2	4	7	10	15	22	32	41	58
328	2	3	4	7	11	16	29	35	46	64
329	2	3	5	9	15	22	34	45	63	91
330	1	2	3	5	7	10	14	19	25	33
331	2	3	5	8	14	23	36	55	82	120
332	1	3	7	12	19	29	42	61	84	115
333	1	3	6	11	19	33	47	69	98	144
334	2	5	13	26	49	94	165	278	452	700
335	1	3	4	7	11	16	26	34	46	63
336	1	2	3	5	6	9	13	20	28	35
337	1	2	3	4	6	8	11	15	23	30
338	1	3	4	6	9	13	18	24	32	38
339	1	1	2	4	5	8	11	16	23	32
340	1	2	3	4	5	7	9	12	15	19
341	1	2	3	4	5	6	8	10	13	16
342	1	2	3	4	5	6	8	10	13	16
343	1	2	2	3	3	4	5	6	7	8
344	1	2	4	6	9	13	20	28	35	46
345	2	4	7	13	22	38	57	87	131	186
346	1	3	4	5	8	11	15	22	29	34
347	1	2	2	3	5	6	7	9	12	14
348	2	3	6	11	19	31	49	72	105	152
349	1	2	3	4	5	7	9	12	16	20
350	1	2	3	4	5	6	7	9	11	14
351	1	2	2	3	4	5	5	7	8	10
352	2	3	7	14	24	40	64	99	149	219
353	2	4	8	15	27	49	90	154	247	404
354	2	4	6	11	19	33	48	73	108	160
355	2	3	6	10	17	28	47	75	116	180
356	2	4	7	13	22	37	55	82	121	169
357	1	1	2	2	3	4	5	5	6	8
358	1	2	2	3	4	5	6	6	7	9
359	2	4	8	15	26	44	70	109	160	237
360	1	3	4	7	11	16	25	33	43	62
361	1	2	3	4	6	9	13	18	24	32
362	1	2	3	4	5	7	9	12	15	18
363	1	1	2	2	3	4	5	6	7	8
364	1	3	5	8	14	23	37	61	92	148
365	2	5	11	23	43	76	128	208	326	500
366	2	4	8	14	24	40	62	93	137	197
367	2	4	7	13	21	33	49	72	104	150
368	1	3	7	12	18	27	39	53	71	93

No.	$N_{\varepsilon_v=0.1\%}$	$N_{0.2\%}$	$N_{0.3\%}$	$N_{0.4\%}$	$N_{0.5\%}$	$N_{0.6\%}$	$N_{0.7\%}$	$N_{0.8\%}$	$N_{0.9\%}$	$N_{1\%}$
369	2	5	12	24	45	79	131	208	324	476
370	2	4	9	18	31	49	78	121	179	267
371	2	5	11	22	43	78	138	232	383	615
372	2	4	9	16	31	49	77	120	178	266
373	1	4	9	18	34	60	101	162	250	374
374	2	4	7	13	21	33	51	74	108	152
375	1	3	6	11	19	30	45	66	95	140
376	2	4	7	13	21	34	49	70	99	140
377	1	2	4	8	13	19	28	40	57	78
378	1	3	5	9	13	21	30	39	55	69
379	2	3	6	9	15	24	34	43	62	76
380	1	2	4	5	7	10	14	18	24	31
381	1	2	4	6	9	12	17	23	31	37
382	1	2	3	5	7	10	14	20	27	34
383	1	2	3	4	6	9	12	15	20	26
384	1	2	2	4	5	6	9	11	14	18
385	1	1	2	3	4	5	6	7	8	10
386	1	1	2	3	4	5	6	7	8	10
387	1	3	5	7	10	15	23	32	39	56
388	1	2	3	4	6	8	11	15	20	26
389	1	2	3	5	6	8	10	13	18	23
390	1	2	3	5	6	8	11	15	20	25
391	1	1	2	2	3	4	5	6	7	8
392	2	5	10	20	38	65	107	172	269	403
393	2	4	8	14	25	43	69	106	161	232
394	2	3	6	11	20	34	54	82	122	179
395	1	3	5	9	14	21	32	43	62	85
396	1	3	5	9	14	21	30	39	53	67
397	1	2	4	7	11	17	24	34	45	60
398	1	2	4	6	9	12	17	24	32	38
399	1	2	2	4	5	7	9	11	14	19
400	1	1	2	2	3	4	5	6	6	8
401	2	3	5	8	13	26	34	62	91	122
402	1	2	4	6	9	14	21	29	36	50
403	2	3	4	5	7	9	13	19	27	34
404	1	2	3	4	5	6	7	9	13	16
405	2	6	14	30	63	122	225	412	683	1091
406	2	6	16	36	78	167	350	653	1169	1927
407	1	3	4	6	9	12	18	25	33	40
408	1	3	4	6	8	12	19	28	34	50
409	2	3	4	5	7	10	13	20	27	34
410	1	2	4	5	6	8	11	14	21	27
411	1	2	3	4	5	6	7	9	13	16
412	1	2	3	4	5	6	7	9	13	17

No.	$N_{\varepsilon_v=0.1\%}$	$N_{0.2\%}$	$N_{0.3\%}$	$N_{0.4\%}$	$N_{0.5\%}$	$N_{0.6\%}$	$N_{0.7\%}$	$N_{0.8\%}$	$N_{0.9\%}$	$N_{1\%}$
413	2	6	14	31	61	117	209	382	673	1052
414	2	4	8	14	24	40	64	98	150	222
415	2	5	11	24	47	93	163	296	500	821
416	2	4	8	16	33	62	99	167	276	442
417	2	4	6	10	16	31	40	62	91	127
418	2	3	7	13	23	40	66	102	155	237
419	2	3	5	9	15	26	35	54	69	94
420	2	4	9	19	36	68	123	217	384	646
421	2	5	10	20	39	73	131	230	362	559
422	2	4	9	17	34	51	83	130	200	296
423	2	3	6	10	16	28	38	62	91	121
424	2	4	8	15	25	41	64	94	135	193
425	2	4	7	14	26	45	77	125	209	303
426	2	3	4	6	9	12	20	29	35	51
427	1	2	4	4	6	9	12	19	25	33
428	1	2	3	4	5	5	6	8	10	14
429	2	4	9	18	34	56	92	142	216	325
430	2	3	6	10	17	32	44	69	105	159
431	1	3	5	7	11	17	29	37	59	74
432	2	4	5	10	16	31	40	63	92	122
433	2	6	14	33	72	150	295	537	936	1548
434	2	5	10	23	49	98	188	354	645	1053
435	2	4	7	15	32	61	119	221	411	702
436	2	3	6	11	20	37	63	104	179	295
437	1	3	5	8	14	28	40	64	119	178
438	2	4	8	19	41	91	207	440	906	1781
439	2	4	8	17	34	71	149	323	673	1314
440	2	3	6	11	21	37	64	119	185	324
441	2	3	6	11	19	33	59	91	128	208
442	1	3	5	7	12	23	34	61	92	126
443	1	3	4	7	11	23	33	61	91	129
444	1	2	4	5	9	14	31	37	63	92
445	1	2	4	5	8	11	19	29	35	60
446	1	2	3	5	6	9	15	23	33	43
447	1	2	3	4	6	9	13	24	33	40
448	1	2	3	4	5	5	7	9	14	19
449	1	2	3	4	5	5	7	8	11	17
450	2	3	4	5	8	13	27	34	62	69
451	1	2	3	4	6	9	13	21	31	37
452	1	2	3	4	5	6	7	9	13	18
453	2	4	9	19	45	105	246	558	1198	2160
454	2	3	7	14	31	60	122	251	470	876
455	2	4	8	18	37	76	149	280	527	905
456	2	4	7	15	32	61	95	179	296	499

No.	$N_{\varepsilon_v=0.1\%}$	$N_{0.2\%}$	$N_{0.3\%}$	$N_{0.4\%}$	$N_{0.5\%}$	$N_{0.6\%}$	$N_{0.7\%}$	$N_{0.8\%}$	$N_{0.9\%}$	$N_{1\%}$
457	2	3	5	8	14	29	38	63	94	150
458	1	3	4	5	9	14	30	36	62	91
459	1	2	4	5	7	10	18	27	34	55
460	1	2	3	4	5	6	8	10	14	18
461	1	3	5	8	12	19	29	43	64	96
462	1	3	5	8	12	19	29	43	64	96
463	1	3	5	8	12	19	29	43	64	96
464	1	3	5	8	12	19	29	43	64	96
465	1	3	5	8	12	19	29	43	64	96
466	1	3	5	8	12	19	29	43	64	96

B.2 Irregular-Amplitude Results

Two types of irregular loading tests were performed: 1. Variable-amplitude sinusoidal loading, and 2. Earthquake-type loadings. Table B.3 summarizes the results of these tests. D_r is the relative density of the test, γ_{max} is the maximum shear strain of the test, and $\varepsilon_{v,fin.}$ is the volumetric strain at the end of the test.

Table B.3: Summary of Irregular-loading Seismic Compression CSS Tests

No.	D_r	σ_v	γ_{max}	$\varepsilon_{v,fin.}$	Duration	Type	File
	(%)	(kPa)	(%)	(%)	(s)		
1	55	100	0.344	0.282	26.3	Earthquake	139.csv
2	55	100	0.336	0.332	257.8	Earthquake	140.csv
3	55	100	0.338	0.375	257.8	Earthquake	141.csv
4	55	100	0.341	0.394	257.9	Earthquake	143.csv
5	55	100	0.338	0.392	257.8	Earthquake	144.csv
6	55	100	0.346	0.300	27.6	Earthquake	145.csv
7	55	100	0.341	0.238	26.1	Earthquake	146.csv
8	55	100	0.365	0.262	43.5	Earthquake	162.csv
9	55	100	0.367	0.319	43.4	Earthquake	163.csv
10	55	100	0.361	0.271	50.8	Earthquake	164.csv
11	55	100	0.348	0.218	65.8	Earthquake	178.csv
12	55	100	0.346	0.245	65.8	Earthquake	179.csv
13	55	100	0.350	0.282	69.9	Earthquake	180.csv
14	55	100	0.345	0.250	65.7	Earthquake	181.csv
15	55	100	0.346	0.424	258.1	Earthquake	142.csv
16	55	100	0.341	0.258	26.3	Earthquake	147.csv
17	55	100	0.352	0.272	65.8	Earthquake	177.csv
18	55	100	0.949	0.773	105.6	Variable-Sine	100.csv
19	55	100	0.914	0.633	105.6	Variable-Sine	101.csv
20	55	100	0.956	0.745	105.6	Variable-Sine	102.csv
21	55	100	0.944	0.696	105.6	Variable-Sine	105.csv
22	55	100	0.953	0.581	105.6	Variable-Sine	106.csv
23	55	100	0.938	0.544	105.6	Variable-Sine	107.csv
24	55	100	0.947	0.643	105.6	Variable-Sine	108.csv
25	55	100	0.929	0.670	105.6	Variable-Sine	110.csv
26	55	100	0.924	0.705	105.6	Variable-Sine	111.csv
27	55	100	0.952	0.789	105.6	Variable-Sine	112.csv
28	55	100	0.922	0.664	65.7	Variable-Sine	113.csv
29	55	100	0.924	0.724	65.7	Variable-Sine	115.csv
30	55	100	0.924	0.673	65.7	Variable-Sine	116.csv
31	55	100	0.959	0.690	65.7	Variable-Sine	117.csv
32	55	100	0.947	0.694	65.7	Variable-Sine	118.csv
33	55	100	0.952	0.967	65.7	Variable-Sine	120.csv

No.	D_r	σ_v	γ_{max}	$\varepsilon_{v,fin.}$	Duration	Type	File
34	55	100	0.874	0.619	105.4	Variable-Sine	97.csv
35	55	100	0.922	0.707	105.6	Variable-Sine	98.csv
36	55	100	0.901	0.556	105.6	Variable-Sine	99.csv
37	55	100	0.944	0.771	105.6	Variable-Sine	104.csv
38	55	100	0.914	0.723	105.6	Variable-Sine	109.csv
39	55	100	0.919	0.805	65.7	Variable-Sine	114.csv
40	55	100	0.959	0.549	65.7	Variable-Sine	119.csv

The following figures give the results of the tests found in Table B.3.

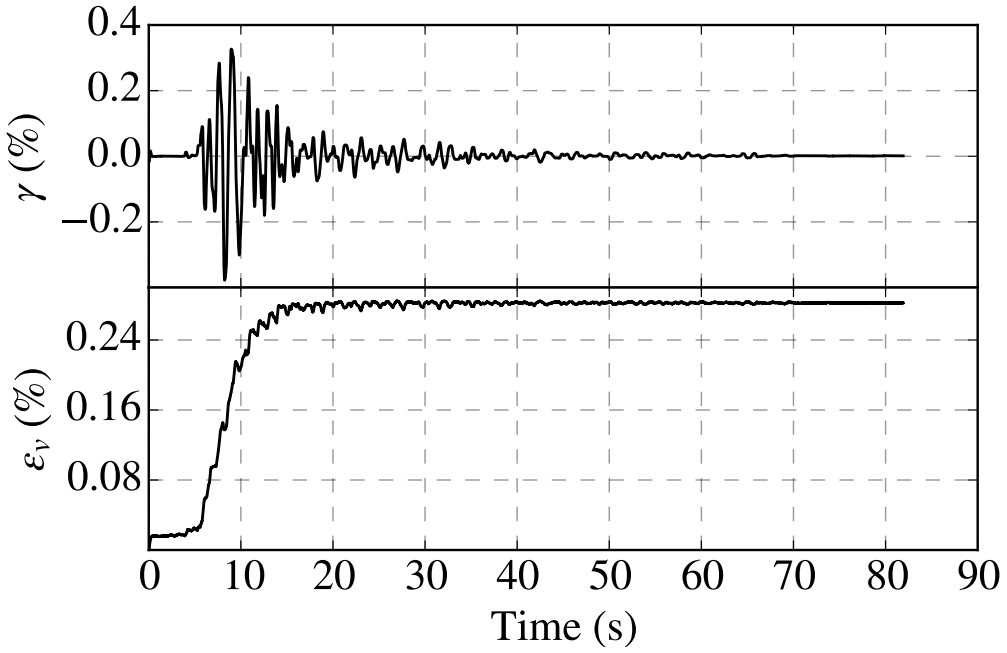


Figure B.1: Test results for irregular-loading seismic compression test *139.csv*.

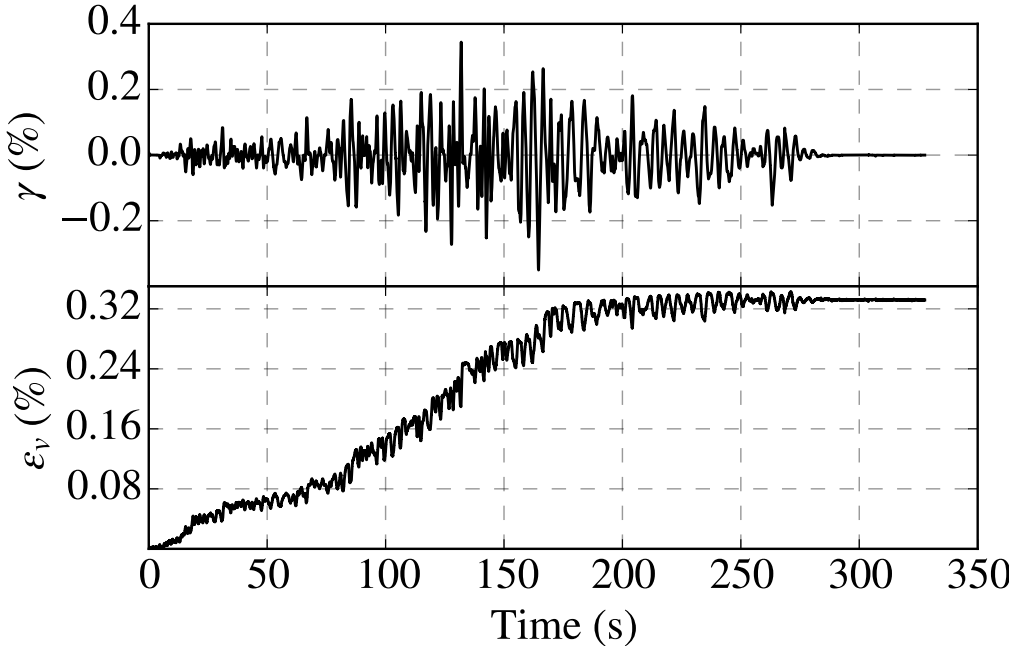


Figure B.2: Test results for irregular-loading seismic compression test *140.csv*.

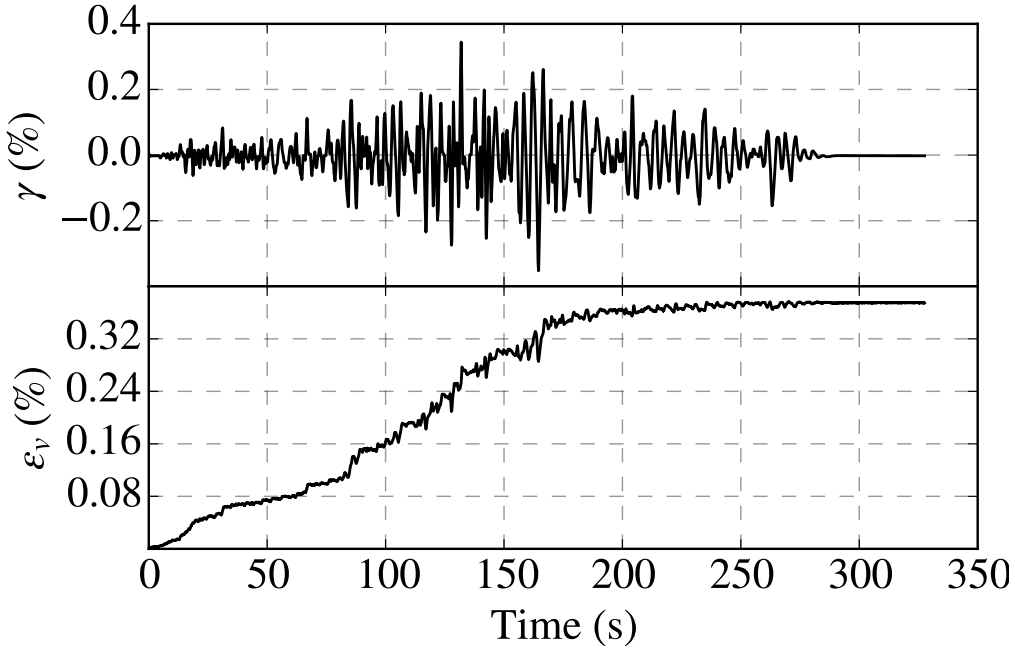


Figure B.3: Test results for irregular-loading seismic compression test *141.csv*.

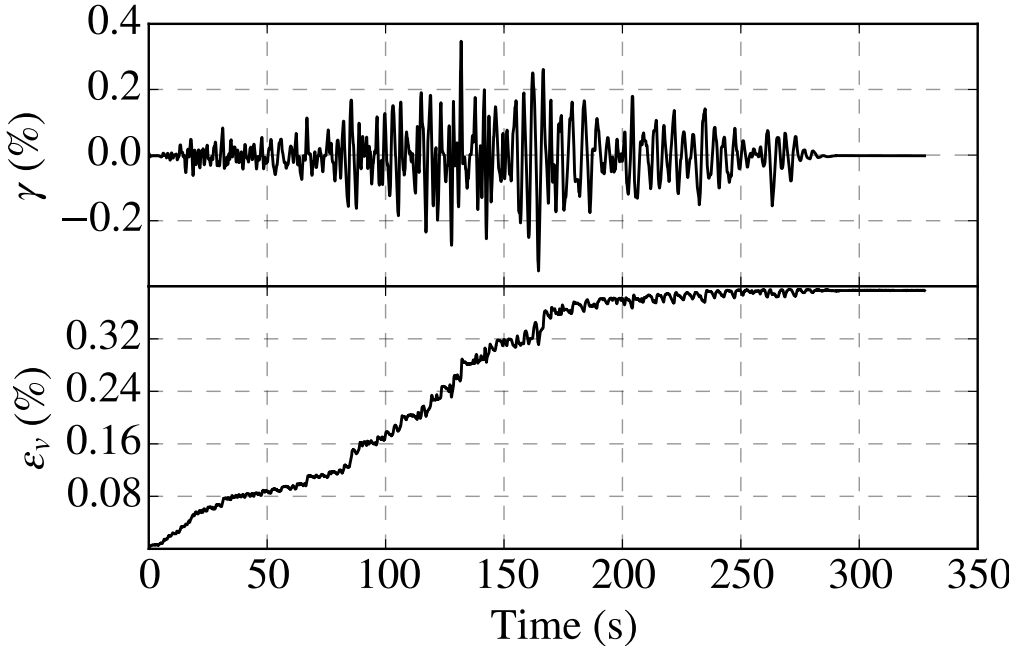


Figure B.4: Test results for irregular-loading seismic compression test *143.csv*.

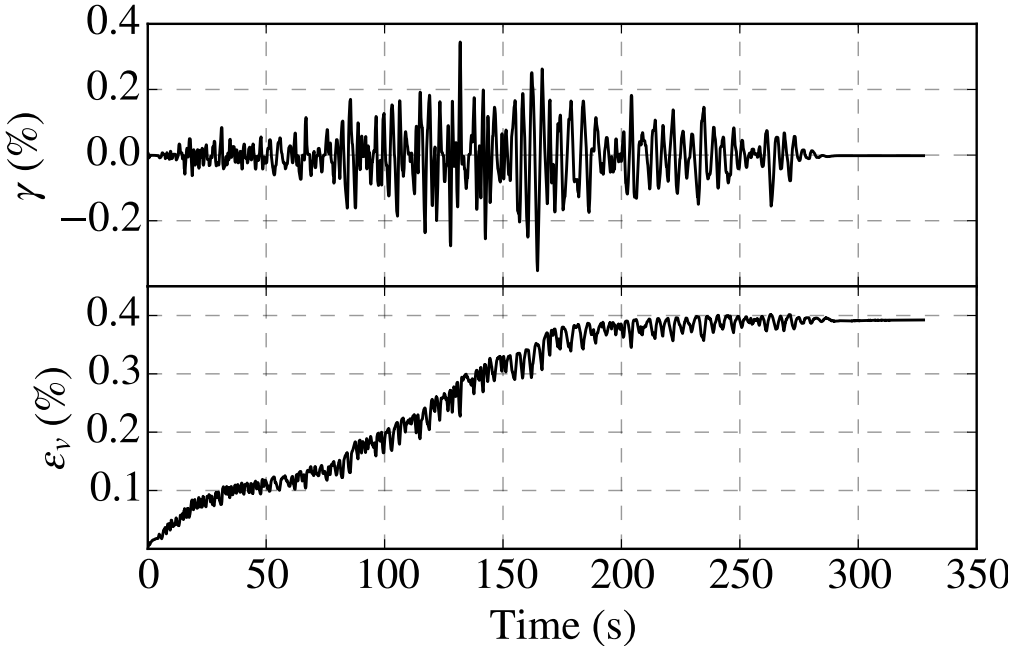


Figure B.5: Test results for irregular-loading seismic compression test *144.csv*.

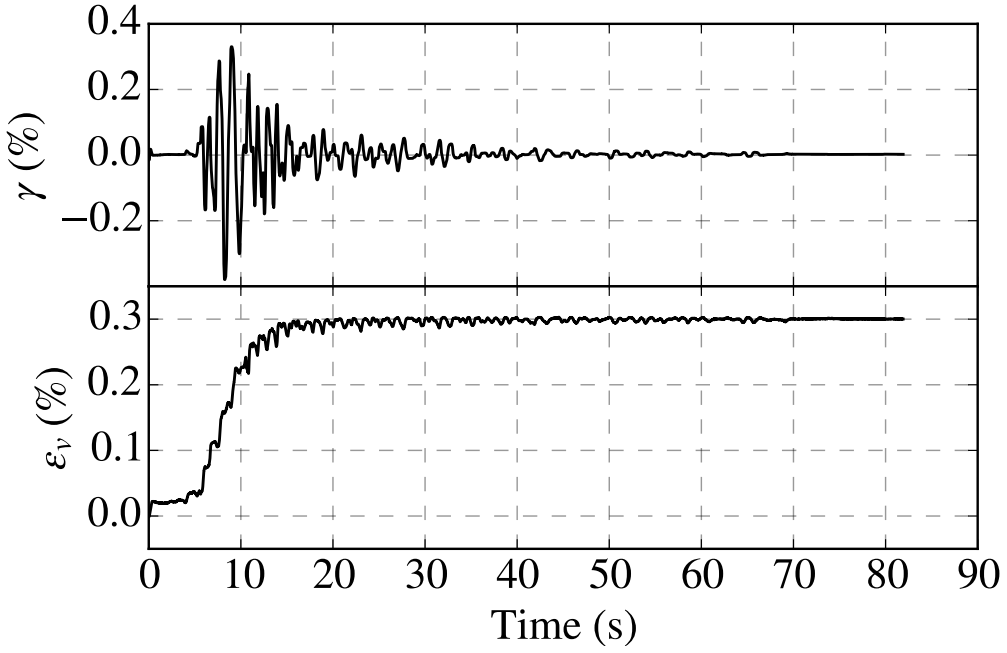


Figure B.6: Test results for irregular-loading seismic compression test *145.csv*.

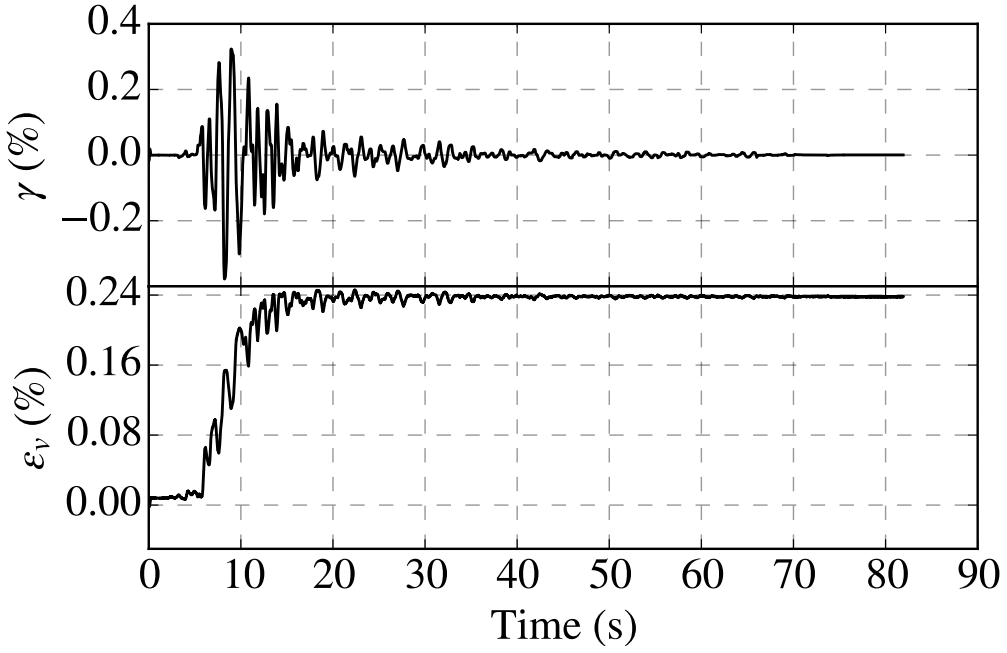


Figure B.7: Test results for irregular-loading seismic compression test *146.csv*.

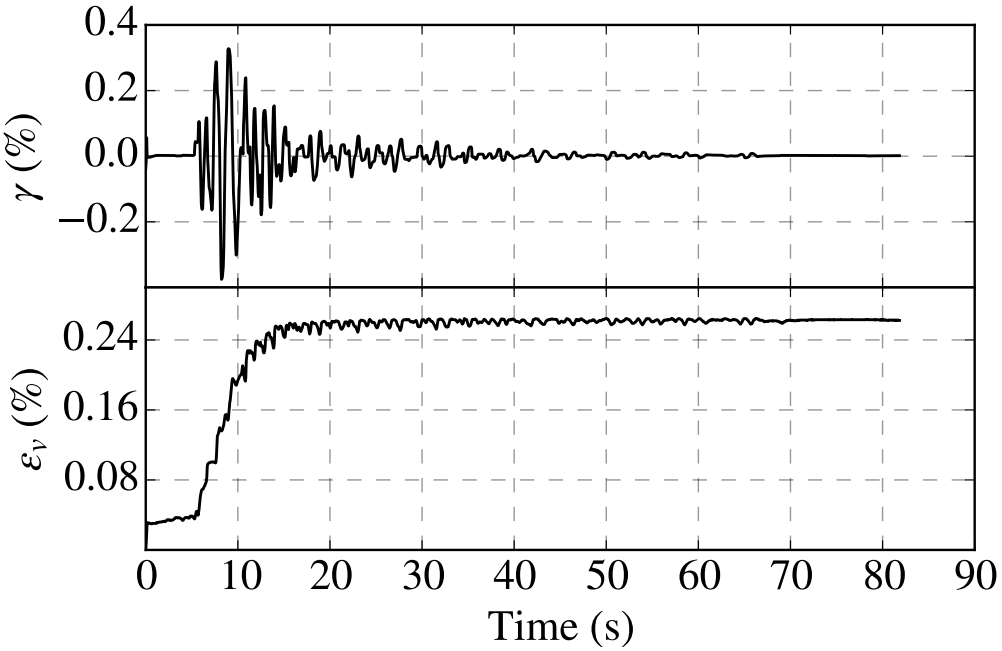


Figure B.8: Test results for irregular-loading seismic compression test *162.csv*.

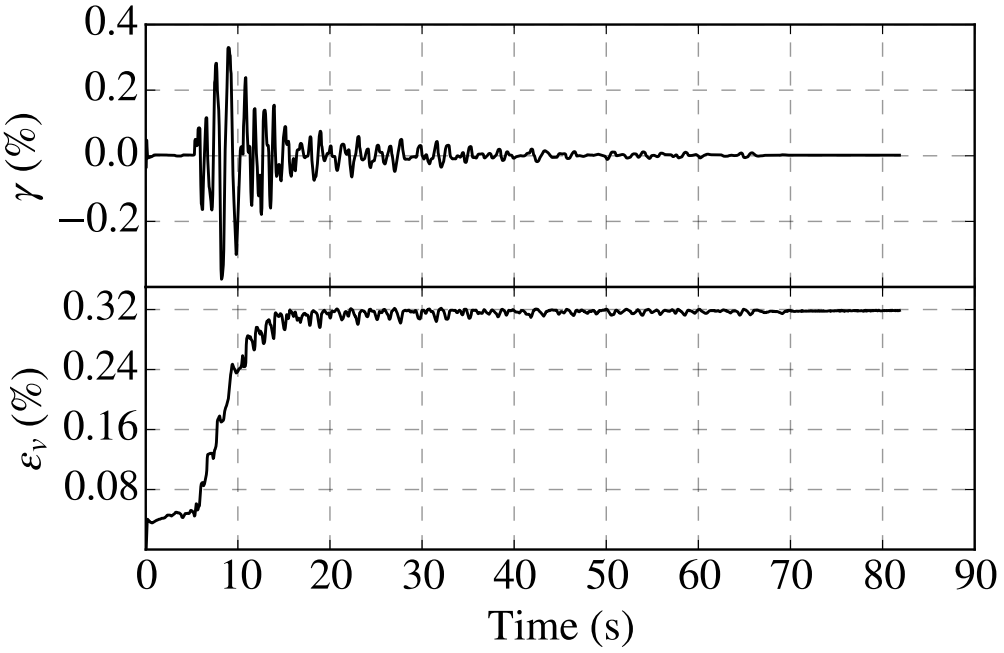


Figure B.9: Test results for irregular-loading seismic compression test *163.csv*.

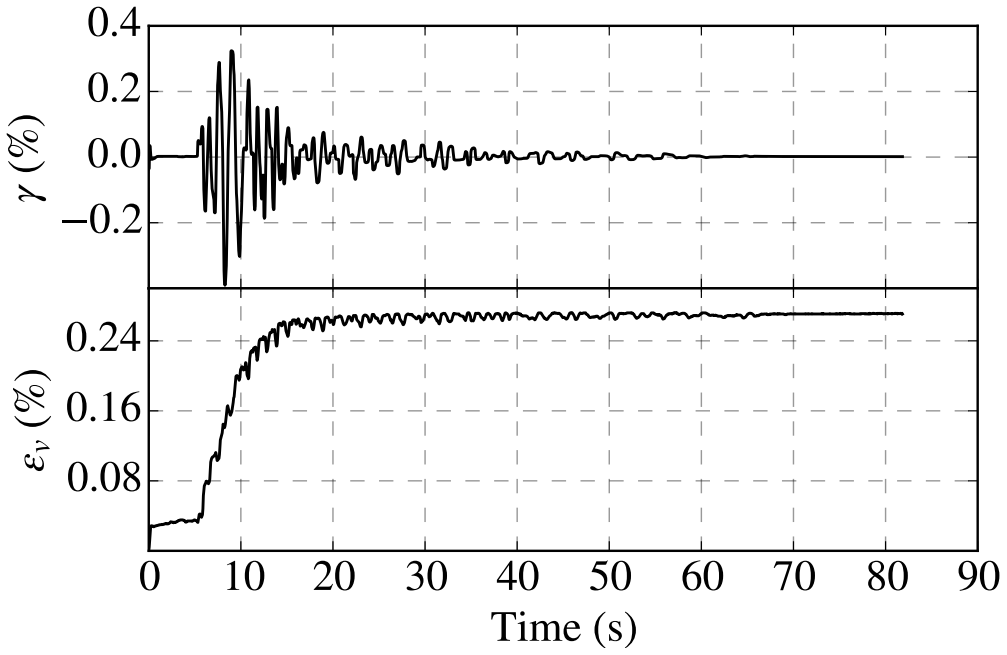


Figure B.10: Test results for irregular-loading seismic compression test *164.csv*.

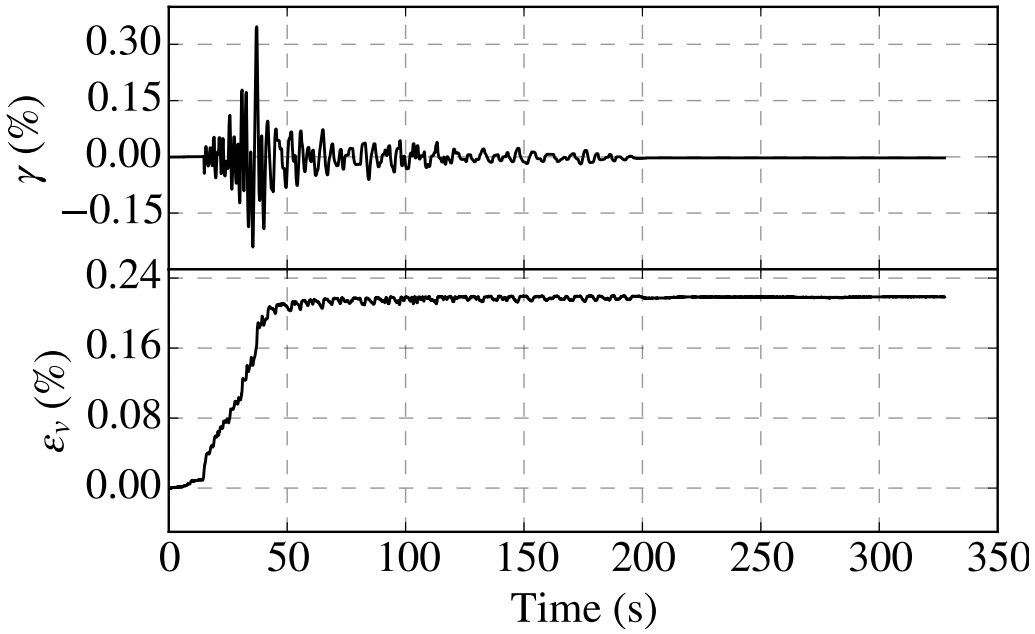


Figure B.11: Test results for irregular-loading seismic compression test *178.csv*.

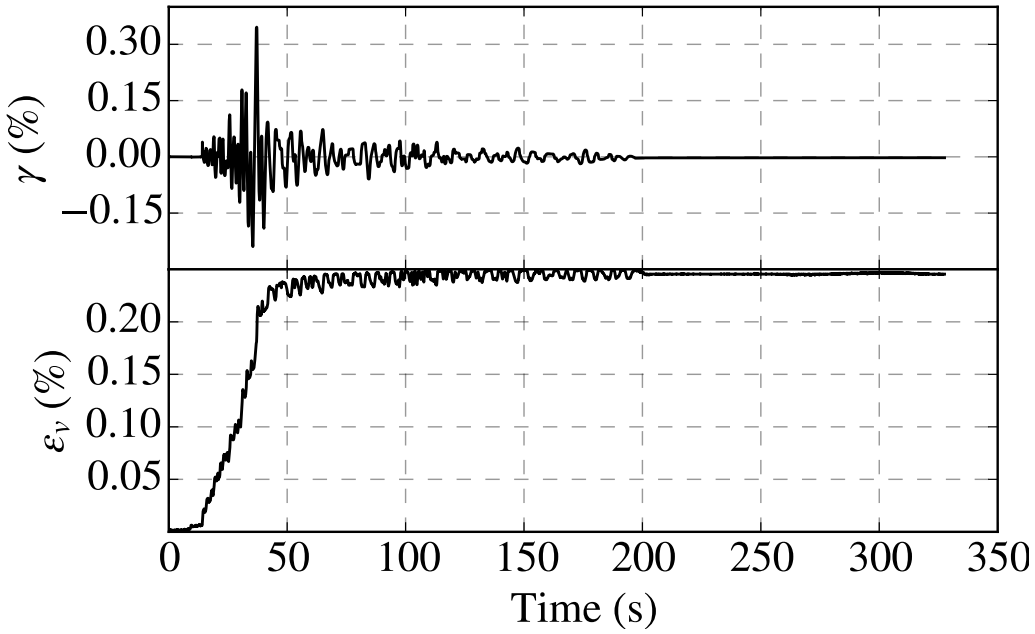


Figure B.12: Test results for irregular-loading seismic compression test *179.csv*.

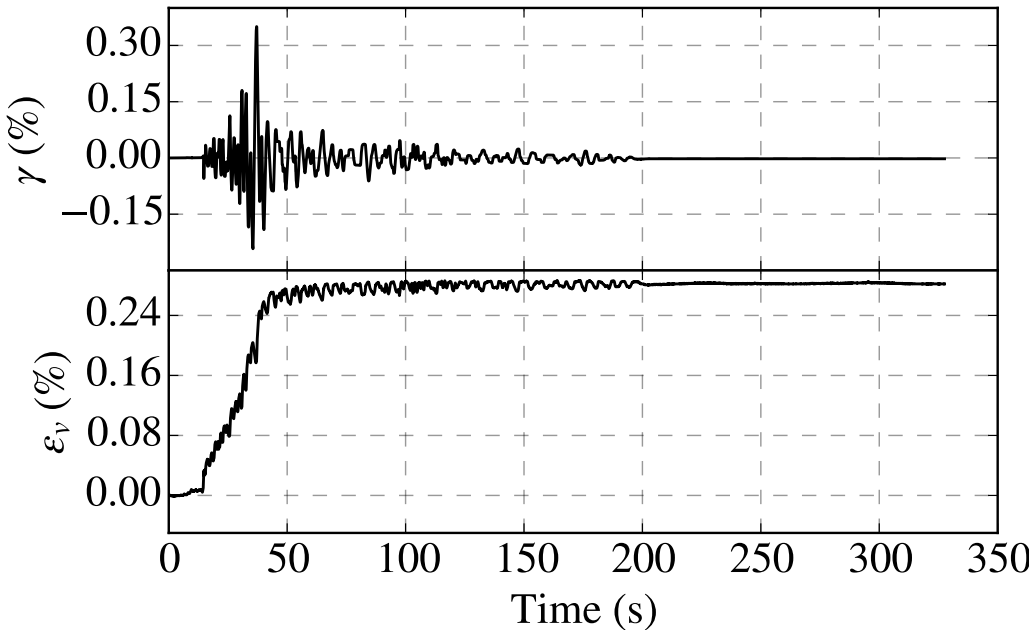


Figure B.13: Test results for irregular-loading seismic compression test *180.csv*.

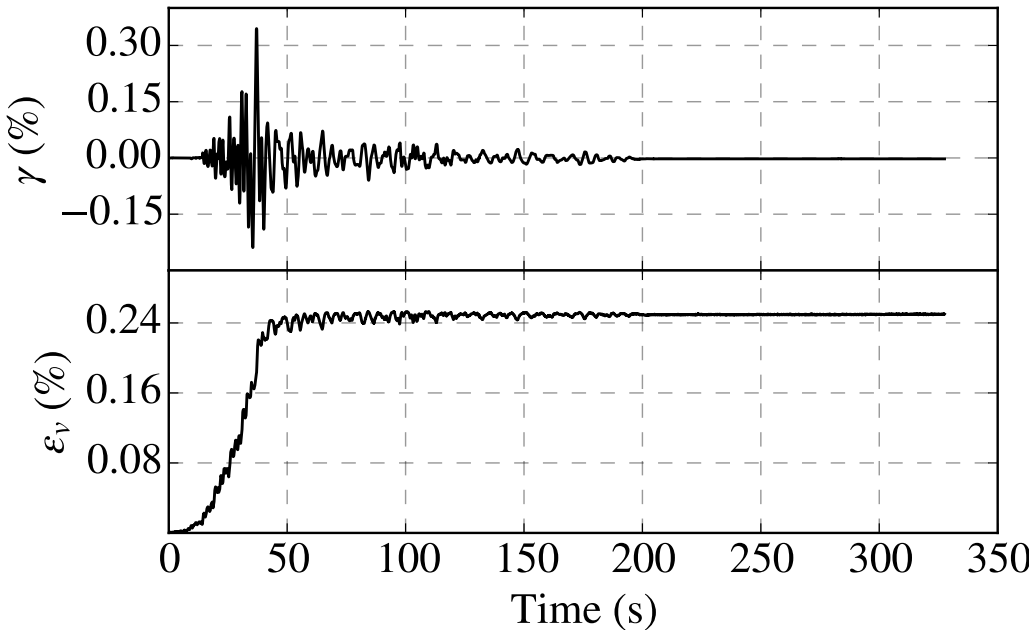


Figure B.14: Test results for irregular-loading seismic compression test *181.csv*.

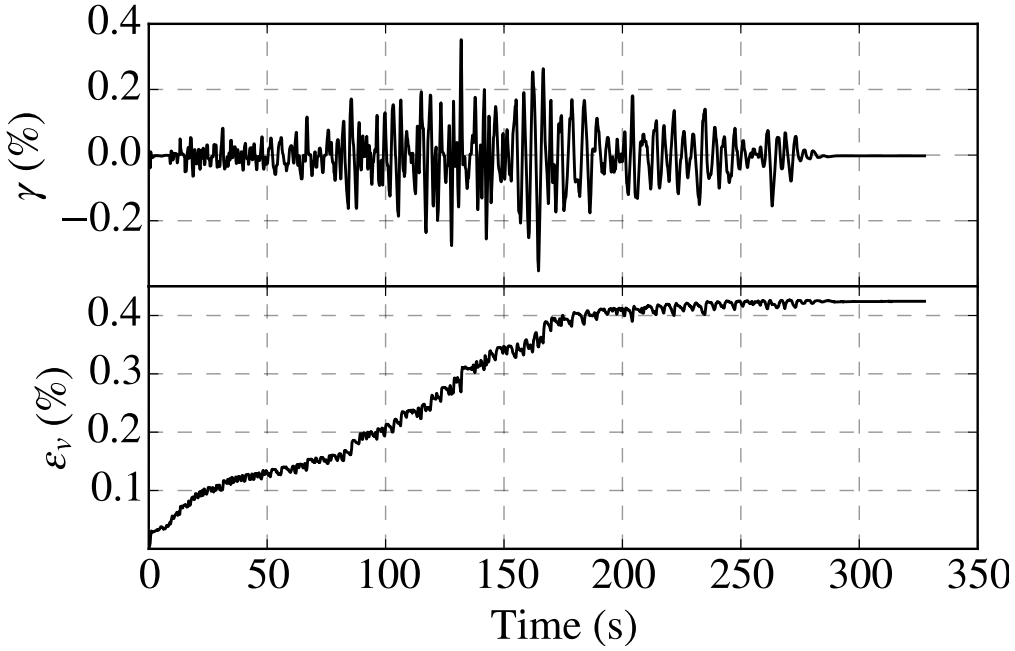


Figure B.15: Test results for irregular-loading seismic compression test *142.csv*.

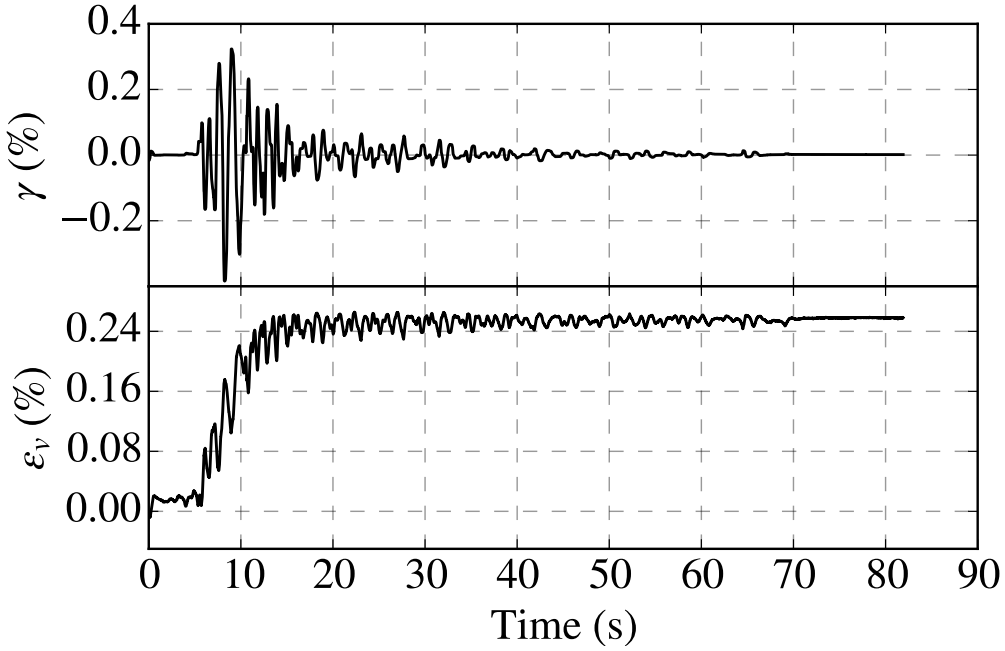


Figure B.16: Test results for irregular-loading seismic compression test *147.csv*.

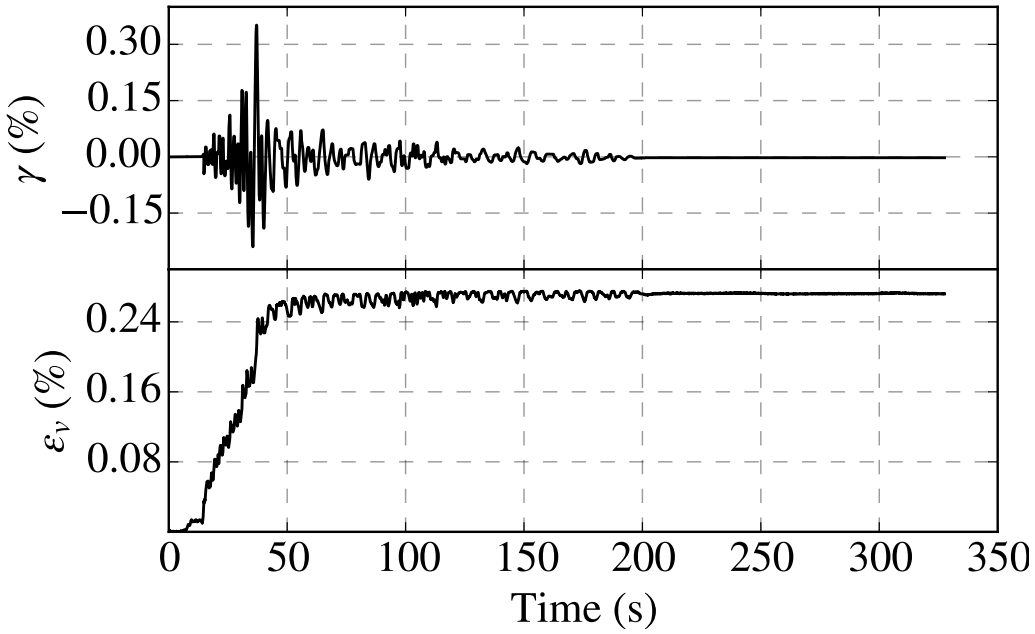


Figure B.17: Test results for irregular-loading seismic compression test *177.csv*.

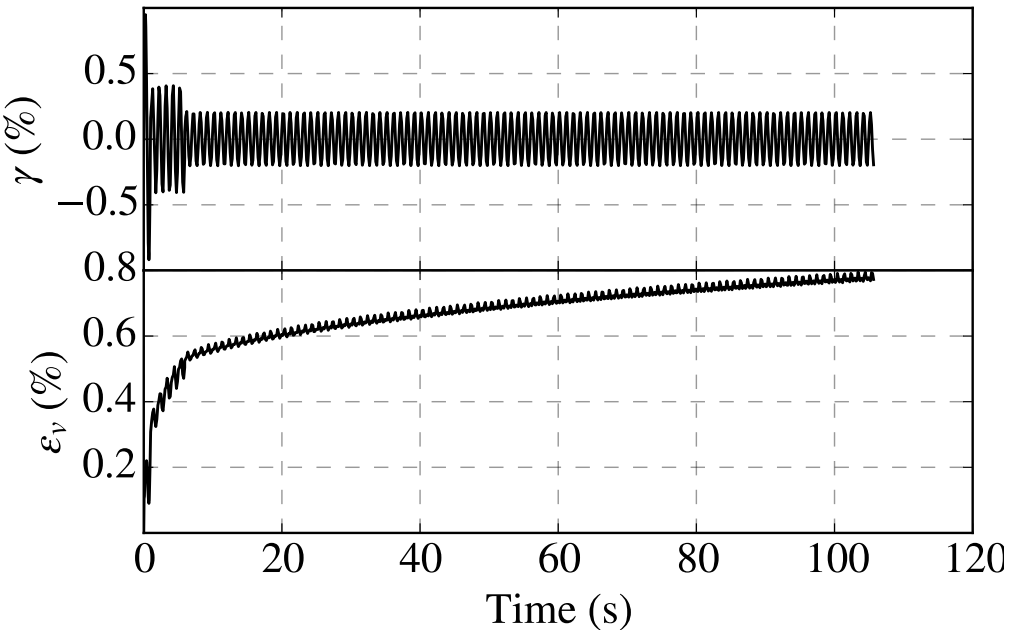


Figure B.18: Test results for irregular-loading seismic compression test *100.csv*.

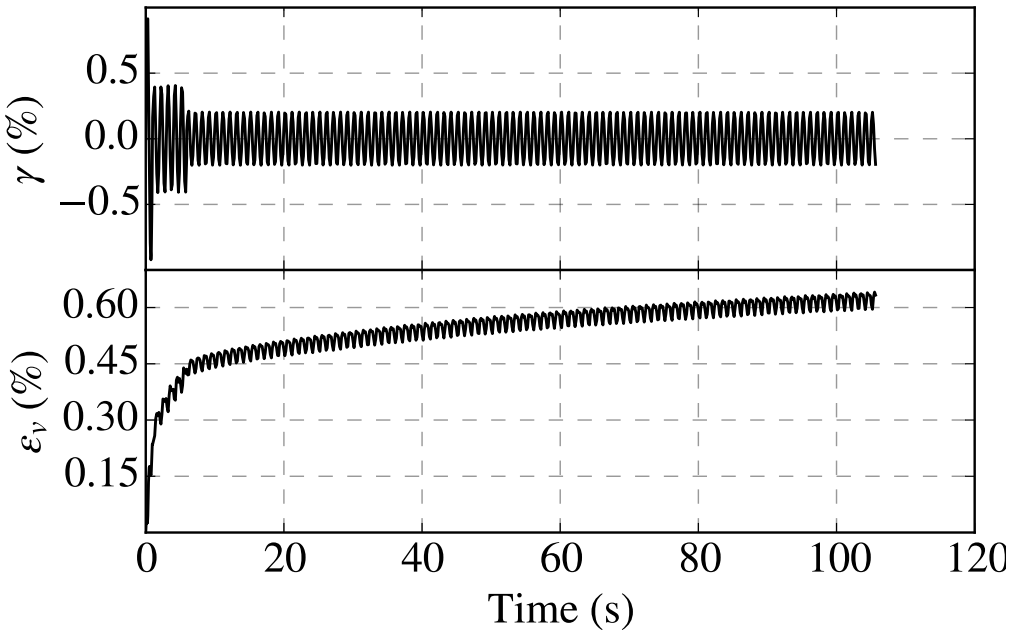


Figure B.19: Test results for irregular-loading seismic compression test *101.csv*.

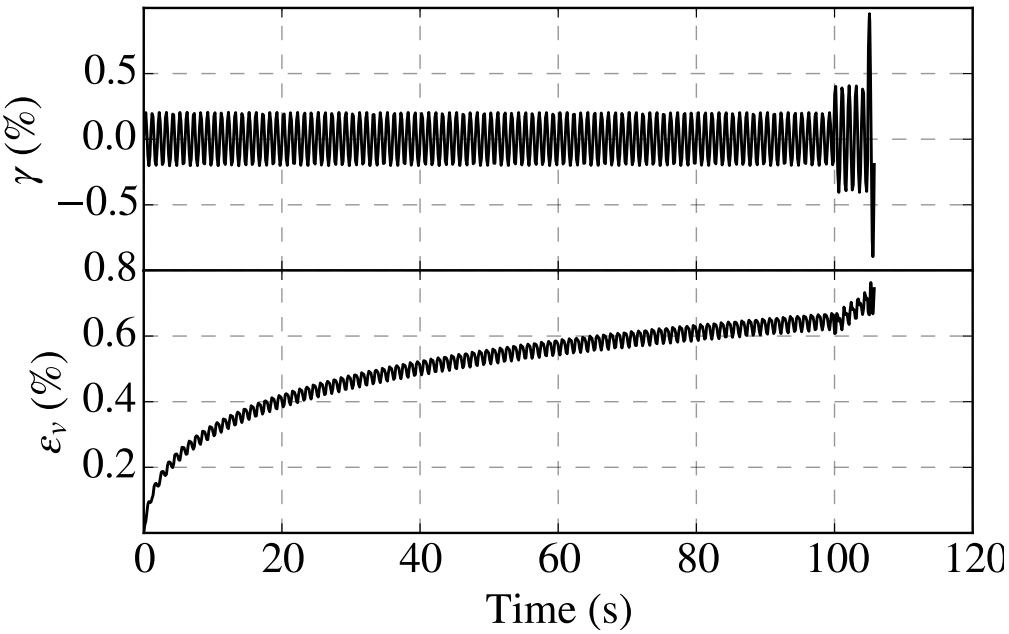


Figure B.20: Test results for irregular-loading seismic compression test *102.csv*.

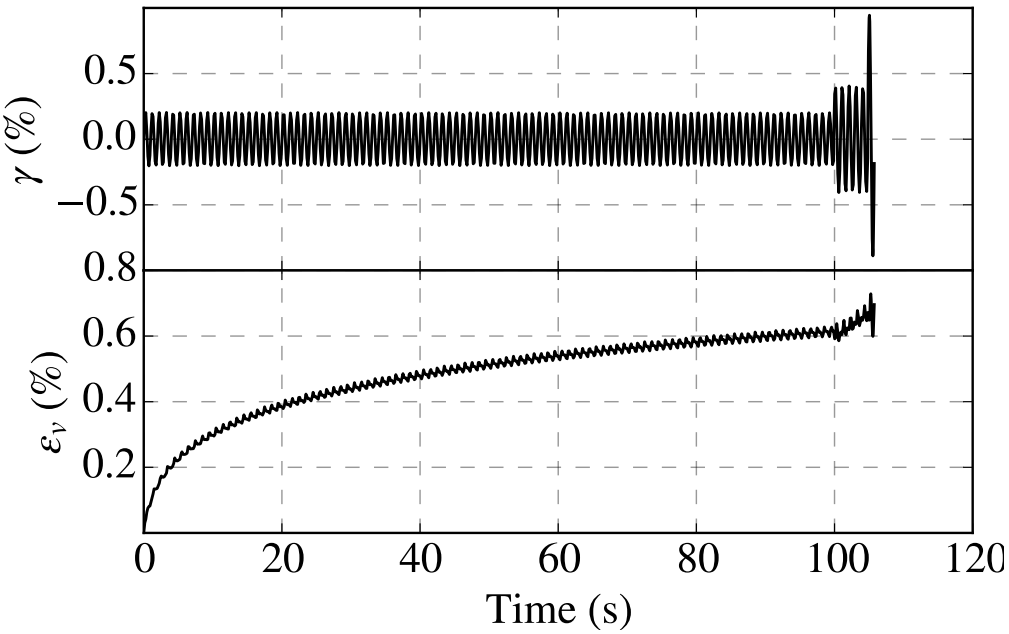


Figure B.21: Test results for irregular-loading seismic compression test *105.csv*.

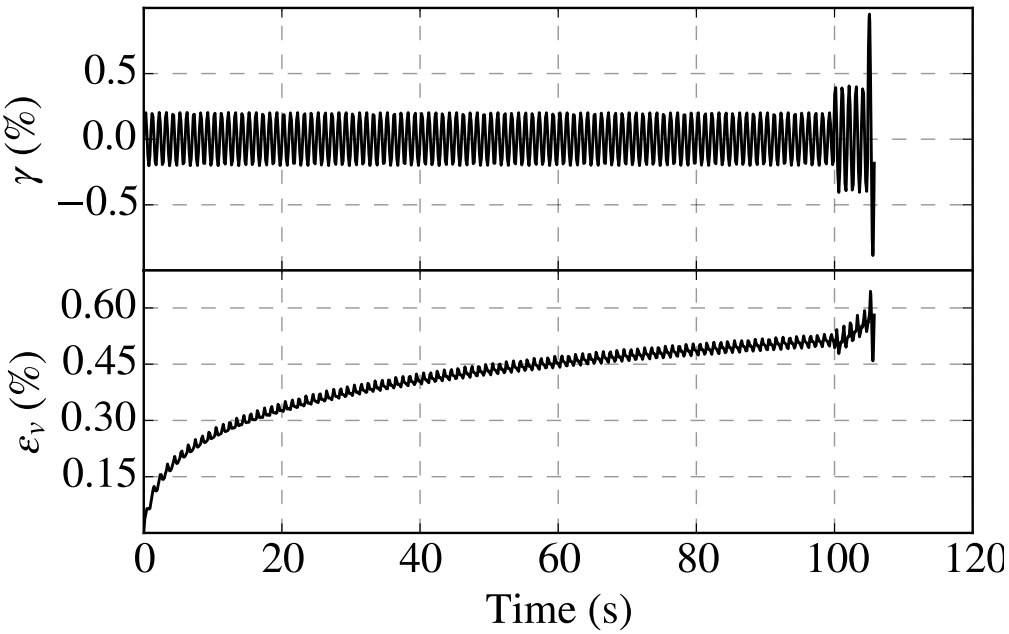


Figure B.22: Test results for irregular-loading seismic compression test *106.csv*.

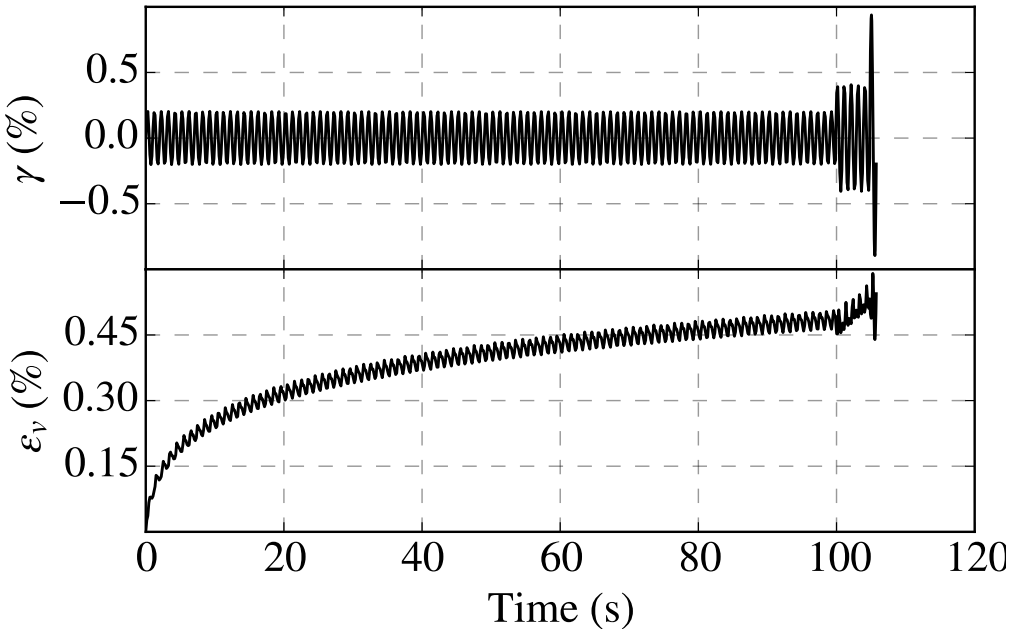


Figure B.23: Test results for irregular-loading seismic compression test *107.csv*.

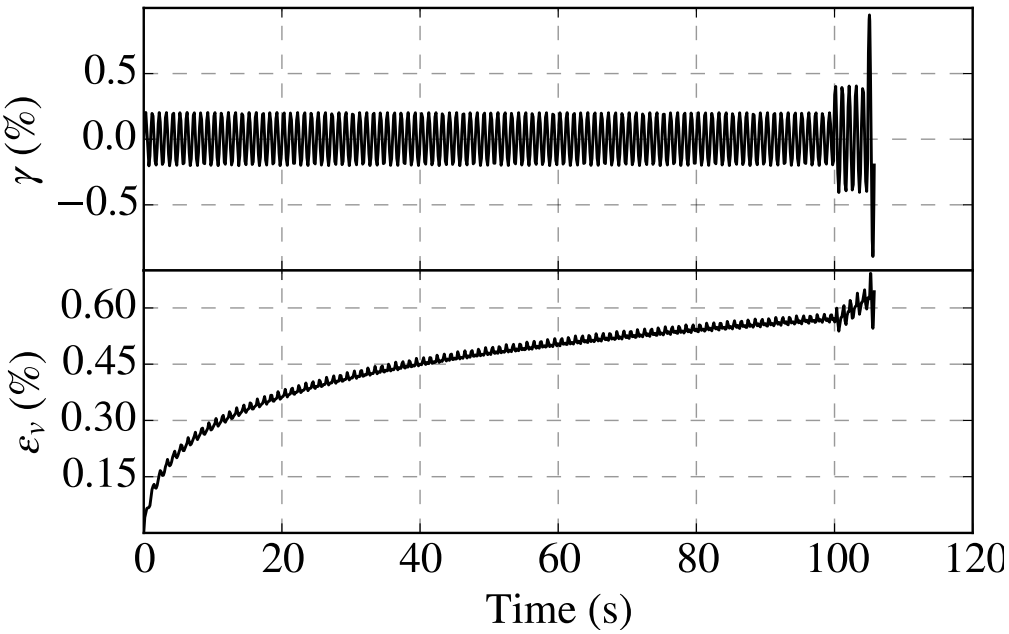


Figure B.24: Test results for irregular-loading seismic compression test *108.csv*.

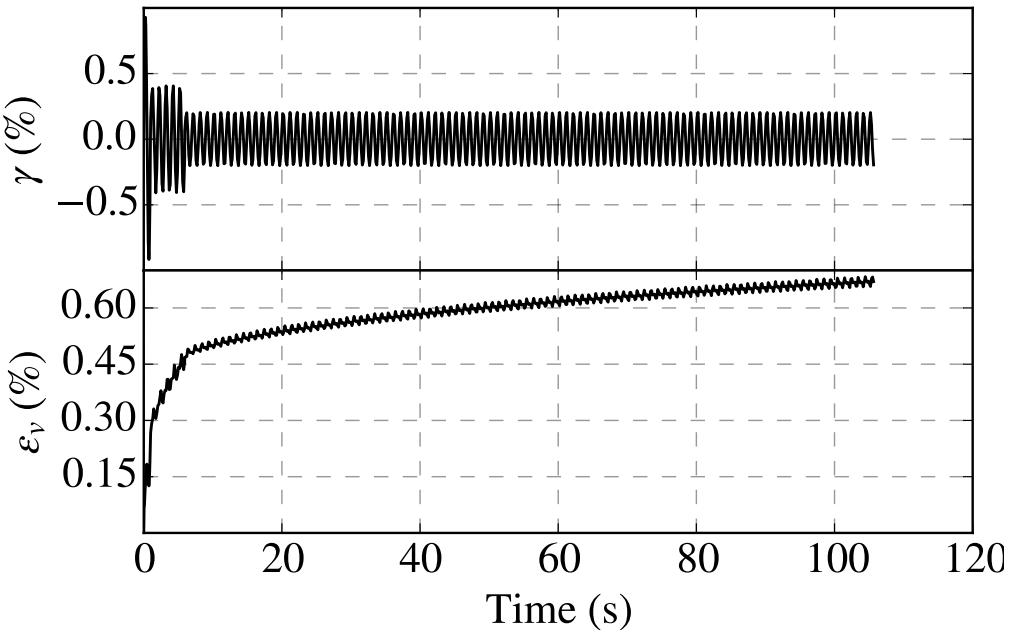


Figure B.25: Test results for irregular-loading seismic compression test *110.csv*.

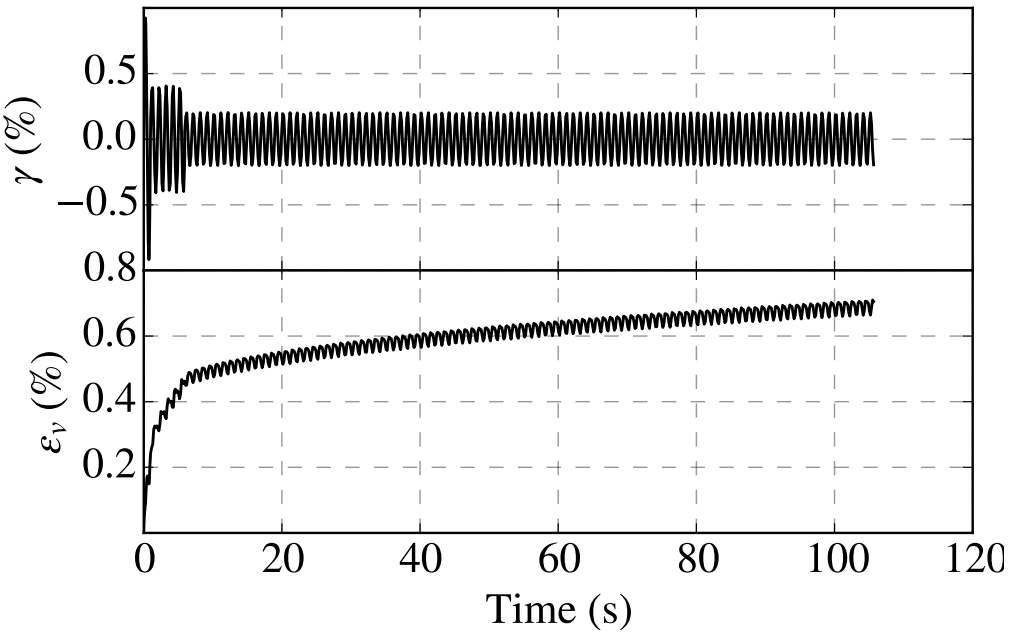


Figure B.26: Test results for irregular-loading seismic compression test *111.csv*.

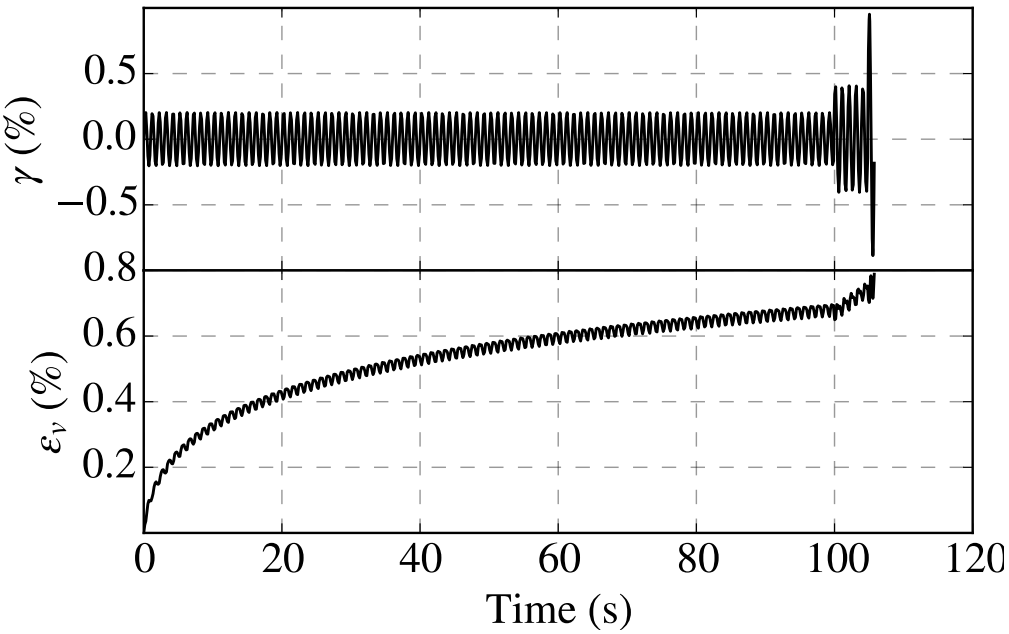


Figure B.27: Test results for irregular-loading seismic compression test *112.csv*.

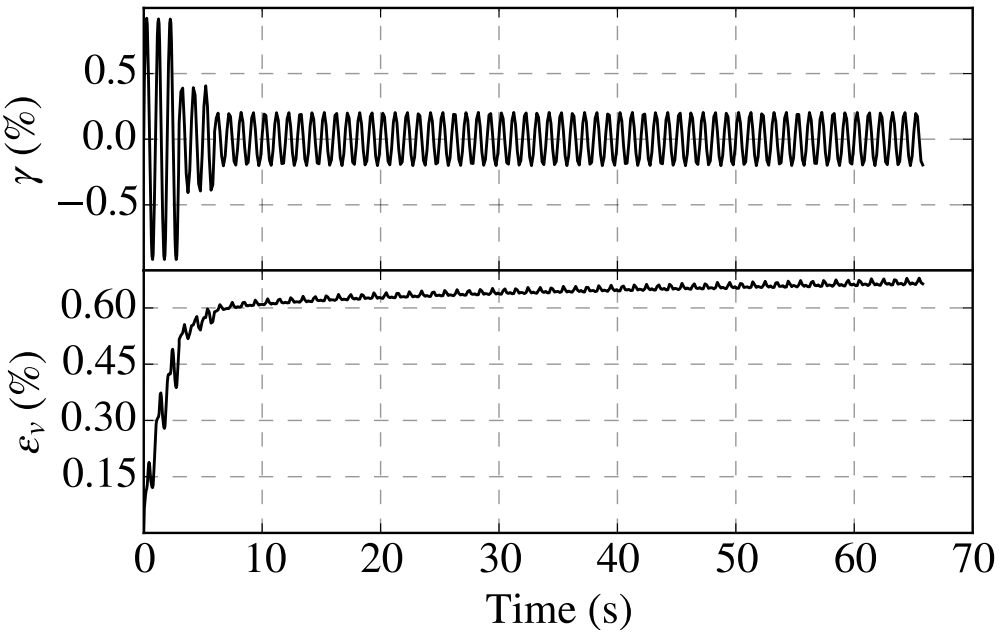


Figure B.28: Test results for irregular-loading seismic compression test *113.csv*.

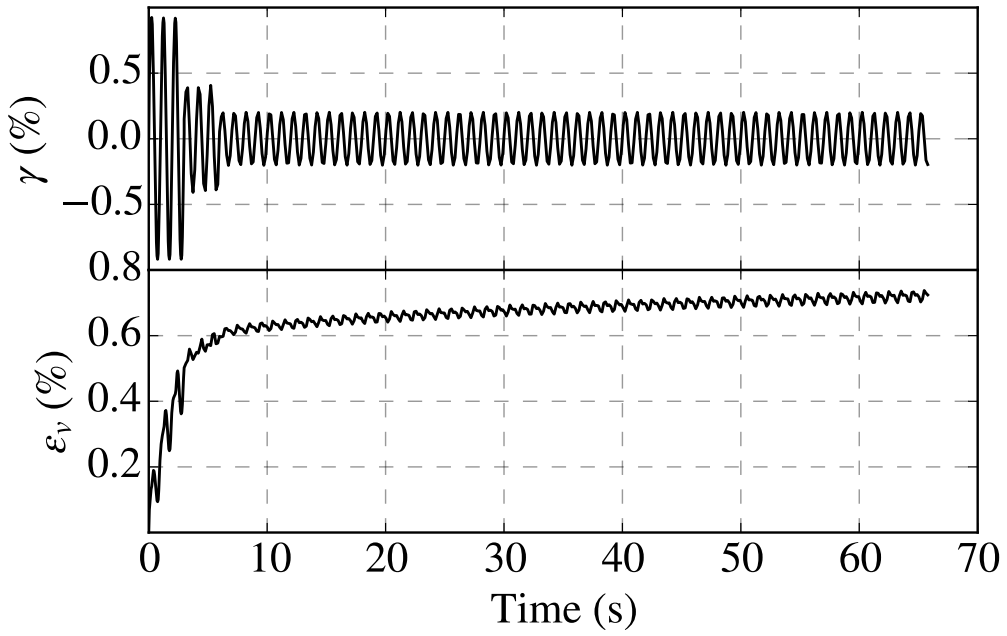


Figure B.29: Test results for irregular-loading seismic compression test *115.csv*.

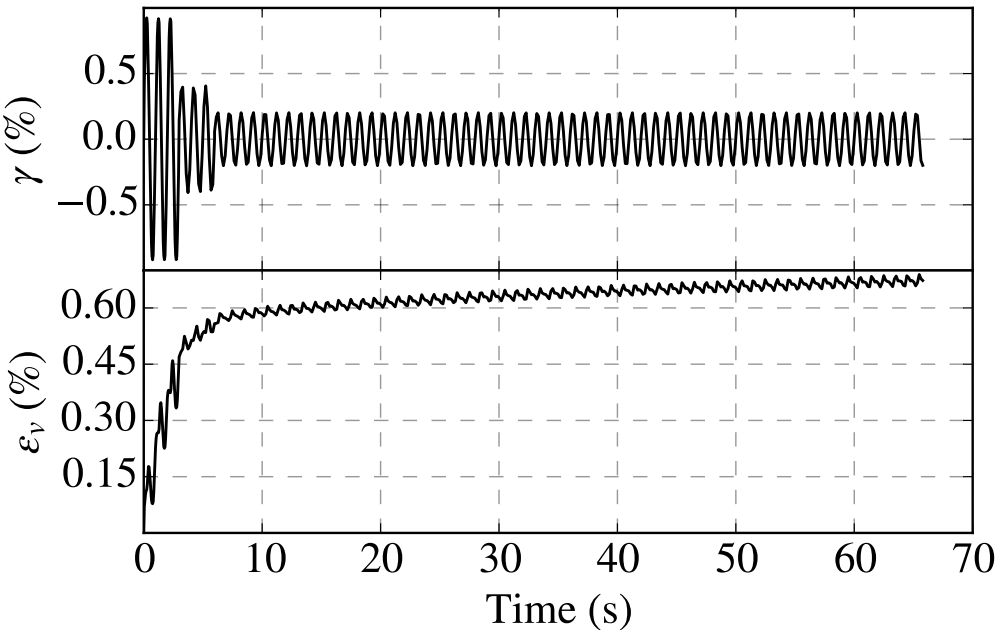


Figure B.30: Test results for irregular-loading seismic compression test *116.csv*.

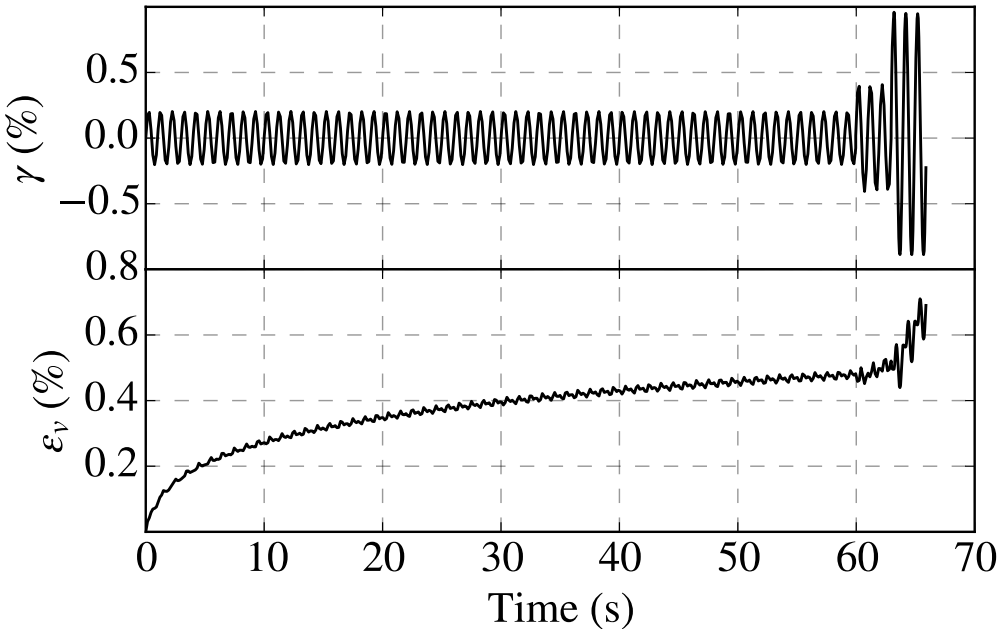


Figure B.31: Test results for irregular-loading seismic compression test *117.csv*.

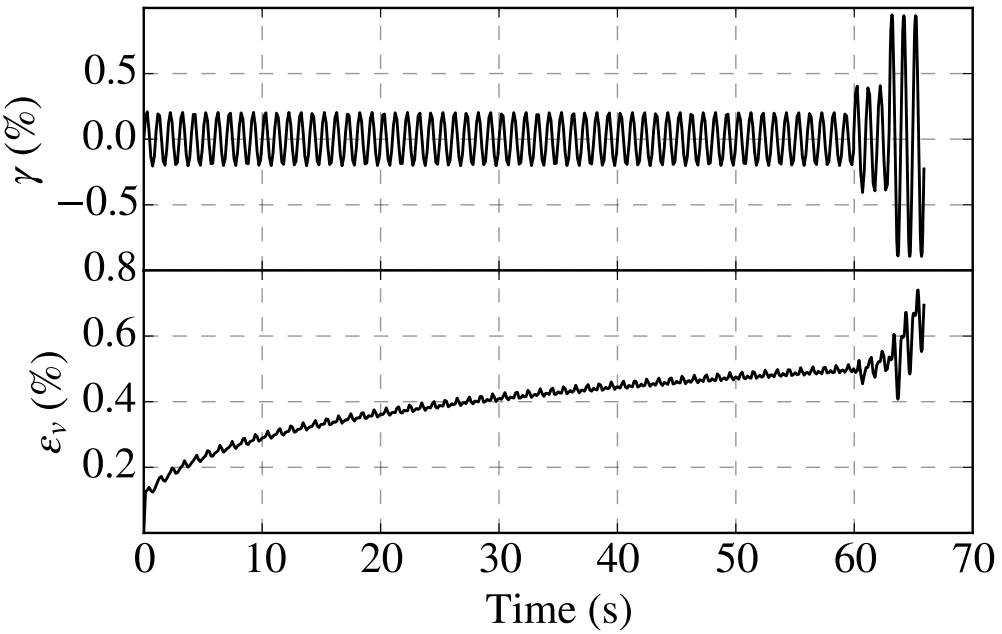


Figure B.32: Test results for irregular-loading seismic compression test *118.csv*.

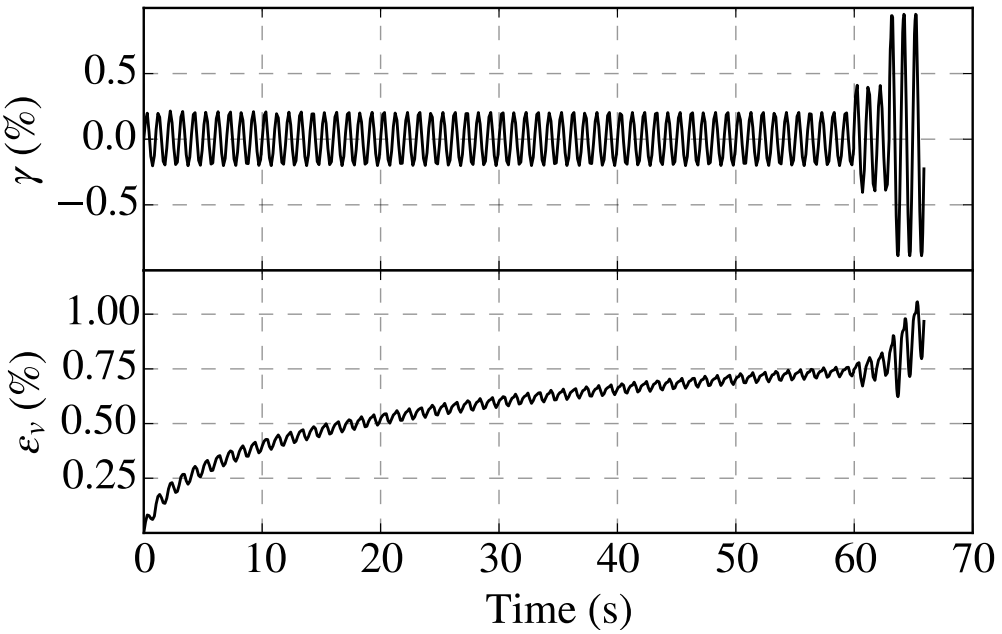


Figure B.33: Test results for irregular-loading seismic compression test *120.csv*.

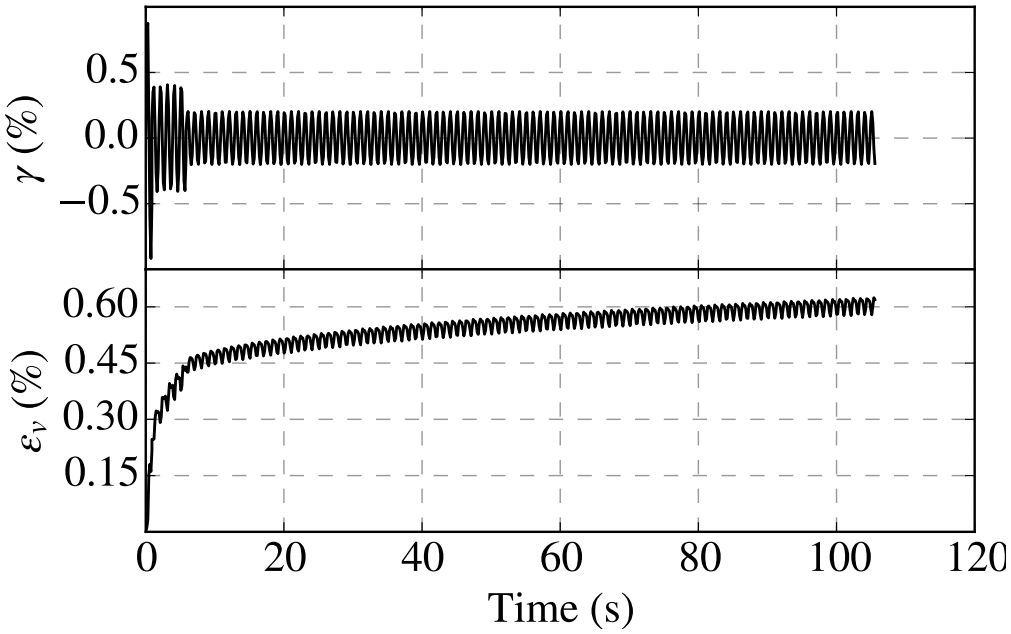


Figure B.34: Test results for irregular-loading seismic compression test *97.csv*.

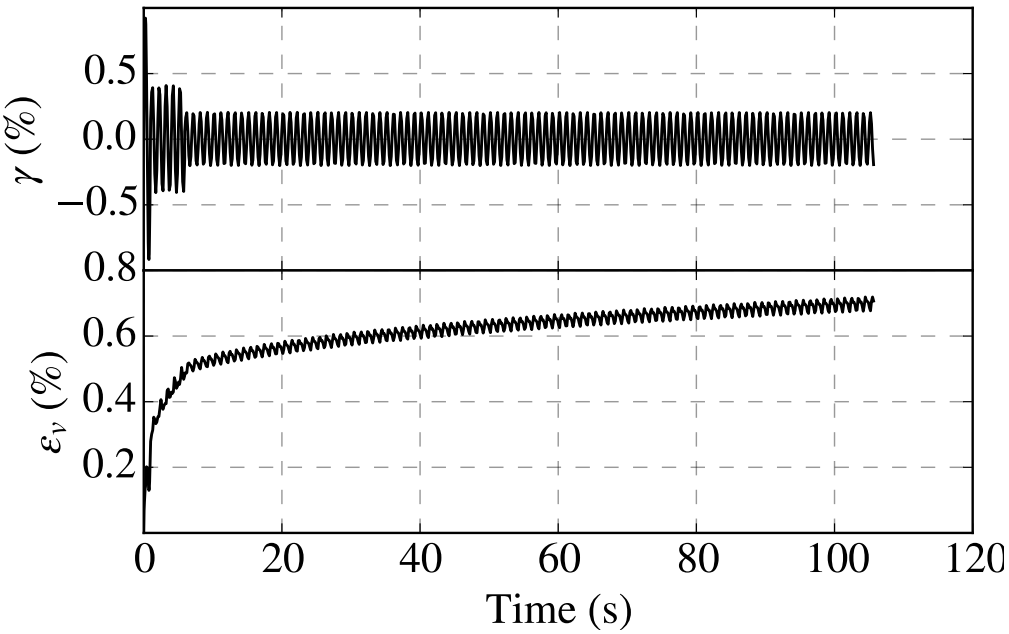


Figure B.35: Test results for irregular-loading seismic compression test *98.csv*.

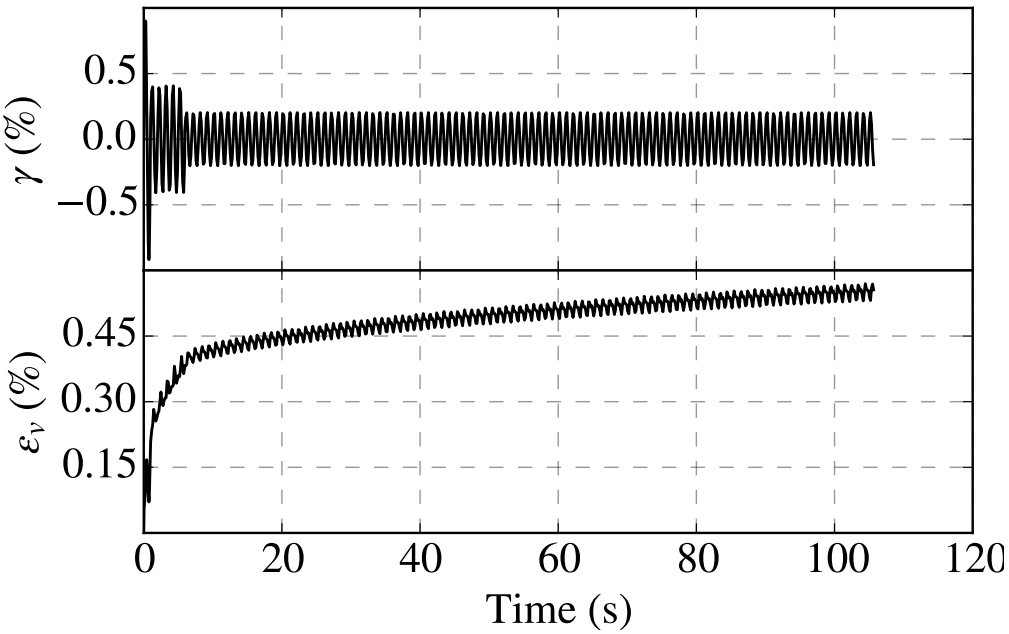


Figure B.36: Test results for irregular-loading seismic compression test *99.csv*.

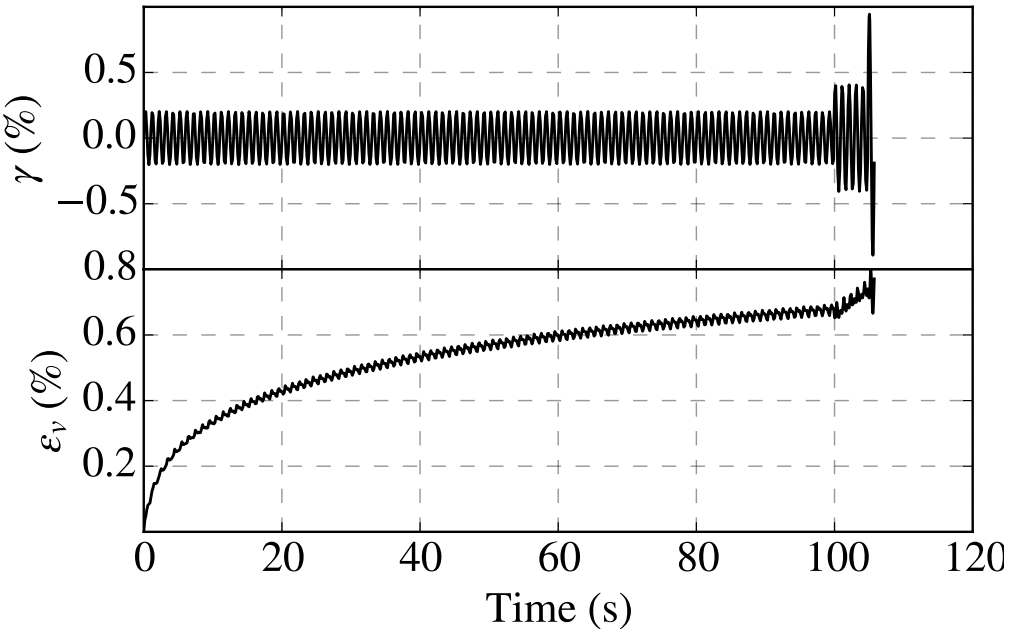


Figure B.37: Test results for irregular-loading seismic compression test *104.csv*.

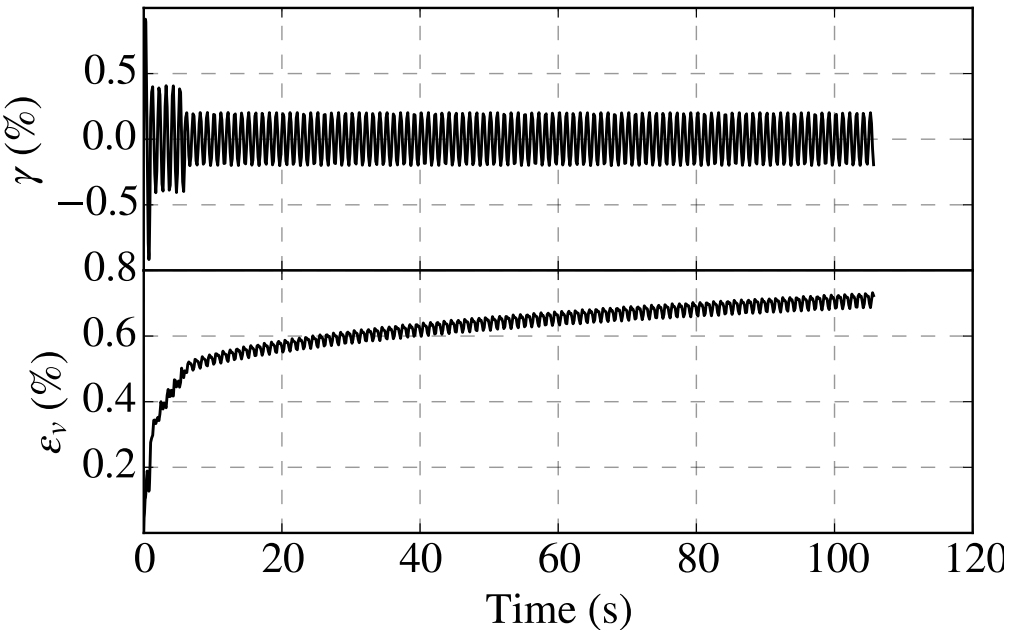


Figure B.38: Test results for irregular-loading seismic compression test *109.csv*.

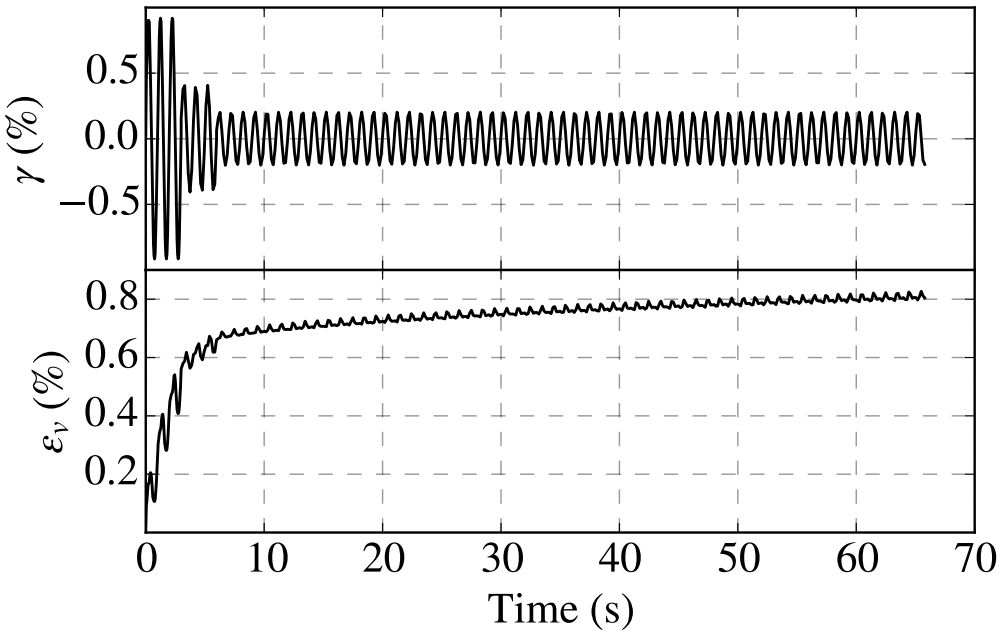


Figure B.39: Test results for irregular-loading seismic compression test *114.csv*.

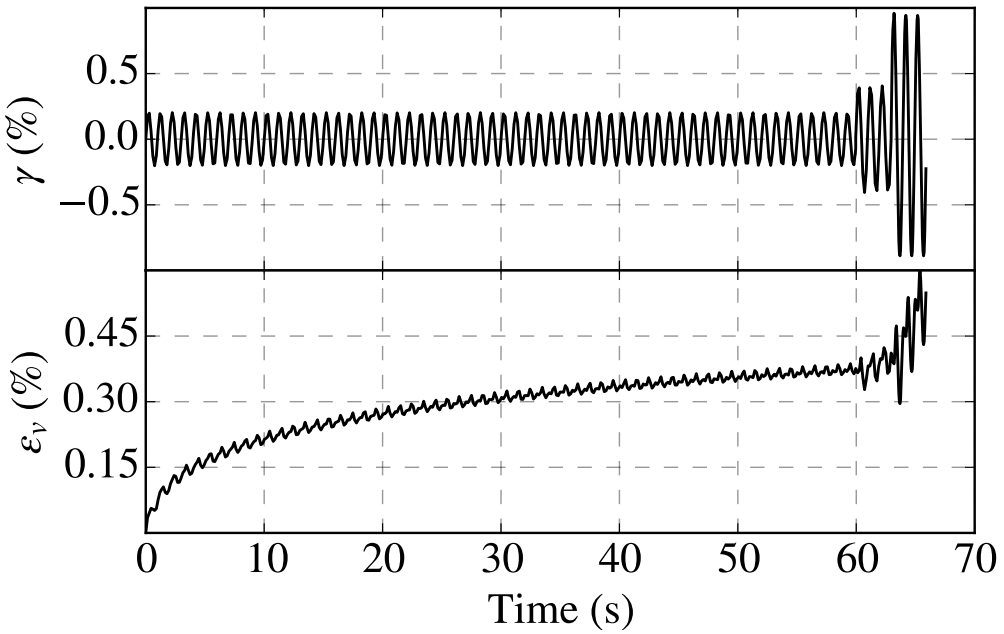


Figure B.40: Test results for irregular-loading seismic compression test *119.csv*.

Appendix C

Cycle Counting Methods for Seismic Compression

In order to estimate seismic compression using the Richart-Newmark (1948) method (Lasley et al., *in preparation*) for an irregular-loading situation, it is first necessary to obtain an array of loading sets with amplitudes and cycle counts (e.g. 2 cycles of amplitude x, 1 cycle of amplitude y, 3.5 cycles of amplitude z, etc.) from a loading time history. This technical note describes three methods or algorithms from ASTM Standard E1049 (2011) that were used in this research. In the ASTM specification, these methods are referred to as mean crossing peak counting (ZC), range counting (PP), and rainflow (RF) counting methods.

The mean crossing peak counting method counts as one cycle the largest peak between each mean-crossing. The mean used in this research was zero. Figure C.1 shows a random loading time history. For this loading time history, the load first crosses the mean (0) at about 7. Thus, the first cycle count is one cycle with an amplitude of 0.96. The mean is next crossed at about 10 with a peak between 7 and 10 of 0.7; the second cycle count is one cycle with an amplitude of 0.7. The pattern continues until there are no more mean crossings.

The range counting algorithm counts as one half cycle the difference between any two adjacent local peaks. Figure C.2 shows the amplitudes of two different one-half cycle counts. Each local peak is a stress reversal; the method counts as one half cycle the difference between each successive stress reversal. This method is arguable better for seismic compression than the mean crossing peak counting method because it takes into account all stress reversals.

The rainflow counting algorithm is somewhat different from the other two cycle counting algorithms. From a loading time history, the difference between three consecutive peaks/-valleys is compared and a cycle is counted only if the second difference is greater than or equal to the first. If Y contains the starting point, it is counted as one-half cycle, otherwise

it is counted as one full cycle. If the second difference isn't greater than or equal to the first, the next peak or valley is added to the list and the differences between the last three peaks/valleys are compared. The differences between leftover peaks are added in at the end. In Figure C.3, the first and second differences between peaks/valleys of the first step are labeled Y_1 and X_1, respectively. Since X_1 is greater than Y_1, the first one half cycle with amplitude Y_1 is counted. The algorithm continues, as shown in Table C.1, until no peaks or valleys are left. One issue with the rainflow counting algorithm is that cycles are sometimes counted out-of-order which, for seismic compression with the R-N model, changes the predicted damage.

For use in this research, the returned amplitude of loading from the rainflow and range counting methods was divided by two to match the calibration of the models. Similarly, the cycle counts returned by the mean crossing peak counting method was also divided by two for use in these procedures. In this way, the cycle amplitudes and counts across methods are nearly the same for a sinusoidal time history.

Table C.2 shows the median and interquartile range of the percent error in prediction for both earthquake-type and variable-amplitude sinusoidal loadings. The results of all three cycle counting methods is shown. For the R-N and Byrne methods, the mean crossing peak counting (ZC) method gives slightly better predictions than the other two algorithms. For the P-M method, the best predictions are made using the range counting (PP) method.

C.1 References

ASTM Standard E1049, 1985 (2011). "Practices for Cycle Counting in Fatigue Analysis." *ASTM International*, West Conshohocken, PA, www.astm.org

Lasley, S.J., Green, R.A., Chen, Q., and Rodriguez-Marek, A. (2015). "An Approach for Estimating Seismic Compression Using Site Response Analyses," *in preparation.*

Richart, F.E. and Newmark, N.M. (1948). "An hypothesis for determination of cumulative damage in fatigue," *ASTM Proceedings*, 48: 767-800.

C.2 Tables

Table C.1: Stepwise Illustration of the Rainflow Counting Algorithm

Step	Start Point	Y	X	Decision
1	A	|A-B|	|B-C|	$X \geq Y \Rightarrow$ |A-B| is one half cycle; discard A
2	B	|B-C|	|C-D|	$X < Y \Rightarrow$ Read next peak or valley
3	B	|C-D|	|D-E|	$X < Y \Rightarrow$ Read next peak or valley
4	B	|D-E|	|E-F|	$X \geq Y \Rightarrow$ |D-E| is one cycle, discard D & E
5	B	|B-C|	|C-F|	$X \geq Y \Rightarrow$ |B-C| is one half cycle, discard B
6	C	|C-F|	|F-G|	$X < Y \Rightarrow$ Read next peak or valley
7	C	|F-G|	|G-H|	$X \geq Y \Rightarrow$ |F-G| is one cycle, discard F & G
8	C	|C-H|	|H-I|	$X \geq Y \Rightarrow$ |C-H| is one half cycle, discard C
9	H	|H-I|	|I-J|	$X \geq Y \Rightarrow$ |H-I| is one half cycle, discard H
10	I	|I-J|	|J-K|	$X < Y \Rightarrow$ Read next peak or valley
11	I	|J-K|	|K-L|	$X < Y \Rightarrow$ Read next peak or valley
12	I	|K-L|	|L-M|	$X < Y \Rightarrow$ Read next peak or valley
13	I	|L-M|	|M-N|	$X \geq Y \Rightarrow$ |L-M| is one cycle, discard L & M
14	I	|I-J|	|J-K|	$X < Y \Rightarrow$ Read next peak or valley
15	I	|J-K|	|K-N|	$X \geq Y \Rightarrow$ |J-K| is one cycle, discard J & K
15	I	|J-N|	|N-O|	$X \geq Y \Rightarrow$ |I-N| is one half cycle, discard I
16	N	-	-	No more points $\Rightarrow$ |N-O| is one half cycle

Table C.2: Comparison of Cycle Counting Methods As Used with Seismic Compression Methods

Method	R-N	Byrne	P-M
Algorithm	*median*, *IQR*	*median*, *IQR*	*median*, *IQR*
PP	6.6, 20.1	2.9, 20.6	44.6, 64.1
RF	3.6, 22.5	-10.6, 34.7	44.5, 62.8
ZC	5.0, 18.1	-1.5, 17.2	43.6, 62.8
Mean	4.1, 22.2	-4.3, 20.6	43.3, 63.2

C.3 Figures

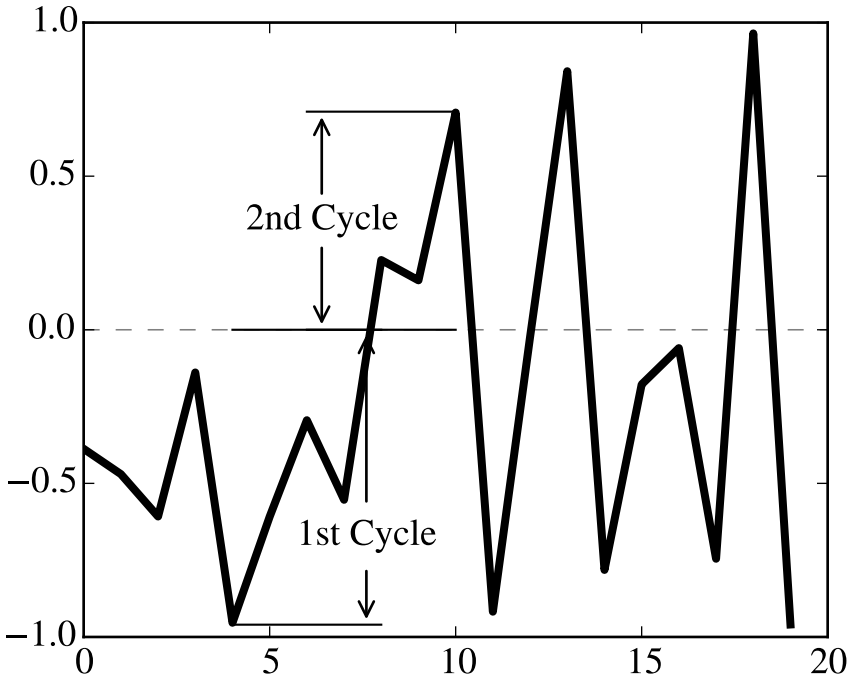


Figure C.1: An illustration of the *mean crossing peak counting* algorithm.

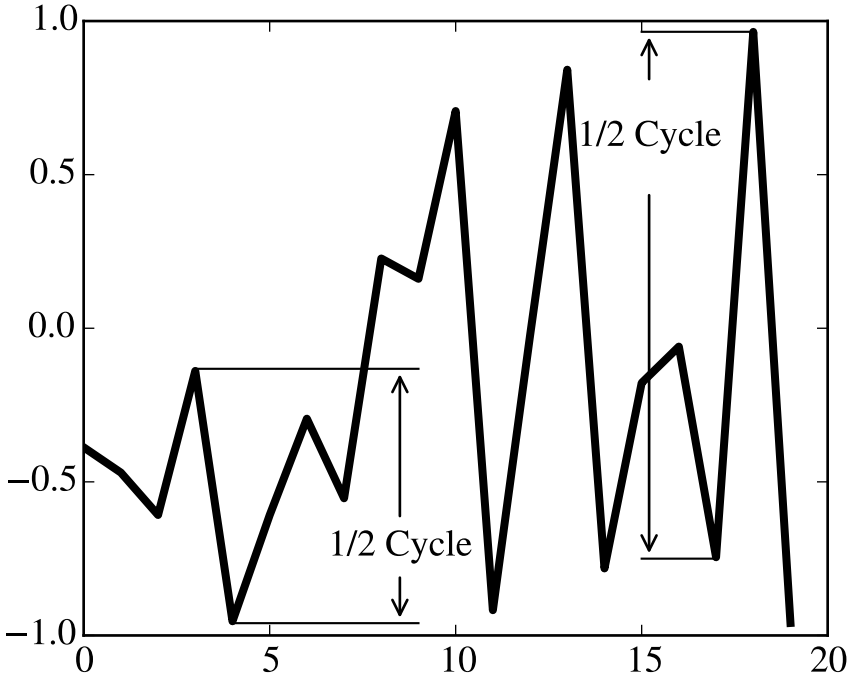


Figure C.2: Two one-half cycles as determined by the *range counting* algorithm.

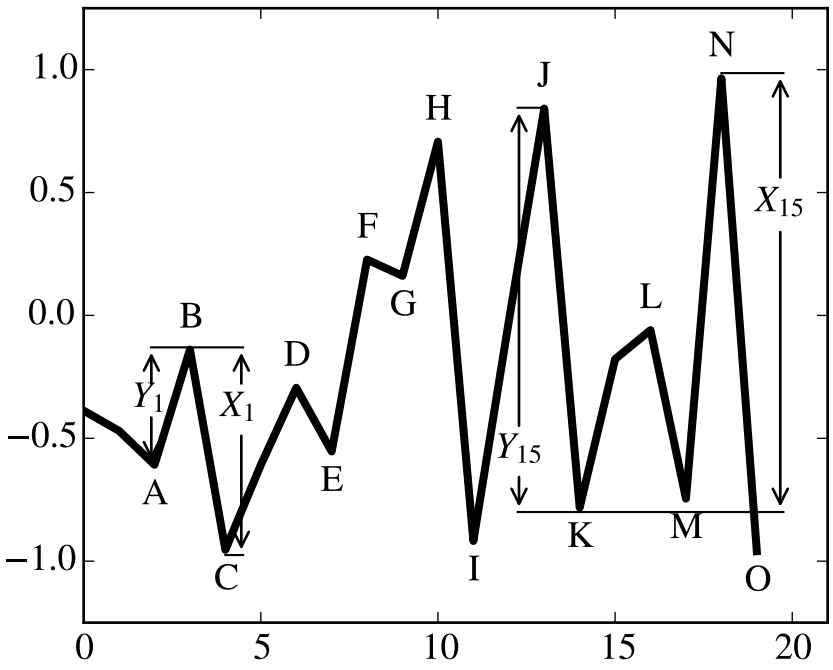


Figure C.3: The *rainflow counting* algorithm.

Appendix D

Cyclic Simple Shear Lab Test Results

D.1 Introduction

A number of cyclic simple shear tests were performed as part of this research. Two different testing programs were followed in this research; one program studied seismic compression, and the other studied liquefaction triggering. The seismic compression testing programming was undertaken from January through April 2011. The liquefaction triggering program was undertaken from April 2014 to May 2015.

This chapter details the testing equipment and testing procedure of both programs and the testing results of the liquefaction triggering program.

D.2 Setup and Equipment

Cyclic laboratory testing was performed using the GCTS SSH-100 cyclic simple shear equipment. This equipment is capable of performing both cyclic simple shear and cyclic triaxial tests with stress- and strain-controlled motions. In addition, the equipment is capable of applying multiple load functions, from sinusoidal loadings to any arbitrary motion.

Two significant modifications were made to the testing equipment in order to obtain better cyclic simple shear test results. In the early stages of the seismic compression testing program, it was observed that significant compliance occurred in the shear direction during the cyclic loading phase. To remedy the issue, large external braces were fit to the equipment. The design of these braces prevents the cell from being used when the full

bracing is installed. However, the cell was unnecessary for either of testing programs described herein.

The second modification simply replaced one of the parts to which the top platen is mounted during testing. This part was resized to allow space for an internal normal pressure cell. This was necessary to perform constant-volume cyclic simple shear tests, as discussed in a subsequent section. Figure D.1 shows the two modifications in place for a test, and Figure D.2 shows the second modification in greater detail.

D.3 Specimen Preparation

For both testing programs, the cyclic simple shear specimens were prepared via dry pluviation. Pluviation mimics the natural deposition of sands and has been shown to provide a uniform fabric (Vaid and Negussey 1984; 1988). However, due to potential for grain-size segregation, pluviation is best used with poorly-graded sands only (Kuerbis and Vaid 1988).

Dry pluviation was performed by placing dry sand into a flask, attaching a stopper and funnel to the flask opening, and by gently swirling the flask as sand was poured through the funnel into the specimen mold. Figure D.3 shows the flask with stopper, funnel, and sand. For the seismic compression program, a specific mass of sand was placed in the flask, the sand was loosely poured into the mold, the surface of the sand was leveled, and the desired relative density was achieved by vibrating the specimen with the top platen in place. For the liquefaction triggering program, the mold was overfilled and the excess sand was removed using a straightedge (Figure D.4). In this case, the density of the sand was controlled by adjusting both the pour height and the diameter of the funnel opening.

D.4 Testing Procedure

The basic testing procedure for both programs involved three phases: specimen preparation, consolidation, and cyclic loading until failure. For both testing programs, dry specimens were used and drained conditions prevailed throughout the consolidation and cyclic loading phases. Consolidation was achieved by applying a normal stress of 50, 100, or 250 kPa (60, 100, 250 kPa for the liquefaction triggering program) to the top of the specimen until vertical strains ceased to accumulate. The void ratio and relative density at the end of consolidation are the values used in subsequent analyses. During consolidation and the cyclic loading phase, lateral deformations were prevented by the brass confining rings outside the specimen (shown in Figures D.1 and D.4).

For the cyclic loading phase of the seismic compression testing program, vertical stresses were held constant (at 50, 100, or 250 kPa) and vertical strains were allowed to occur while cyclic shear strains were applied to the base of the specimen (i.e. the seismic compression

tests were strain controlled). Because lateral deformations were prohibited by the confining rings, vertical strains equaled volumetric strains. Cyclic loading was stopped after volumetric strains exceeded one percent.

The liquefaction triggering program was conducted using constant-volume (C-V) tests (Finn and Vaid 1977; Finn et al. 1979); vertical strains were minimized during cyclic loading, and the vertical stress was allowed to change as cyclic loading progressed. For constant-volume tests, a decrease in vertical stress corresponds to an increase in pore water pressure; the excess pore water ratio reported from these tests is a *simulated* value. Both stress- and strain-controlled tests were performed, with both sinusoidal and earthquake motions used as loading functions for the stress-controlled tests. The cyclic phase continued until liquefaction had occurred as determined by a specific failure criterion. The specific failure criterion depended on the type of loading and will be detailed in subsequent sections.

D.5 Materials

Two different sands were used; a nondescript quartz sand was used in the seismic compression program, and Monterey 0/30 sand was used for the liquefaction triggering program. Minimum and maximum void ratios were determined following ASTM (2000) specifications D4254 and D4253, respectively. Both sands are poorly-graded (see Figure D.5) and suitable for use with the dry pluviation preparation method. Table D.1 summarizes some of the index properties of each sand.

D.6 Results

A discussion on the results of the seismic compression program is given in Chapter 3; this section will focus on the results of the liquefaction triggering program. As mentioned in a previous section, stress-controlled tests were performed using sinusoidal and earthquake loading functions, and strain-controlled tests were also performed using sinusoidal loading functions. The results from each of these loading functions is given in this section, and a comparison between the three is also given. Finally, the results from this study are compared to those from two other studies.

D.6.1 Stress-Controlled Sinusoidal Tests

A number of stress-controlled sinusoidal-loading tests were performed on the Monterey 0/30 sand. The relative density of the test specimens ranged from approximately 30 to 80%, and overburden pressures of 60, 100, and 250 kPa were imposed for the entire range of densities. A failure criterion of 3.75% single-amplitude shear strain was adopted (Vaid

and Sivathayalan 1996; Sivathayalan and Ha 2011), and this section discusses the results of 49 tests that exceeded this failure criterion.

Figures D.6 and D.7 show typical results for loose and dense specimens, respectively. From top to bottom, the figures show plots of the progression of normal and shear stresses, normal strain, and simulated pore pressure ratio ($r_u = u_{excess}/\sigma_v$ where u_{excess} is defined here as $\sigma_{v0} - \sigma_v$). At the bottom is a plot of shear stress versus shear strain. Loose specimens experienced a "brittle" failure where large shear strains are observed in the last few cycles only. Dense specimens, on the other hand, experienced what might be termed a "ductile" failure with large shear strains for many cycles before failure. While an attempt was made to prevent any volumetric strain from occurring during the cyclic loading phase, some volumetric strain did occur with some tests exceeding 0.3% volumetric strain. For loose specimens, most of this volumetric strain occurred in the last few cycles only, but this was not the case for the dense specimens.

Figure D.8 shows a plot of cyclic stress ratio (CSR) versus the number of cycles to failure ($N_{liq.}$). Contours of relative density are shown, and only four tests with relative densities greater than about 55% exceeded the shear strain failure criterion of 3.75%. The logarithm of CSR was regressed as a function of relative density and the logarithm of $N_{liq.}$ with a resulting $R^2_{adj.}$ value of 0.90.

As the purpose of this research was to incorporate dissipated energy into liquefaction evaluations, the dissipated energy to failure was calculated for each test. As discussed in Chapter 2, the dissipated energy per unit volume is estimated using the trapezoidal rule and the stress and strain time histories of each test. It was calculated up to the last data point of zero shear stress before the failure criterion was exceeded. This data point is illustrated by the green dot in Figures D.6 and D.7; the red dot corresponds to the data point at which the failure criterion is exceeded. The dissipated energy calculated in this manner is $\Delta W_{\text{Effect.}}$, the dissipated energy corresponding to an effective stress analysis.

Figure D.9 shows the normalized dissipated energy ($\Delta W_{\text{Effect.}}/\sigma'_{v0}$) versus relative density (D_r). The dissipated energy to liquefaction increases with increasing initial effective overburden stress, but the normalized dissipated energy is considered to be constant for a given relative density. Using the results of all the stress-controlled sinusoidal-loading tests, the logarithm of the normalized dissipated energy is regressed as a function of the square of the relative density with an R^2 value of 0.75. Judging by the R^2 values, the CSR-$N_{liq.}$-D_r relationship is slightly stronger than the $\Delta W_{\text{Effect.}}/\sigma'_{v0}$-$D_r$ relationship. However, the former relationship corresponds to a stress-controlled sinusoidal loading, and the latter relationship should be insensitive to the loading function.

D.6.2 Earthquake Loading Tests

A few stress-controlled tests were performed with earthquake-motion loading functions. The loading functions were obtained using equivalent-linear site response analyses, a simple sand profile, and a few earthquake motions from the NGA database (Chiou et al. 2008). The profile used in the equivalent-linear analyses is clean sand to bedrock at a depth of

30 meters and a shear wave velocity that increases from 74.6 m/s at the surface to 225.3 m/s at the layer just above the bedrock (Lee 2009). The shear stress motions used as the loading functions were taken from a depth of 5.1 m which corresponds to an overburden pressure of 100 kPa. These response shear stress motions were then scaled so that the CSR values of the tests ranged from 0.14 to 0.3.

Figure D.10 shows a typical test result, and Figure D.11 shows the normalized dissipated energy versus relative density for these tests. Of all the tests, only 24 exceeded the failure criterion of 3.75% shear strain, and all of these corresponded to a single input motion (NGA_no_87_SAD273). Relative densities of the tests were limited to less than 50%, and the trend of normalized dissipated energy to relative density is weak ($R^2 = 0.07$). However, the normalized dissipated energy required to cause liquefaction was fairly constant over all overburden stresses and cyclic stress ratios. The recorded motions used as input functions were not sufficiently long to cause liquefaction for dense specimens; this could be avoided in future testing by using artificial motions, by splicing together several motions, or by repeating a motion multiple times.

D.6.3 Strain-Controlled Tests

A few strain-controlled sinusoidal loading tests were performed to complement the stress-controlled tests. Figures D.12 and D.13 show typical results for loose and dense specimens, respectively. In a stress-controlled test, hysteresis loops increase in size as cyclic loading progresses, but in strain-controlled tests, the area bound by the hysteresis loops decreases as loading progresses. In other words, the shape of the dissipated-energy-accumulation path is different between the two types of tests. This is reflected in the paths of the pore pressure ratios, especially in loose specimens. At the last part of the test, pore pressure ratios increase sharply in a stress-controlled test, but flatten out in a strain-controlled test (see Figures D.6 and D.12). Altogether, comparing the results of stress- and strain-controlled tests provide a way to examine the efficacy of normalized dissipated energy as an indicator of liquefaction triggering.

Figure D.14 shows the single-amplitude shear strain (γ) versus the number of cycles to liquefaction ($N_{liq.}$) for the strain-controlled tests. This is analogous to the CSR-$N_{liq.}$ (Figure D.8) plot of stress-controlled tests. A failure criterion of $r_{u,static} = 0.98$ (i.e. $r_u \geq 0.98$ when the applied shear stress is equal to zero) was adopted for these strain-controlled tests, and 26 tests exceeded that threshold.

Figure D.14 also shows contours of relative density, and the logarithm of γ was regressed as a function of D_r and the logarithm of $N_{liq.}$ with an $R^2_{adj.}$ value of 0.36. Compared to the CSR-$N_{liq.}$-D_r relationship of the stress-controlled tests, the γ-$N_{liq.}$-D_r relationship of the strain-controlled tests is much weaker. However, Figure D.15 shows a strong trend ($R^2 = 0.82$) between the normalized dissipated energy and relative density. This R^2 value is greater than that given for the stress-controlled tests (0.75).

D.6.4 Comparison Between Stress- and Strain-Controlled Tests

Using the results of both the stress- and strain-controlled sinusoidal loading tests, additional regression coefficients were obtained via bootstrapping (Efron and Tibshirani 1994) for the following functional form:

$$\frac{\Delta W_{\text{Effect.}}}{\sigma'_{v0}} = \beta_0 + \beta_1 D_r^2 + \varepsilon \tag{D.1}$$

where $\Delta W_{\text{Effect.}}/\sigma'_{v0}$ is the normalized dissipated energy per unit volume corresponding to effective stress analyses, D_r is the relative density in percent, β_0 and β_1 are regression coefficients, and ε is an error term. This is the same functional form presented in Figures D.9, D.11, and D.15. The regression coefficients presented in those figures were obtained using ordinary least squares regression. The bootstrapping method used in this section also implements ordinary least squares regression, but in a different manner:

1. From each dataset, 50 data points were randomly sampled with replacement (each dataset has less than 50 data points).
2. Using these 50 data points, ordinary least squares regression was performed to obtain the regression coefficients of β_0 and β_1.
3. Steps 1 and 2 were repeated for a total of 200 times, recording the regression coefficients for each iteration.

By using this bootstrapping method, the variability in each regression coefficient can be examined. In other words, the results of bootstrapping indicate how well each regression coefficient is constrained by the data. Table D.2 shows the mean and standard deviations of each regression coefficient for the two datasets and for a dataset that combined the two. For all three datasets (stress-controlled, strain-controlled, and combined), the mean and standard deviations of the regression coefficients are very similar. This is illustrated in Figure D.16 where the stress- and strain-controlled curves show good agreement; these curves use the non-bootstrapped regression coefficients.

In order to evaluate whether the regression coefficients are significantly close to be considered the same, two Welch's (1947) t-tests were performed on the sets of regression coefficients from the bootstrapping. Welch's t-test is used to test whether two populations have the same means, and it was chosen in this case (instead of the Student's t-test) because Welch's test does not assume that the samples have equal variances. For this t-test, the null hypothesis (H_0) and alternative hypothesis (H_a) are defined as:

- $H_0 : \overline{\beta_{0,stress}} = \overline{\beta_{0,strain}}$ and $\overline{\beta_{1,stress}} = \overline{\beta_{1,strain}}$
- $H_a : \overline{\beta_{0,stress}} \neq \overline{\beta_{0,strain}}$ and $\overline{\beta_{1,stress}} \neq \overline{\beta_{1,strain}}$

The Welch's t-test returns a t value and corresponding p-value. If the p-value is less than the level of significance, α, then the null hypothesis is rejected. For this study, an α value of 0.05 was adopted. Table D.3 shows the results of the t-tests. For both the intercept (β_0) and slope (β_1) terms, the p-value is greater than α which indicates that the null hypothesis cannot be rejected based on this data. In other words, the differences between the stress-controlled and strain-controlled intercepts and slopes are not significantly different. It should be noted, however, that the intercept term is barely insignificant with a p-value of 0.0501. Also, as the number of bootstrap iterations increases, the p-values decrease and the null hypotheses will then be rejected.

D.6.5 Comparison With Other Test Programs

As part of his research in energy-based liquefaction evaluation, Green (2001) examined the results of a number of stress-controlled cyclic triaxial tests that were performed on Monterey and Yatesville sands. Figure D.17 shows the normalized dissipated energy calculated for each test on Monterey sand ($FC \leq 5\%$) as a function of relative density. The failure criterion used in these tests was $r_u = 1.0$ for the majority of the tests, and a double-amplitude axial strain (ε_{DA}) of 5% for a few cases. Also shown in Figure D.17 are the regression coefficients for the same $\Delta W_{\text{Effect.}}/\sigma'_{v0}$-$D_r^2$ relationship. Compared to the regression coefficients of the fit to all the laboratory tests of this study (Figure D.18), the Green (2001) data shows a very similar trend.

Riemer et al. (1994) performed stress- and strain-controlled cyclic triaxial tests on Monterey #0 sand. The main purpose of their work was to study the effects of loading frequency on the liquefaction triggering of a clean sand. Accordingly, they performed their tests at frequencies of 0.1, 1, 10, 15, and 20 Hz. They also studied the accumulation of dissipated energy as loading progressed, and a plot of normalized dissipated energy versus relative density (Figure D.19) from their results is of interest here. Their tests were performed at relative densities of 40, 50, and 60%, and the trend with relative density is weaker than that of Green (2001) or that of this research.

Figure D.20 shows the data and trends from all three datasets: Green (2001), Riemer et al. (1994), and the results of all cyclic simple shear tests from the liquefaction triggering program. All three datasets have similar values of normalized dissipated energy at relative densities between 40 to 50%. At higher relative densities, the data from Green (2001) shows a trend of lower normalized dissipated energy when compared to the data from this study. At a relative density of 60%, the Riemer et al. (1994) data has a large spread of normalized dissipated energies; the trend lines from Green (2001) and this study fall within this spread.

Despite the fact that the three datasets were obtained from two different tests (cyclic triaxial and cyclic simple shear), two different sands (Monterey 0/30 and Monterey #0), and four different failure criteria ($r_u = 98\%$, $r_u = 100\%$, $\varepsilon_{DA} = 5\%$, and $\gamma = 3.75\%$), the resulting $\Delta W_{\text{Effect.}}/\sigma'_{v0}$-$D_r^2$ relationships are remarkably similar between the datasets. This level of agreement is not possible with traditional relationships (e.g. CSR-$N_{liq.}$-D_r) because both stress- and strain-controlled test results are included here. Normalized

dissipated energy and relative density provide a robust liquefaction triggering criterion that merges the stress-based and strain-controlled laboratory testing results.

D.7 Summary

A number of cyclic simple shear tests were performed in support of the seismic compression and liquefaction triggering research. The test specimens were prepared via dry pluviation of quartz sand for seismic compression and Monterey 0/30 sand for liquefaction triggering.

Constant-volume cyclic simple shear tests were used in the liquefaction-triggering testing program, and the results show that normalized dissipated energy and relative density together are overall better predictors of initial liquefaction than CSR, $N_{liq.}$, and D_r or γ, $N_{liq.}$, and D_r. This was illustrated with the results of stress-controlled sinusoidal loading tests and strain-controlled sinusoidal-loading tests. Due to a lack of results at high relative densities, the regressed $\Delta W_{\text{Effect.}}/\sigma'_{v0}$-$D_r^2$ curve from the earthquake loading data is poorly constrained. However, the regressed curves from the stress- and strain-controlled sinusoidal-loading tests give good agreement for the range of relative densities for which tests were performed ($\sim$ 30 to 80%).

The results from this study were compared with those from Green (2001) and Riemer et al. (1994). Both Green (2001) and Riemer et al. (1994) performed cyclic triaxial tests. Despite the differences in laboratory test and failure criteria, all three sets of results show a similar trend of increasing normalized dissipated energy with relative density and similar magnitudes of normalized dissipated energy to liquefaction. The proposed $\Delta W_{\text{Effect.}}/\sigma'_{v0}$-$D_r^2$ relationship provides a good fit for all three datasets and merges the stress- and strain-controlled laboratory testing results.

D.8 References

ASTM (2000), "Standard test methods for maximum index density and unit weight of soils using a vibratory table," Designation: D4253-00, ASTM International, West Conshohocken, PA.

ASTM (2000), "Standard test methods for minimum index density and unit weight of and calculation of relative density," Designation: D4254-00, ASTM International, West Conshohocken, PA.

Chiou, B., Darragh, R., Gregor, N., and Silva, W. (2008). "NGA Project Strong-Motion Database." *Earthquake Spectra*, 24(1), 23–44.

Efron, B., and Tibshirani, R. J. (1994). *An Introduction to the Bootstrap.* CRC press.

Finn, W. D. L., and Vaid, Y. P. (1977). "Liquefaction potential from drained constant volume cyclic simple shear tests." *Proceedings of the 6th World Conference on Earthquake Engineering*, New Delhi, India, 10–14.

Finn, W. D. ., Vaid, Y. P., and Bhatia, S. K. (1979). "Constant volume cyclic simple shear testing." *Proceedings of the Second International Conference on Microzonation for Safer Construction - Research and Application*, 839 – 851.

Green, R. A. (2001). "Energy-based evaluation and remediation of liquefiable soils." Ph.D., Virginia Polytechnic Institute and State University, Blacksburg, Virginia.

Kuerbis, R., and Vaid, Y. P. (1988). "Sand sample preparation-the slurry deposition method." *Soils and Foundations*, 28(4), 107–118.

Lee, J. (2009). "Engineering characterization of earthquake ground motions." PhD Dissertation, University of Michigan, Ann Arbor, MI.

Riemer, M. F., Gookin, W. ., Bray, J. B., and Arango, I., I. (1994). "Effects of loading frequency and control on the liquefaction behavior of clean sands." Report No. UCB/GT/94-07, University of California, Berkeley, Berkeley, CA.

Sivathayalan, S., and Ha, D. (2011). "Effect of static shear stress on the cyclic resistance of sands in simple shear loading." *Canadian Geotechnical Journal*, 48(10), 1471–1484.

Vaid, Y. P., and Negussey, D. (1984). "Relative density of pluviated sand samples." *Soils and Foundations*, 24(2), 101–105.

Vaid, Y. P., and Negussey, D. (1988). "Preparation of reconstituted sand specimens." *Advanced Triaxial Testing of Soil and Rock*, ASTM STP, 977, 405–417.

Vaid, Y. P., and Sivathayalan, S. (1996). "Static and cyclic liquefaction potential of Fraser Delta sand in simple shear and triaxial tests." *Canadian Geotechnical Journal*, 33(2), 281–289.

Welch, B. L. (1947). "The generalization of"Student's" problem when several different population variances are involved," *Biometrika*, 34(1–2), 28–35.

D.9 Tables

Table D.1: Index Properties of the Sands Tested.

Property	Quartz Sand	Monterey 0/30
e_{min}	0.68	0.53
e_{max}	0.90	0.845
G_s (assumed)	2.65	2.66
D_{50} (mm)	0.58	0.59
C_u	1.90	1.87
$\gamma_{d,min}$ g/cm^3	1.39	1.44
$\gamma_{d,max}$ g/cm^3	1.58	1.74

Table D.2: Mean and Standard Deviations of Regression Coefficients for the $\Delta W_{\text{Effect.}}/\sigma'_{v0}$-$D_r$ Relationship Obtained Via Bootstrapping

Test Type	$\overline{\beta_0}$	s_{β_0}	$\overline{\beta_1}$	s_{β_1}
Stress	-5.41	0.127	0.0005098	8.002e-05
Strain	-5.39	0.123	0.0005126	3.242e-05
Combined	-5.42	0.092	0.0005126	4.427e-05

Table D.3: Results of Welch's t-tests

	t	p-value	Result
β_0	-1.97	0.0501	Cannot Reject H_0
β_1	-0.446	0.656	Cannot Reject H_0

D.10 Figures

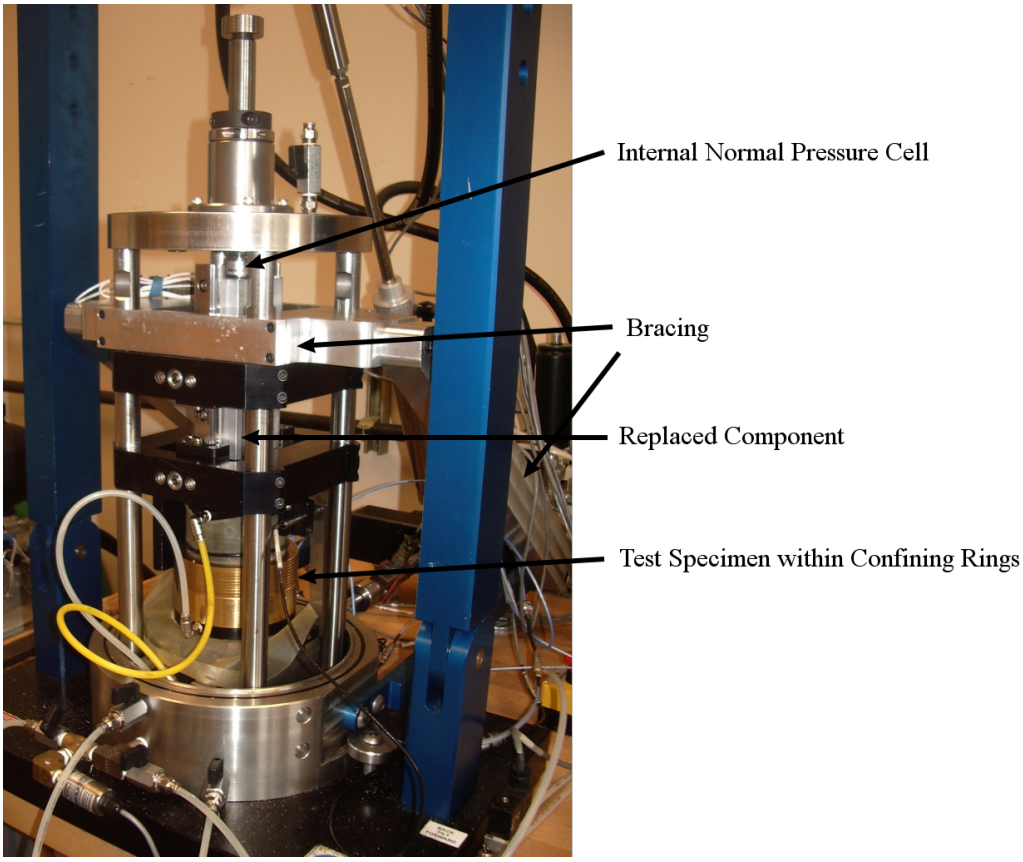


Figure D.1: GCTS equipment with CSS specimen and modifications.

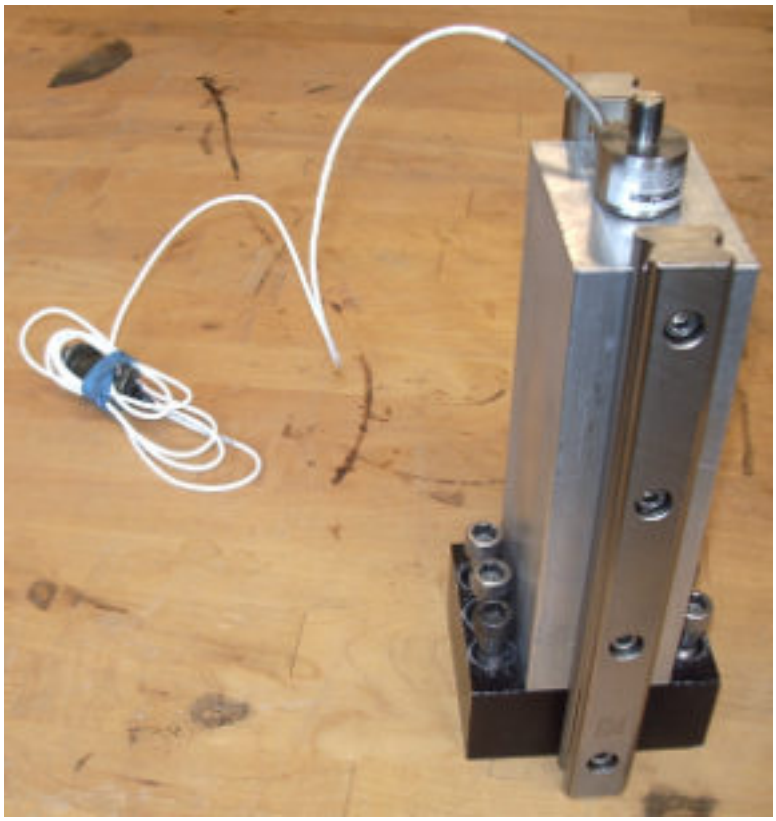


Figure D.2: Part of the top platen mount that has been modified. The silver-colored block is new; it replaces a larger (taller) block to allow room for the internal normal pressure cell (shown).

Figure D.3: Flask, funnel, and stopper used for dry pluviation.

Figure D.4: Removing excess sand from a liquefaction-evaluation specimen.

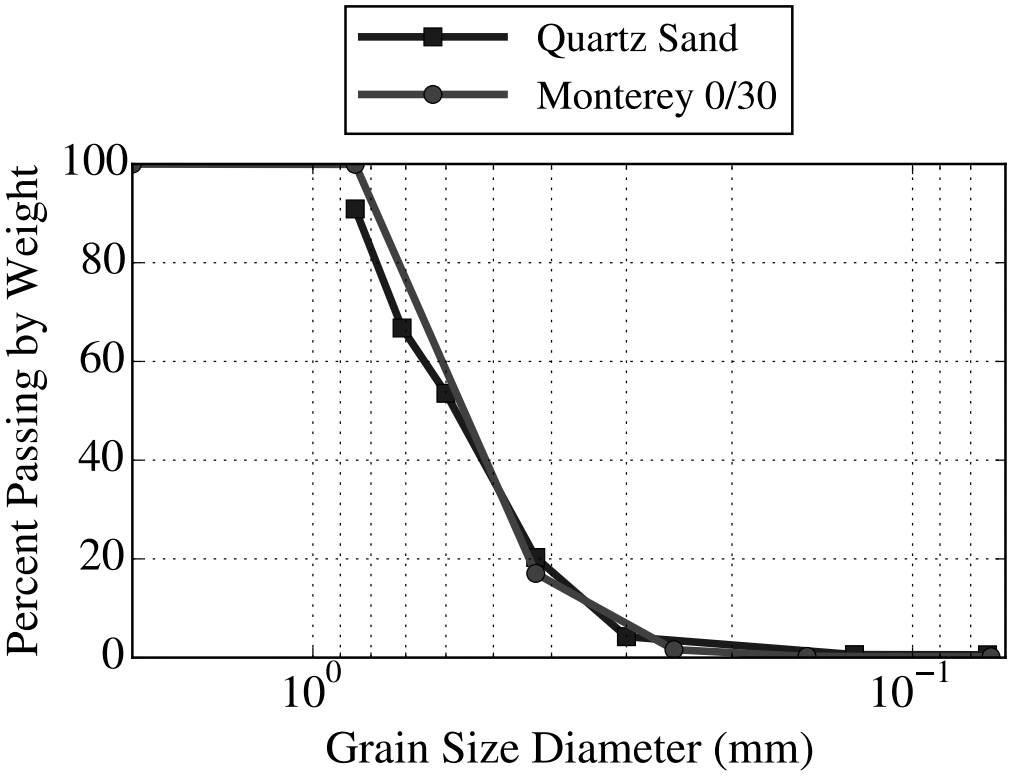


Figure D.5: Gradations of the sands used in cyclic simple shear testing.

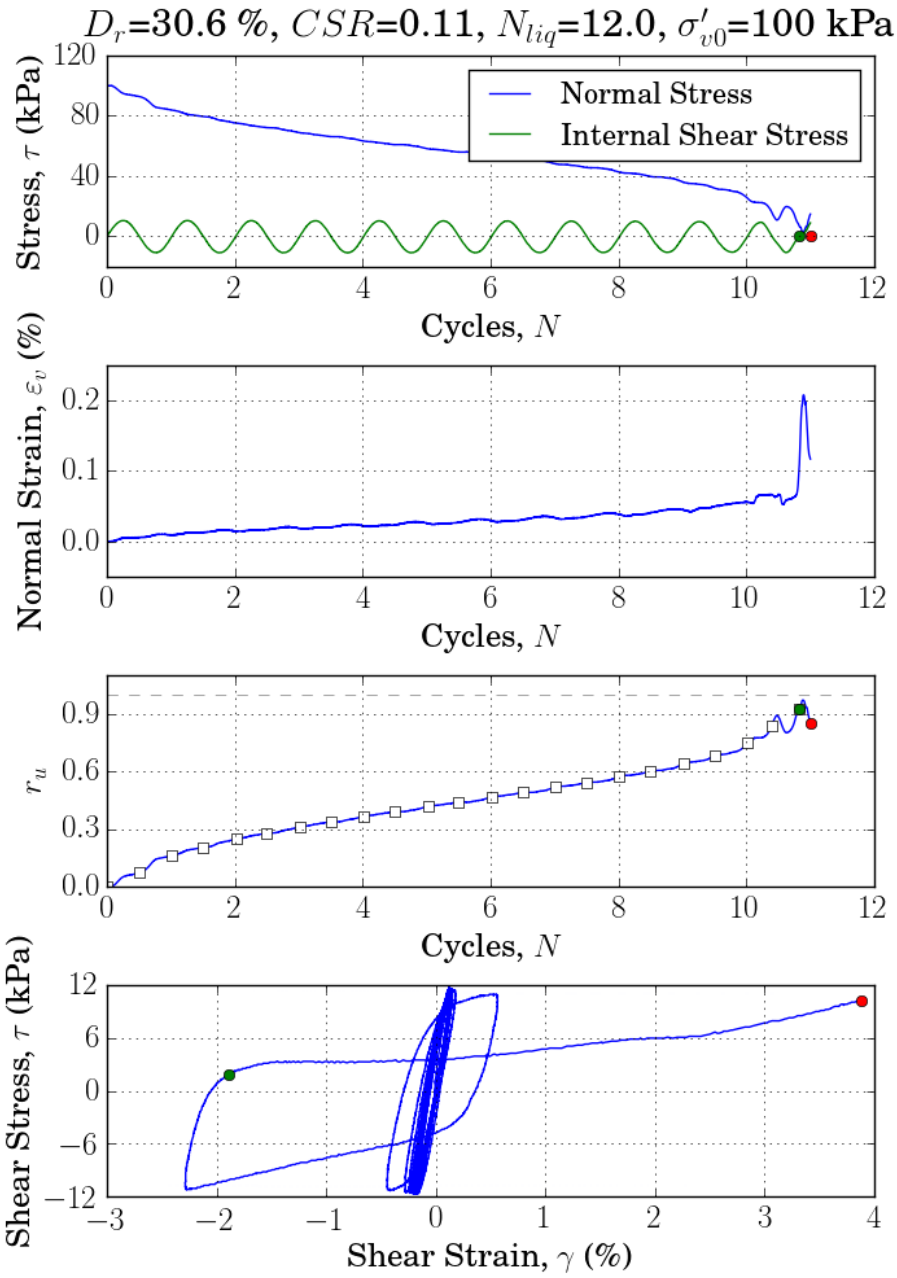


Figure D.6: Typical results for a stress-controlled sinusoidal test on a loose specimen.

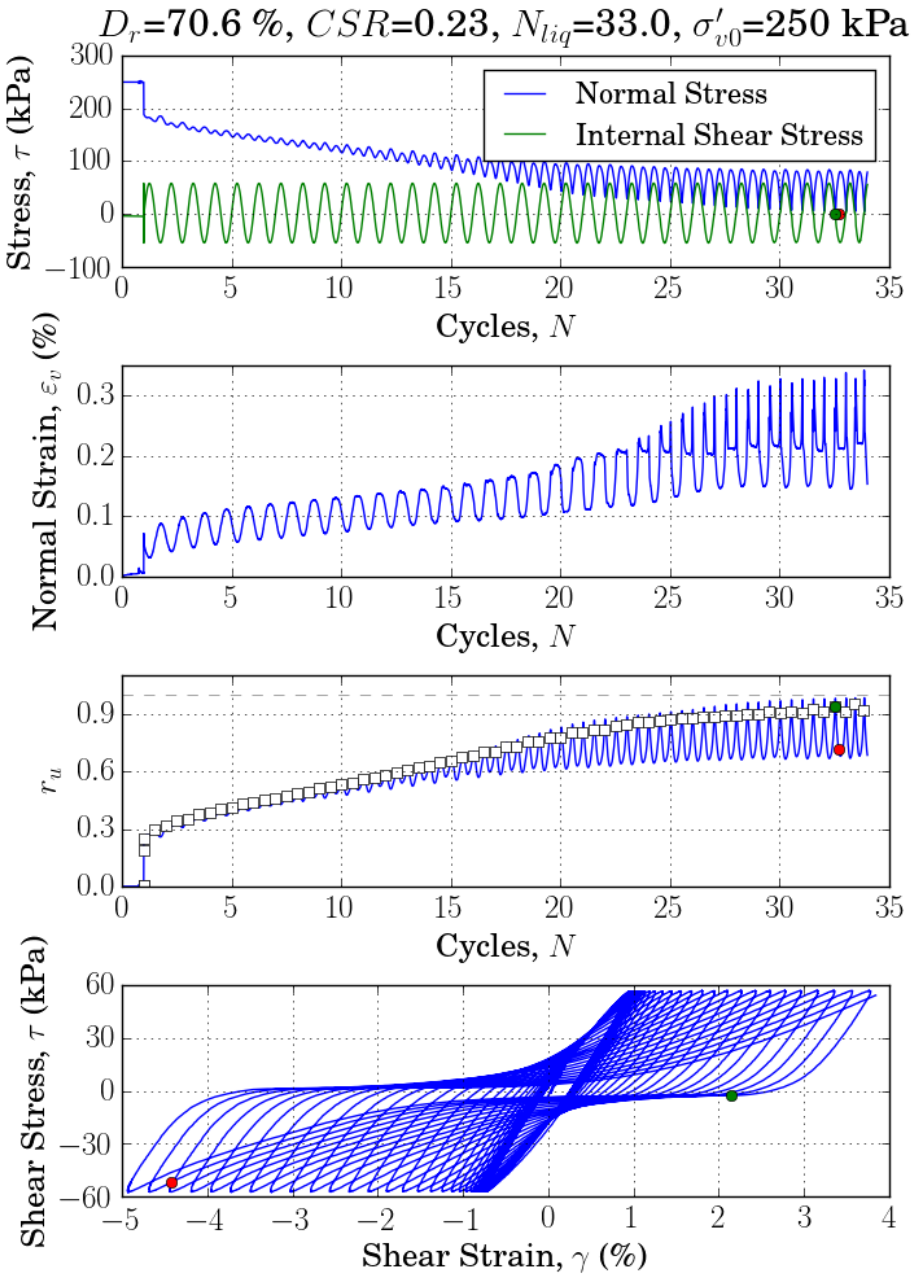


Figure D.7: Typical results for a stress-controlled sinusoidal test on a dense specimen.

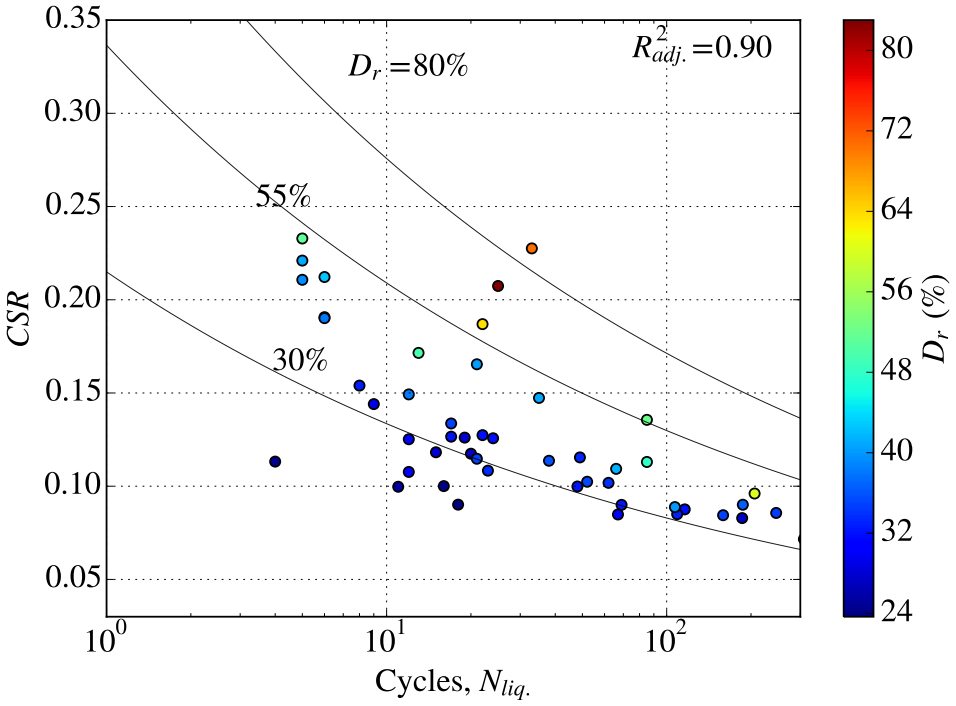


Figure D.8: Cyclic stress ratio, CSR, versus number of cycles to liquefaction, $N_{liq.}$ for stress-controlled sinusoidal tests.

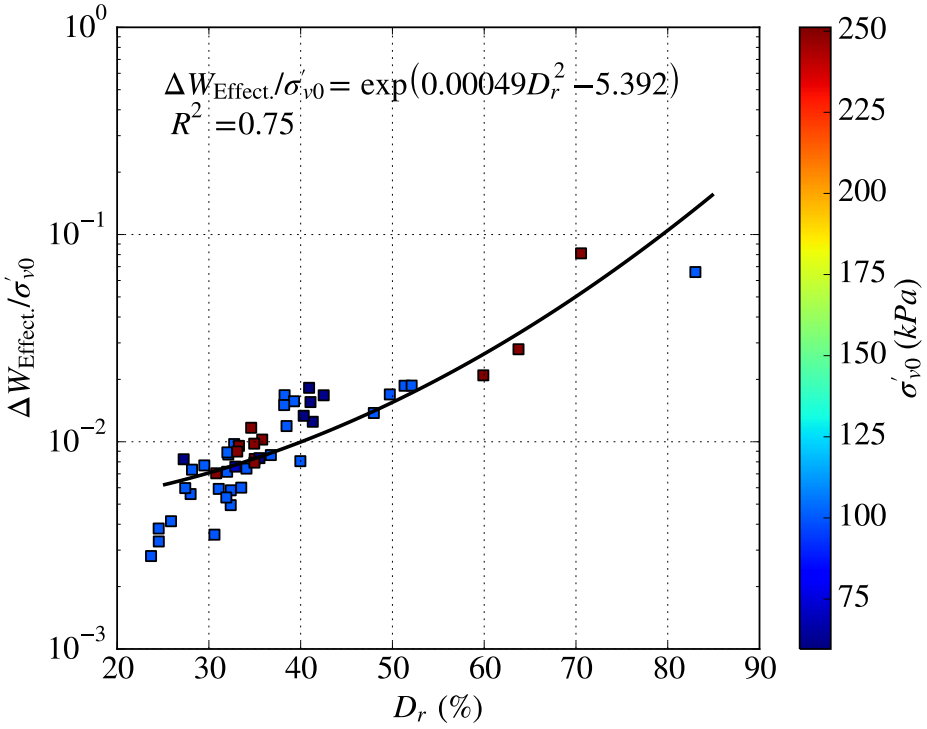


Figure D.9: Normalized dissipated energy, $\Delta W_{\text{Effect.}}/\sigma'_{v0}$, versus relative density, D_r, for stress-controlled sinusoidal tests.

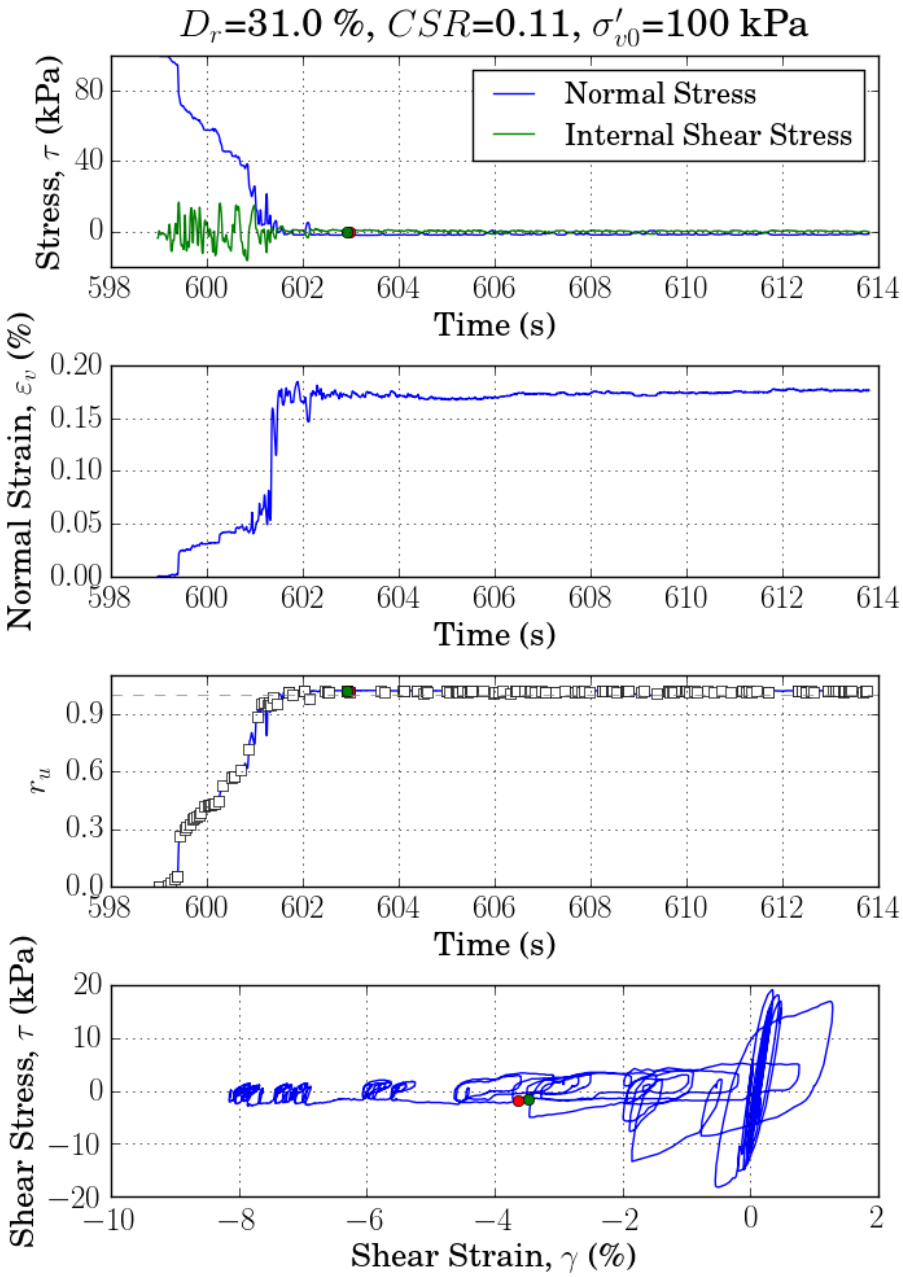


Figure D.10: Typical results for a stress-controlled earthquake-loading test.

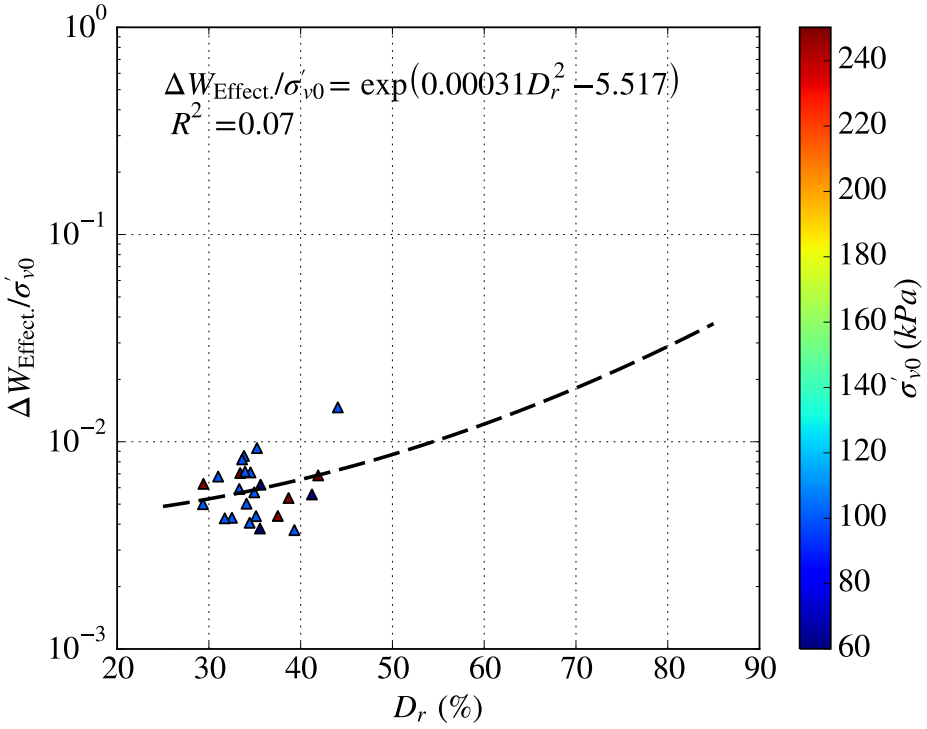


Figure D.11: Normalized dissipated energy, $\Delta W_{\text{Effect.}}/\sigma'_{v0}$, versus relative density, D_r, for stress-controlled earthquake-loading tests.

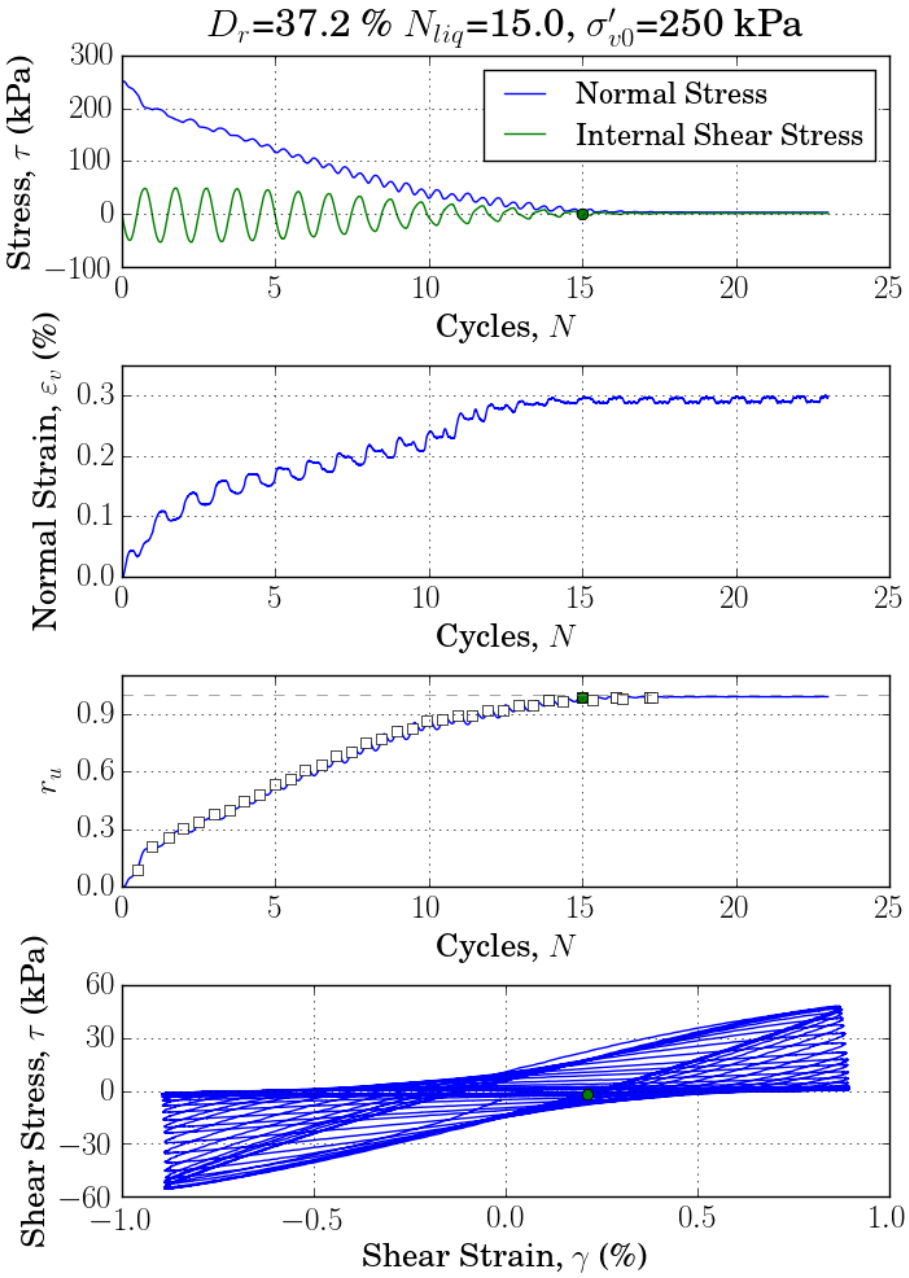


Figure D.12: Typical results for a strain-controlled sinusoidal test on a loose specimen.

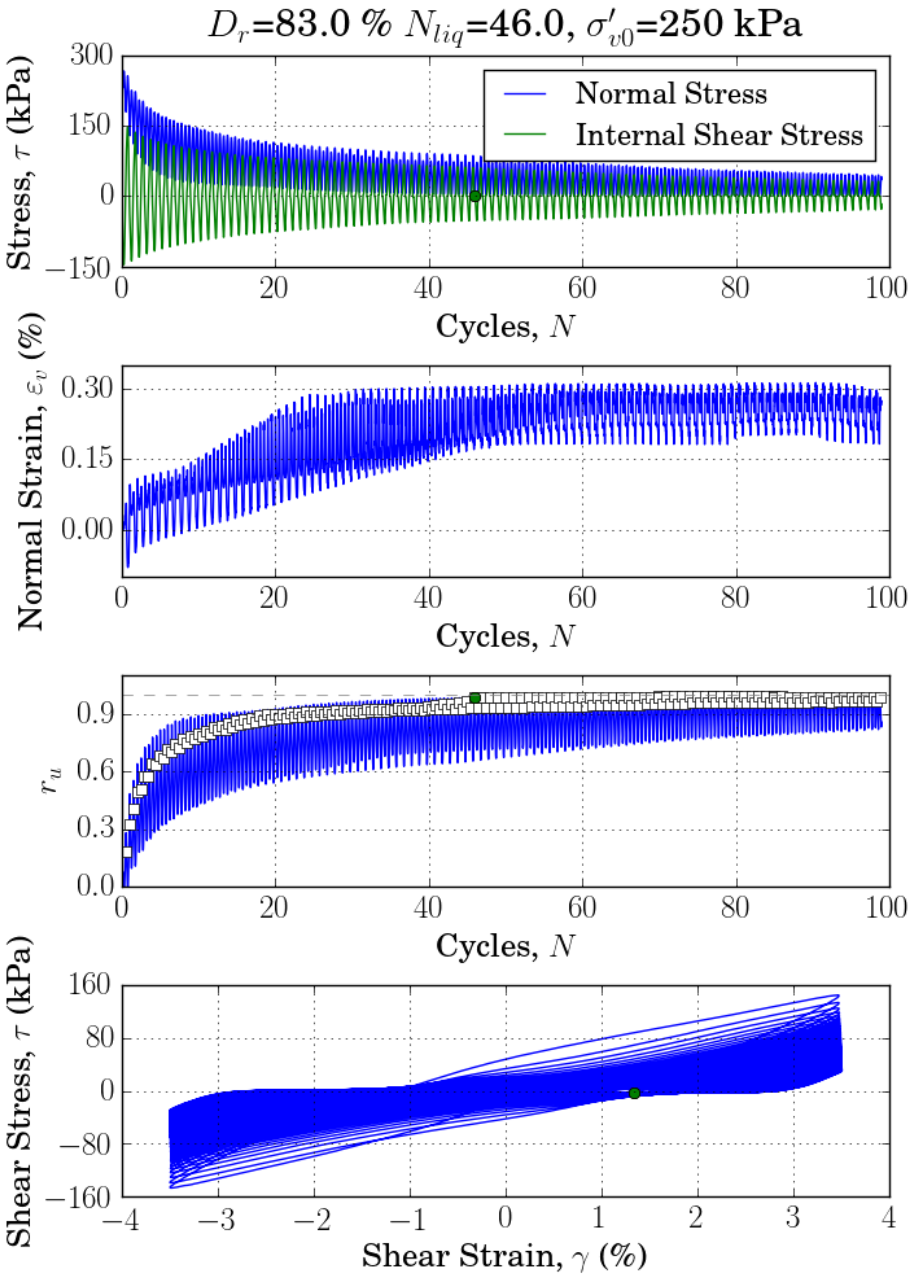


Figure D.13: Typical results for a strain-controlled sinusoidal test on a dense specimen.

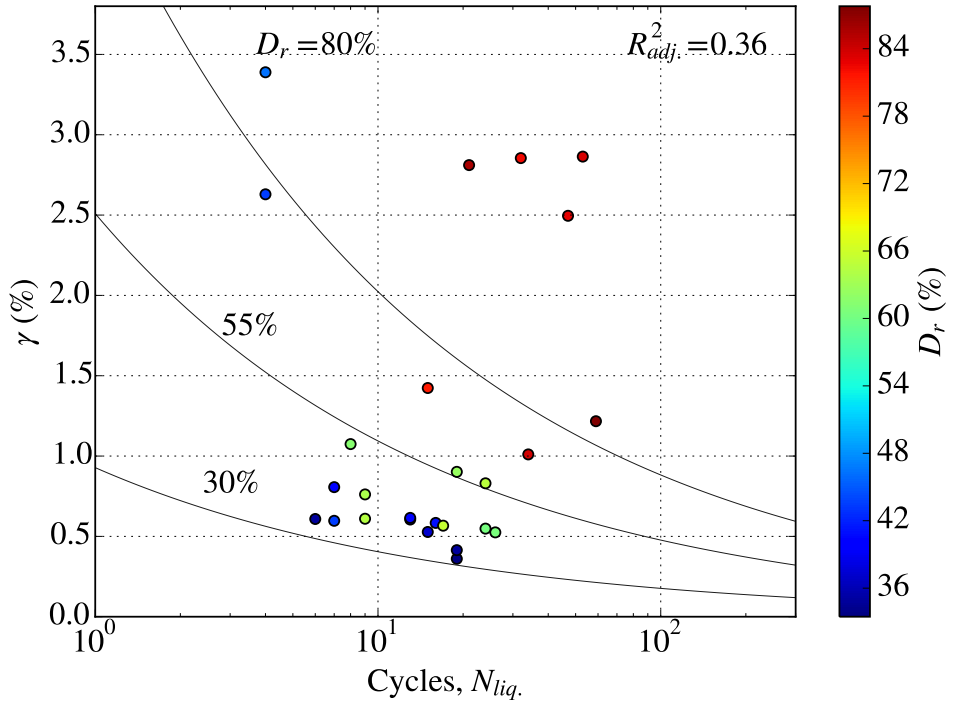


Figure D.14: Amplitude of cyclic shear strain, γ, versus number of cycles to liquefaction, $N_{liq.}$ for strain-controlled sinusoidal tests.

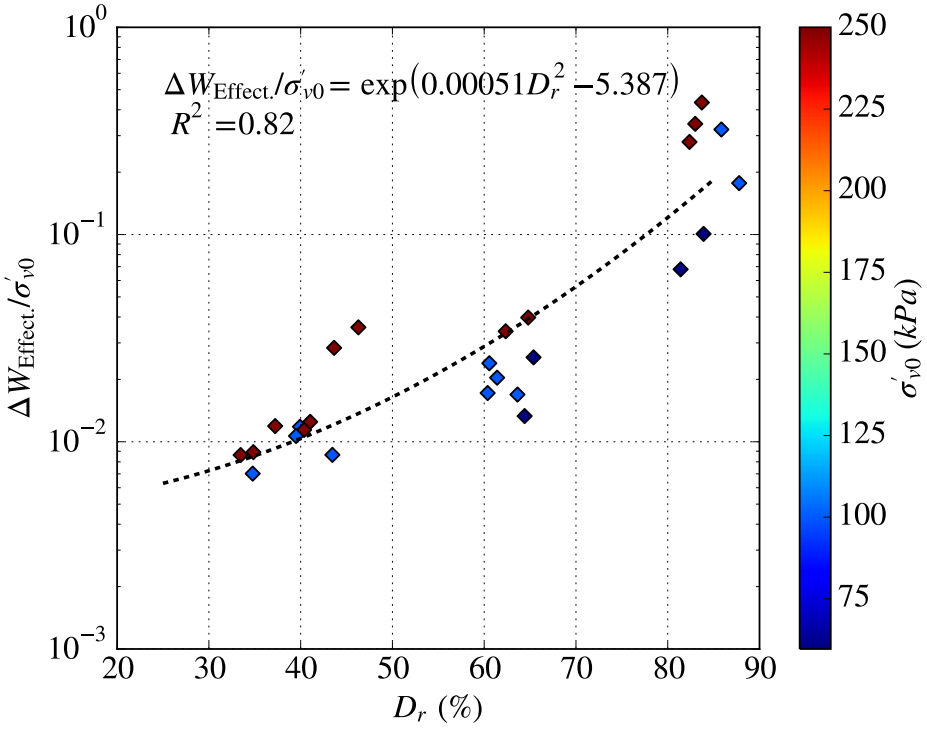


Figure D.15: Normalized dissipated energy, $\Delta W_{\text{Effect.}}/\sigma'_{v0}$, versus relative density, D_r, for strain-controlled sinusoidal tests.

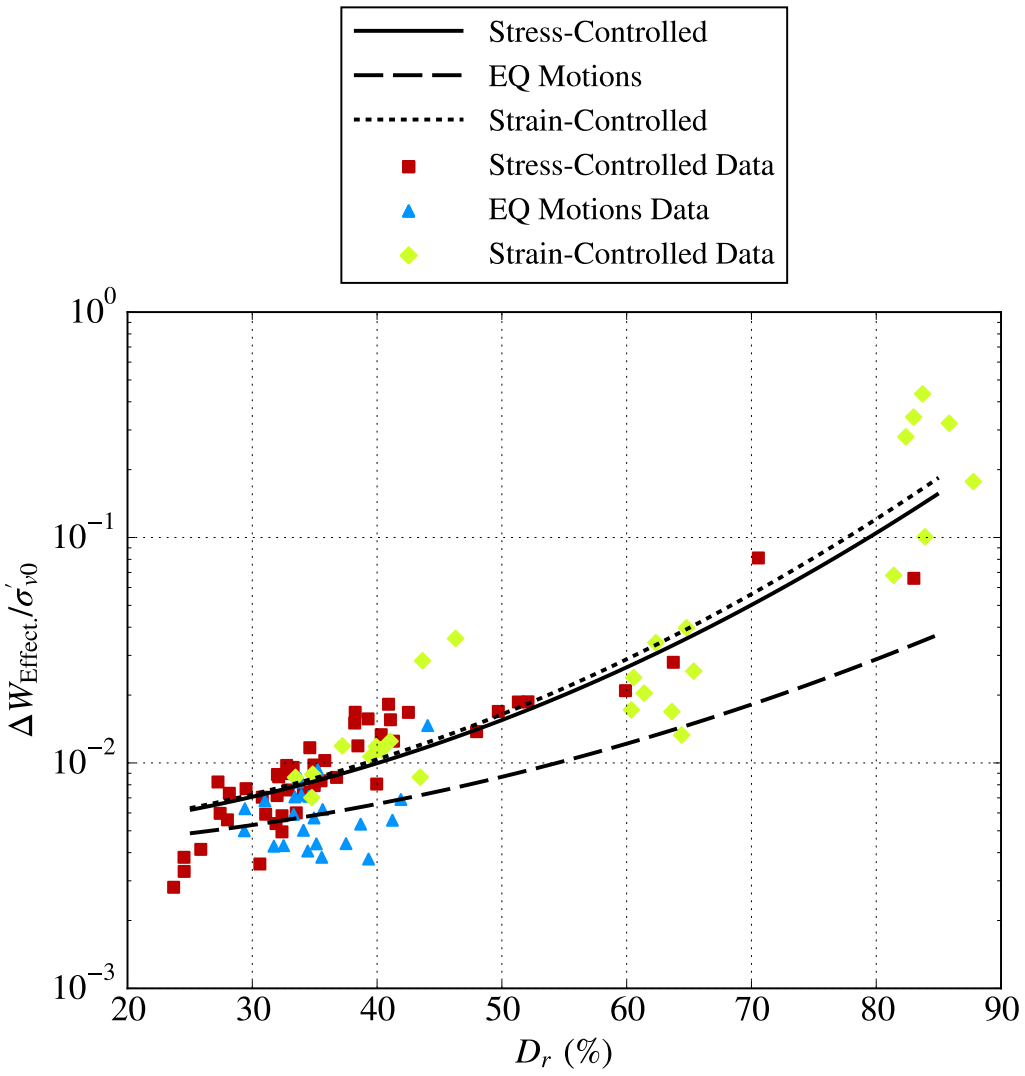


Figure D.16: Normalized dissipated energy, $\Delta W_{\text{Effect.}}/\sigma'_{v0}$, versus relative density, D_r, for all tests from this study.

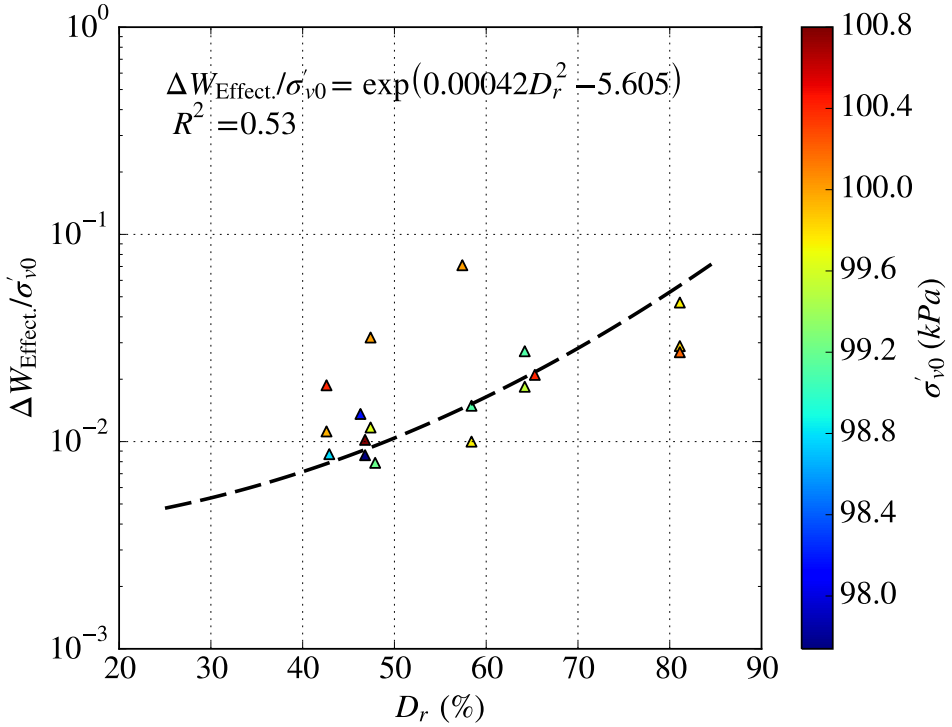


Figure D.17: Normalized dissipated energy, $\Delta W_{\text{Effect.}}/\sigma'_{v0}$, versus relative density, D_r, for cyclic triaxial tests performed on Monterey sand (from Green 2001).

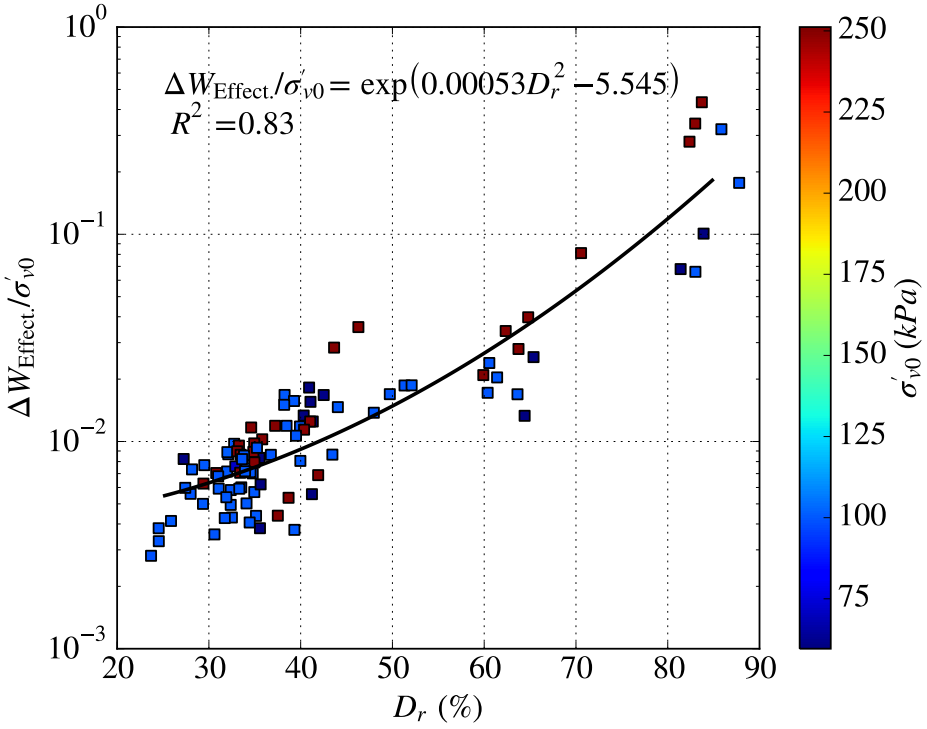


Figure D.18: Normalized dissipated energy, $\Delta W_{\text{Effect.}}/\sigma'_{v0}$, versus relative density, D_r, for all tests performed as part of the liquefaction triggering program.

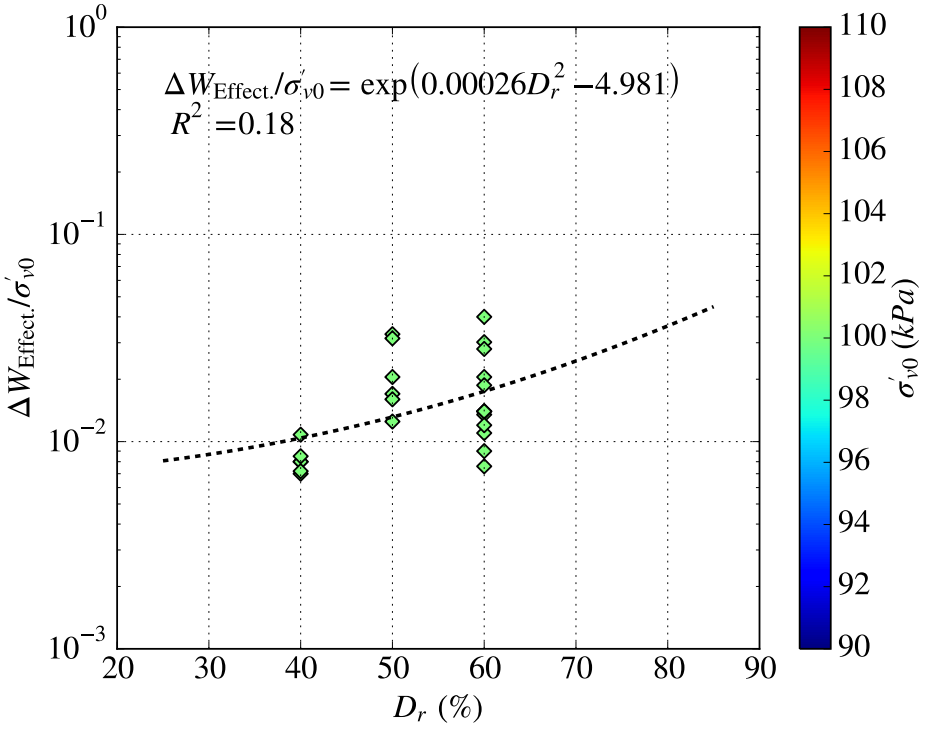


Figure D.19: Normalized dissipated energy, $\Delta W_{\text{Effect.}}/\sigma'_{v0}$, versus relative density, D_r, for cyclic triaxial tests performed on Monterey #0 sand (Riemer et al. 1994).

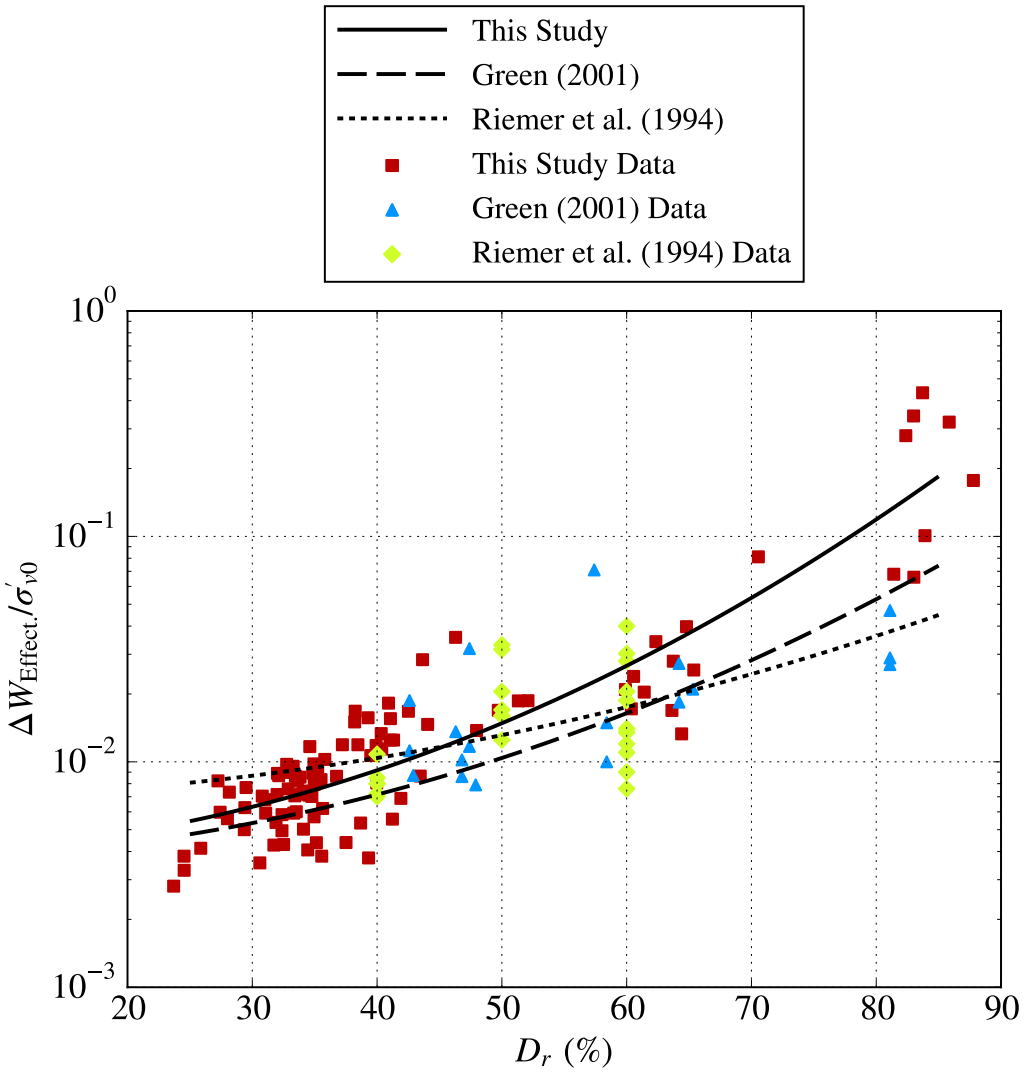


Figure D.20: Normalized dissipated energy, $\Delta W_{\text{Effect.}}/\sigma'_{v0}$, versus relative density, D_r, for all laboratory testing (this study; Green 2001; Riemer et al. 1994).

–>

D.11 Sinusoidal Loading Results

Table D.4 summarizes the results of the stress-controlled sinusoidal tests. In the table heading, D_r is the relative density of the sample (in percent) at the beginning of the cyclic phase of testing, CSR is the cyclic stress ratio of the loading, σ'_{v0} is the effective overburden stress felt by the sample at the beginning of the cyclic phase in kPa, $N_{liq.}$ is the number of cycles to 'liquefaction' defined as 3.75% single amplitude shear strain, $\Delta W_{\text{Effect.}}$ is the dissipated energy up to initial liquefaction in kPa (calculated from effective stress analyses), and ΔW_{Total} is the dissipated energy up to liquefaction corrected to remove effects of softening (i.e. corresponding to total stress analyses; see Green 2001[1]).

Table D.4: Results of stress-controlled sinusoidal CV-CSS tests.

No.	D_r	CSR	σ'_{v0}	$N_{liq.}$	$\Delta W_{\text{Effect.}}$	ΔW_{Total}	Name
1	51.3	0.23	100	5	1.9	0.49	2014-10-11-2-dss
2	49.7	0.17	100	13	1.7	0.35	2014-10-11-3-dss
3	52.1	0.14	100	85	1.9	0.51	2014-10-14-2-dss
4	42.5	0.21	61	6	1	0.31	2014-10-17-1-dss
5	41.3	0.11	60	66	0.75	0.14	2014-10-21-4-dss
6	48.0	0.11	100	85	1.4	0.34	2014-10-14-1-dss
7	40.3	0.17	60	21	0.8	0.19	2014-10-17-2-dss
8	70.6	0.23	250	33	20	4	2014-11-25-1-dss
9	32.4	0.09	100	69	0.5	0.19	2014-05-30-2-dss
10	32.0	0.09	100	116	0.72	0.24	2014-05-30-3-dss
11	36.8	0.09	100	187	0.86	0.37	2014-06-02-1-dss
12	23.7	0.09	99	18	0.28	0.094	2014-06-02-3-dss
13	33.5	0.11	100	23	0.6	0.16	2014-06-04-1-dss
14	30.6	0.11	100	12	0.36	0.1	2014-06-04-2-dss
15	32.4	0.1	100	48	0.58	0.17	2014-06-05-1-dss
16	25.9	0.1	100	11	0.41	0.066	2014-06-05-2-dss
17	24.5	0.1	99	16	0.33	0.093	2014-06-05-3-dss
18	24.5	0.11	100	4	0.38	0.072	2014-06-09-1-dss
19	28.0	0.12	100	15	0.56	0.16	2014-06-09-2-dss
20	27.4	0.12	100	20	0.6	0.17	2014-06-09-3-dss
21	31.1	0.13	100	12	0.59	0.15	2014-06-11-1-dss
22	32.1	0.13	100	17	0.87	0.19	2014-06-11-2-dss
23	28.2	0.13	100	19	0.74	0.19	2014-06-11-4-dss
24	29.5	0.14	100	9	0.77	0.19	2014-06-12-2-dss

[1]Green, R. A. (2001). "Energy-Based Evaluation and Remediation of Liquefiable Soils." Ph.D., Virginia Polytechnic Institute and State University, Blacksburg, Virginia.

No.	D_r	CSR	σ'_{v0}	$N_{liq.}$	$\Delta W_{\text{Effect.}}$	ΔW_{Total}	Name
25	32.8	0.15	100	8	0.98	0.21	2014-06-13-2-dss
26	35.0	0.13	251	17	2.1	0.76	2014-07-03-1-dss
27	34.1	0.1	100	62	0.74	0.19	2014-06-05-4-dss
28	32.0	0.13	101	24	0.9	0.24	2014-06-10-3-dss
29	31.9	0.13	100	22	0.54	0.22	2014-06-11-3-dss
30	33.3	0.12	250	49	2.4	0.95	2014-07-07-4-dss
31	33.1	0.08	250	109	2.2	0.92	2014-07-09-4-dss
32	27.3	0.08	60	186	0.49	0.14	2014-07-14-1-dss
33	35.5	0.1	60	52	0.5	0.11	2014-07-16-1-dss
34	35.0	0.11	250	21	2	0.44	2014-07-07-2-dss
35	35.8	0.11	250	38	2.6	0.85	2014-07-09-1-dss
36	30.8	0.08	250	67	1.8	0.64	2014-07-10-1-dss
37	34.6	0.09	250	246	2.9	1.4	2014-07-10-2-dss
38	34.9	0.08	250	159	2.4	1	2014-07-10-3-dss
39	32.9	0.07	72	309	0.54	0.2	2014-07-14-2-dss
40	63.7	0.19	250	22	7	1.7	2014-08-26-4-dss
41	59.9	0.1	251	206	5.3	1.6	2014-09-02-3-dss
42	83.0	0.21	100	25	6.6	5.6	2014-09-12-2-dss
43	38.5	0.15	100	12	1.2	0.22	2014-12-27-11-dss
44	38.2	0.19	100	6	1.7	0.48	2015-1-2-4-dss
45	39.3	0.21	100	5	1.6	0.56	2015-1-2-5-dss
46	41.1	0.22	60	5	0.94	0.17	2014-12-27-10-dss
47	40.9	0.15	60	35	1.1	0.23	2014-12-27-4-dss
48	38.2	0.19	100	6	1.5	0.2	2015-1-2-3-dss
49	40.0	0.09	100	107	0.81	0.24	2014-05-30-1-dss

D.12 Earthquake Loading Results

Table D.5 summarizes the results of stress-controlled tests that use earthquake motions as the loading function. The stress time history used as the loading function was obtained by applying an earthquake acceleration time history at the base of a simple sand profile (30 m thick) and performing an equivalent-linear site response analysis. The stress time history is the response at an effective overburden stress of 100 kPa. The columns of Table D.5 are defined as follows: D_r is relative density of the specimen at the beginning of the cyclic phase (%), CSR is defined as $0.65 \cdot \tau_{max}/\sigma'_{v,i}$ of the recorded stress, σ'_{v0} is the initial effective overburden stress, $\Delta W_{\text{Effect.}}$ is the dissipated energy up to initial liquefaction in kPa, ΔW_{Total} is the dissipated energy up to liquefaction corrected to remove effects of softening (kPa) (i.e. corresponding to total stress analyses), $r_{u,final}$ is the 'synthetic' excess pore pressure ratio at failure obtained from the effective overburden stress ($r_u = (\sigma'_{v0} - \sigma'_v)/\sigma'_{v0}$), and 'Motion' is the name of the input time history. If the motion name ends in '_inv', the stress time history was multiplied by -1, i.e. it was inverted. For these results, failure or initial liquefaction was considered to have occurred when the single-amplitude shear strain exceeded 3.75 percent. Thus, $r_{u,final}$ is not equal to one in all cases.

Table D.5: Results of Stress-Controlled Earthquake-Loading Constant Volume Cyclic Simple Shear Tests

No.	D_r	CSR	σ'_{v0}	$\Delta W_{\text{Effect.}}$	ΔW_{Total}	$r_{u,final}$	Motion/Filename
1	31.0	0.11	100	0.68	nan	1.02	NGA_no_87_SAD273 EQ2014-07-28-1-dss
2	33.3	0.11	100	0.59	nan	1.01	NGA_no_87_SAD273 EQ2014-07-30-2-dss
3	29.4	0.1	100	0.5	nan	1.01	NGA_no_87_SAD273 EQ2014-07-28-2-dss
4	35.2	0.13	100	0.93	nan	1.12	NGA_no_87_SAD273 EQ2014-07-30-3-dss
5	33.8	0.13	100	0.86	nan	1.01	NGA_no_87_SAD273 EQ2014-07-31-3-dss
6	32.5	0.1	100	0.43	nan	0.99	NGA_no_87_SAD273 EQ2014-08-01-2-dss
7	38.7	0.12	250	1.3	nan	0.83	NGA_no_87_SAD273 EQ2014-08-11-3-dss
8	29.4	0.1	250	1.6	nan	0.94	NGA_no_87_SAD273 EQ2014-08-12-3-dss
9	37.5	0.12	250	1.1	nan	0.84	NGA_no_87_SAD273 EQ2014-08-14-1-dss
10	31.7	0.1	100	0.43	nan	1.01	NGA_no_87_SAD273 EQ2014-07-31-2-dss
11	35.1	0.11	100	0.44	nan	0.97	NGA_no_87_SAD273 EQ2014-08-01-3-dss

No.	D_r	CSR	σ'_{v0}	$\Delta W_{\text{Effect.}}$	ΔW_{Total}	$r_{u,final}$	Motion/Filename
12	33.6	0.12	100	0.82	0.61	1.02	NGA_no_87_SAD273 EQ2014-08-07-2-dss
13	33.4	0.12	250	1.8	0.01	0.90	NGA_no_87_SAD273 EQ2014-08-11-2-dss
14	41.9	0.16	250	1.7	0.00058	0.71	NGA_no_87_SAD273 EQ2014-08-15-1-dss
15	41.2	0.1	60	0.33	0.38	0.99	NGA_no_87_SAD273 EQ2014-08-18-1-dss
16	35.6	0.08	60	0.23	0.0045	0.98	NGA_no_87_SAD273 EQ2014-08-20-1-dss
17	34.5	0.09	100	0.71	nan	1.00	NGA_no_87_SAD273 EQ2014-07-30-1-dss
18	33.9	0.11	100	0.72	nan	1.01	NGA_no_87_SAD273 EQ2014-07-31-1-dss
19	39.3	0.11	100	0.38	nan	0.97	NGA_no_87_SAD273 EQ2014-07-31-4-dss
20	44.1	0.17	100	1.5	nan	0.93	NGA_no_87_SAD273 EQ2014-08-01-dss
21	34.1	0.11	100	0.5	0.0032	0.92	NGA_no_87_SAD273 EQ2014-08-07-1-dss
22	34.4	0.09	100	0.41	0.0029	0.83	NGA_no_87_SAD273 EQ2014-08-07-3-dss
23	34.9	0.1	100	0.57	0.52	0.91	NGA_no_87_SAD273 EQ2014-08-08-1-dss
24	35.6	0.12	60	0.37	0.0066	0.88	NGA_no_87_SAD273 EQ2014-08-20-2-dss

D.13 Strain-Controlled Sinusoidal Loading Results

Table D.6 summarizes the results of strain-controlled sinusoidal constant volume cyclic simple shear tests. Failure or initial liquefaction was determined as the static excess pore pressure ratio exceeded 98% ($r_u \geq 0.98$ when $\tau_{cyclic} = 0$). The cutoff of 0.98% was chosen because loose specimens tended to reach $r_u = 1$ without a problem, but denser specimens had trouble reaching $r_u = 1$ despite very low shear stress response. The columns of Table D.6 are defined as follows: D_r is relative density of the specimen at the beginning of the cyclic phase (%), γ is the single-amplitude shear strain of the loading (%), σ'_{v0} is the initial effective overburden stress (kPa), $\Delta W_{\text{Effect.}}$ is the dissipated energy up to initial liquefaction (kPa) corresponding to effective stress analyses. The dissipated energy values of strain-controlled tests do not need to be adjusted to account for softening of the soil as stress-controlled tests do.

Table D.6: Results of Strain-Controlled CV-CSS Tests

No.	D_r	γ	σ'_{v0}	$\Delta W_{\text{Effect.}}$	Name
1	39.9	0.60	100	1.2	STRAIN2015-01-20-01-dss
2	60.6	0.52	100	2.4	STRAIN2015-01-21-4-dss
3	43.5	0.60	100	0.86	STRAIN2015-01-20-02-dss
4	60.4	0.55	100	1.7	STRAIN2015-01-21-3-dss
5	33.5	0.36	250	2.2	STRAIN2015-01-23-1-dss
6	41.0	0.58	250	3.1	STRAIN2015-01-28-1-dss
7	83.0	2.50	250	85	STRAIN2015-02-02-2-dss
8	65.4	0.57	60	1.5	STRAIN2015-02-13-1-dss
9	85.8	2.81	100	32	STRAIN2015-03-27-6-dss
10	87.8	1.22	100	18	STRAIN2015-01-22-1-dss
11	34.8	0.41	250	2.2	STRAIN2015-01-23-2-dss
12	40.4	0.62	250	2.8	STRAIN2015-01-23-3-dss
13	62.3	0.90	250	8.5	STRAIN2015-01-28-2-dss
14	64.8	0.83	250	9.9	STRAIN2015-01-28-3-dss
15	82.4	2.85	249	70	STRAIN2015-02-02-3-dss
16	83.7	2.86	250	1.1e+02	STRAIN2015-02-02-4-dss
17	63.6	0.76	100	1.7	STRAIN2015-03-27-1-dss
18	61.4	1.07	100	2	STRAIN2015-03-27-2-dss
19	34.8	0.61	100	0.7	STRAIN2015-03-27-3-dss
20	39.5	0.81	100	1.1	STRAIN2015-03-27-4-dss
21	64.4	0.61	60	0.8	STRAIN2015-04-03-1-dss
22	83.9	1.01	60	6	STRAIN2015-04-03-3-dss
23	81.4	1.42	60	4.1	STRAIN2015-04-03-4-dss
24	37.2	0.53	250	3	STRAIN2015-04-04-1-dss
25	46.3	3.39	250	8.9	STRAIN2015-04-04-3-dss
26	43.6	2.63	250	7.1	STRAIN2015-04-04-4-dss

Appendix E

Soil Profiles of the Equivalent-Linear Analyses

Fifty profiles from Cetin (2000)[1] were used for equivalent-linear site response analyses. These profiles were collected from case histories of a number of different different earthquake events. The shear wave velocity of the bedrock layers were increased for use with central-eastern United States (CEUS) motions. For more information, see Chapter 4.

E.1 Profile000 - Balboa Boulevard-Northridge Earthquake (1994)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty sand	1.1	11.0	17.7	147.0	147.0
silty sand	1.1	14.0	19.1	160.0	160.0
sandy clay	1.3	15.0	15.8	134.0	134.0
sandy clay	1.3	19.0	15.3	152.0	152.0
silty sand	1.1	16.0	19.8	190.0	190.0
silty sandy clay	1.5	23.0	17.9	170.0	170.0
silty sand	1.0	16.0	17.3	219.0	219.0
silty sand	1.0	15.0	16.3	241.0	241.0
sandy clay	1.1	11.0	17.1	210.0	210.0

[1]Cetin, K. Ö. (2000). "Reliability-Based Assessment of Seismic Soil Liquefaction Initiation Hazard." Ph.D. Dissertation, University of California, Berkeley.

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
silty sand	1.5	13.0	18.0	266.0	266.0
clay	6.0	25.0	14.3	301.0	301.0
weathered shale	14.3	0.0	24.5	411.0	411.0
weathered shale	14.3	0.0	24.4	487.0	487.0
weathered shale	14.3	0.0	24.8	563.0	563.0
weathered shale	14.3	0.0	23.2	602.0	602.0
weathered bedrock	3.0	0.0	22.9	792.0	1188.0
weathered bedrock	3.1	0.0	22.9	838.0	1675.0
weathered bedrock	3.0	0.0	23.4	884.0	2210.0
weathered bedrock	3.1	0.0	22.5	913.0	2740.0

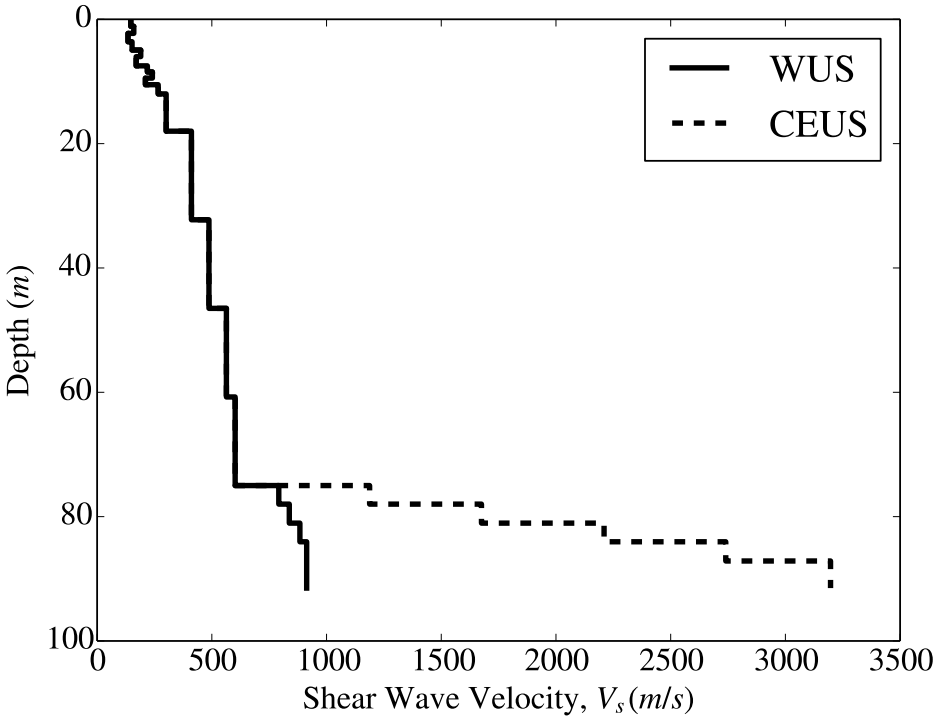


Figure E.1: Shear wave velocities of Profile000

E.2 Profile001 - Malden Street-Northridge Earthquake (1994)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty sand	1.2	15.0	16.7	150.0	150.0
silty sand	1.2	7.0	20.3	130.0	130.0
sandy clay	1.0	33.0	16.7	133.0	133.0
sandy clay	1.0	34.0	17.8	136.0	136.0
sandy clay	2.0	15.0	15.5	140.0	140.0
sandy clay	1.0	10.0	15.8	146.0	146.0
sandy clay	1.0	24.0	17.2	152.0	152.0
silty sand	1.2	17.0	16.4	229.0	229.0
silty sand	1.2	17.0	18.9	240.0	240.0
silty sand	1.2	16.0	16.4	250.0	250.0
clay	6.1	16.0	14.2	301.0	301.0
weathered shale	14.3	0.0	24.9	411.0	411.0
weathered shale	14.3	0.0	24.6	487.0	487.0
weathered shale	14.3	0.0	23.5	563.0	563.0
weathered shale	14.3	0.0	23.2	602.0	602.0
weathered bedrock	3.0	0.0	22.8	792.0	1188.0
weathered bedrock	3.0	0.0	23.0	837.0	1674.0
weathered bedrock	3.1	0.0	22.0	883.0	2209.0
weathered bedrock	3.2	0.0	22.2	914.0	2743.0

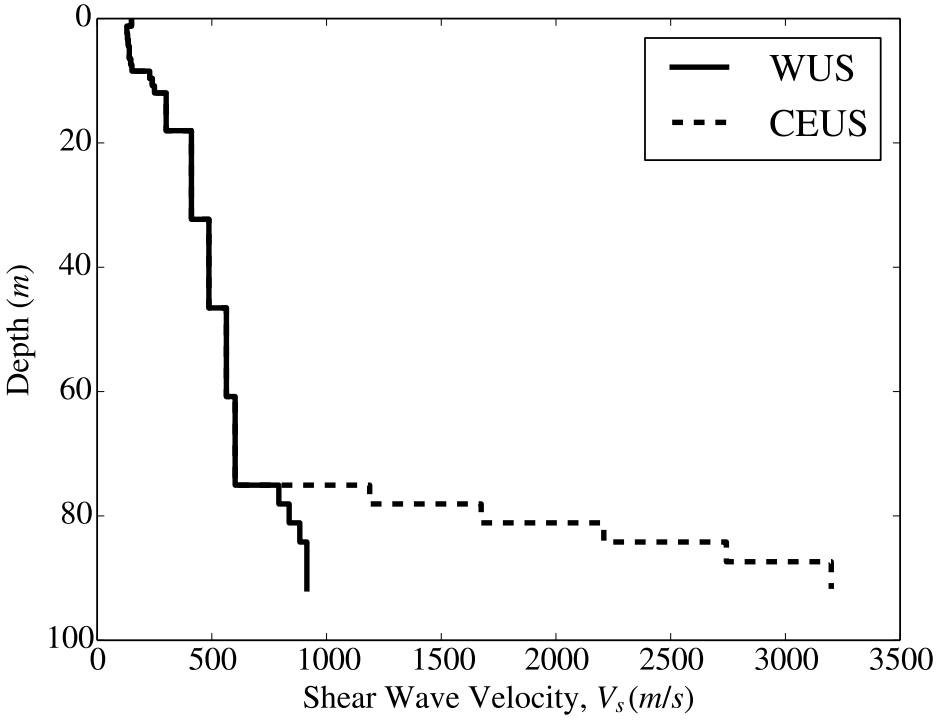


Figure E.2: Shear wave velocities of Profile001

E.3 Profile002 - Wynne Avenue-Northridge Earthquake (1994)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sand	0.8	0.0	19.1	150.0	150.0
clay	0.7	34.0	16.3	140.0	140.0
sand	1.0	0.0	17.9	122.0	122.0
sandy clay	1.2	8.0	15.7	130.0	130.0
sandy clay	1.2	7.0	15.7	141.0	141.0
sandy clay	1.2	21.0	15.1	152.0	152.0
silty sand	1.0	12.0	16.8	188.0	188.0
silty sand	1.0	13.0	18.3	201.0	201.0
clay	3.3	28.0	14.9	235.0	235.0
clay	3.3	24.0	16.2	273.0	273.0
clay	3.3	8.0	15.2	324.0	324.0
weathered shale	14.2	0.0	22.5	411.0	411.0
weathered shale	14.2	0.0	23.3	487.0	487.0
weathered shale	14.2	0.0	22.6	563.0	563.0
weathered shale	14.2	0.0	22.5	601.0	601.0
weathered bedrock	3.0	0.0	22.0	792.0	1188.0
weathered bedrock	3.1	0.0	24.4	837.0	1674.0
weathered bedrock	3.0	0.0	22.2	883.0	2208.0
weathered bedrock	3.0	0.0	22.3	914.0	2742.0

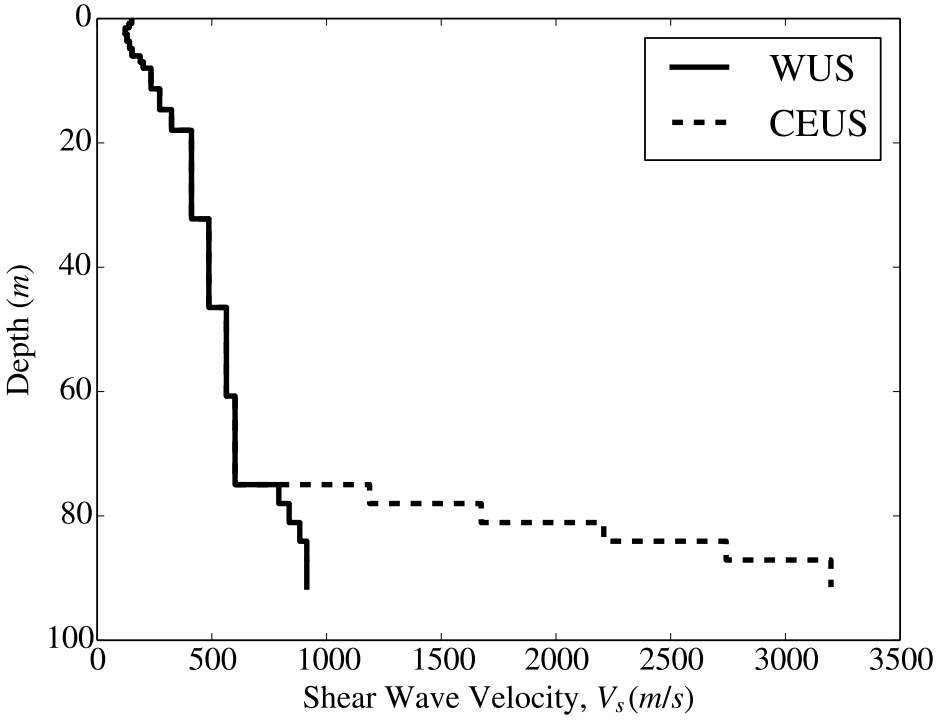


Figure E.3: Shear wave velocities of Profile002

E.4 Profile003 - MBARI NO:3 EB-1-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sand	1.0	0.0	19.6	180.0	180.0
sand	1.0	0.0	19.4	182.0	182.0
sand	1.0	0.0	18.8	186.0	186.0
sand	1.0	0.0	17.4	188.0	188.0
sand	1.0	0.0	17.4	189.0	189.0
sand	1.0	0.0	21.6	191.0	191.0
sand	1.0	0.0	20.6	193.0	193.0
sand	1.0	0.0	19.0	194.0	194.0
sand	1.0	0.0	19.9	197.0	197.0
sand	1.0	0.0	19.0	200.0	200.0
silty clay	1.0	7.0	17.3	180.0	180.0
silty clay	2.0	6.0	14.4	189.0	189.0
silty clay	1.0	7.0	17.1	201.0	201.0
silty clay	3.3	5.0	14.3	290.0	290.0
silty clay	3.3	6.0	16.1	301.0	301.0
silty clay	3.3	7.0	14.5	311.0	311.0
clay	7.1	21.0	15.4	248.0	248.0
fine sand	2.9	0.0	18.1	299.0	299.0
silty clay	5.0	4.0	15.0	352.0	352.0
sandy gravel	19.0	0.0	21.4	498.0	498.0
sandy clay	3.1	8.0	16.8	688.0	688.0
sandy gravel	11.9	0.0	18.9	697.0	697.0
sandy gravelly clay	19.0	27.0	17.7	524.0	524.0
sandy clay	2.1	13.0	15.2	539.0	539.0
gravel	1.0	0.0	22.9	700.0	700.0
sandy clay	28.8	16.0	17.6	709.0	709.0
sandy clay	28.8	9.0	16.8	718.0	718.0
sandy clay	14.4	13.0	16.1	731.0	731.0
weathered bedrock	7.6	0.0	23.6	791.0	1187.0

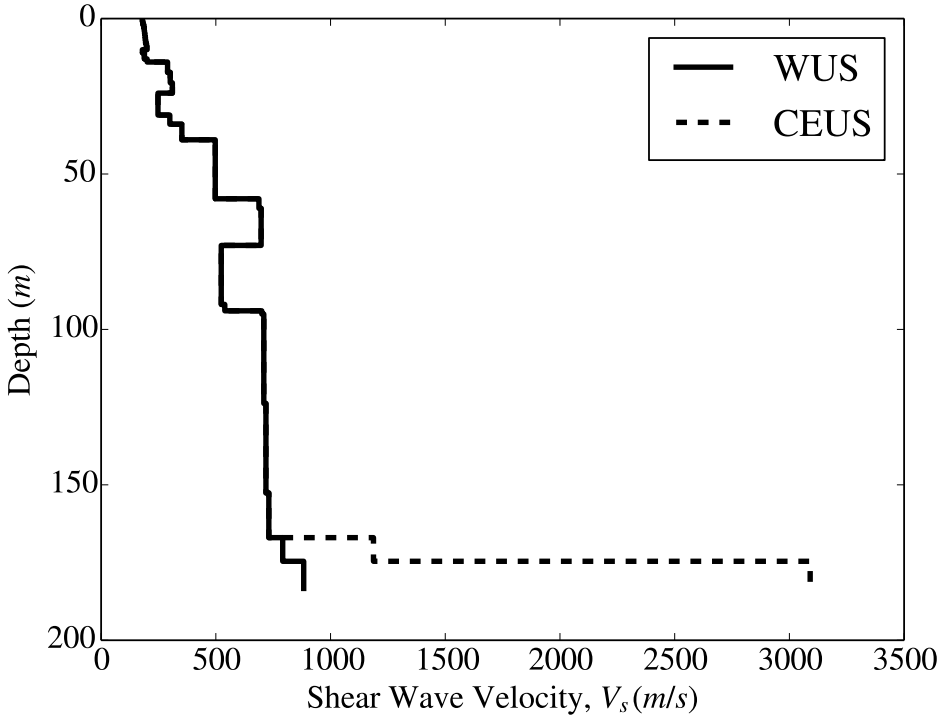


Figure E.4: Shear wave velocities of Profile003

E.5 Profile004 - MBARI NO:3 EB-5-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sand	1.0	0.0	20.2	135.0	135.0
sand	1.0	0.0	21.7	140.0	140.0
sand	1.0	0.0	20.5	146.0	146.0
sand	1.0	0.0	19.8	151.0	151.0
sand	1.0	0.0	20.6	154.0	154.0
sand	1.0	0.0	21.2	158.0	158.0
sand	1.0	0.0	21.9	163.0	163.0
sand	1.0	0.0	19.2	168.0	168.0
sand	1.0	0.0	18.2	174.0	174.0
sand	1.0	0.0	21.7	180.0	180.0
silty clay	1.0	5.0	16.2	159.0	159.0
silty clay	2.0	6.0	16.5	170.0	170.0
silty clay	1.0	5.0	15.9	180.0	180.0
silty clay	3.3	6.0	15.7	290.0	290.0
silty clay	3.3	7.0	15.0	302.0	302.0
silty clay	3.3	5.0	15.3	311.0	311.0
clay	7.1	30.0	16.5	248.0	248.0
fine sand	2.9	0.0	18.5	300.0	300.0
silty clay	5.0	6.0	16.3	352.0	352.0
sandy gravel	19.0	0.0	20.1	498.0	498.0
sandy clay	3.1	8.0	15.4	689.0	689.0
sandy gravel	11.9	0.0	22.6	698.0	698.0
sandy gravelly clay	19.0	21.0	16.1	524.0	524.0
sandy clay	2.1	20.0	15.6	540.0	540.0
gravel	1.0	0.0	21.1	701.0	701.0
sandy clay	28.8	24.0	15.4	710.0	710.0
sandy clay	28.8	23.0	15.0	719.0	719.0
sandy clay	14.4	17.0	17.5	731.0	731.0
weathered bedrock	7.6	0.0	24.6	792.0	1188.0

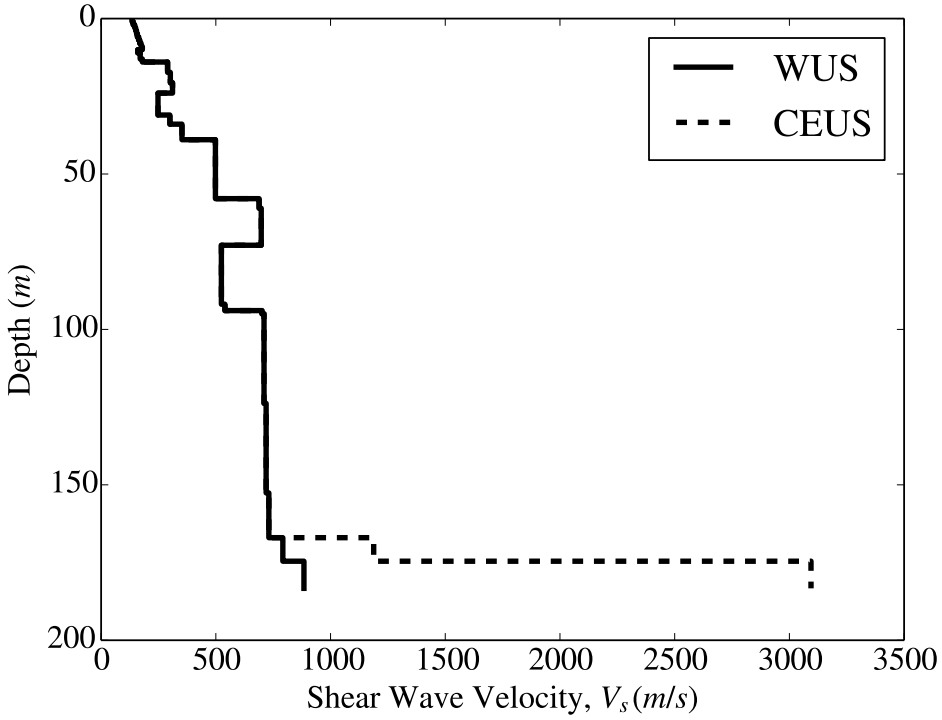


Figure E.5: Shear wave velocities of Profile004

E.6 Profile005 - MBARI NO:3 UC-B-10-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sand	1.0	0.0	20.2	133.0	133.0
sand	1.0	0.0	19.8	137.0	137.0
sand	1.0	0.0	19.3	145.0	145.0
sand	2.0	0.0	17.4	152.0	152.0
clayey silt	1.5	4.0	14.4	170.0	170.0
sand	1.0	0.0	21.4	191.0	191.0
sand	1.0	0.0	20.8	198.0	198.0
sand	2.0	0.0	20.0	206.0	206.0
sand	1.0	0.0	17.8	209.0	209.0
silty clay	1.3	5.0	16.8	160.0	160.0
silty clay	1.3	6.0	15.7	179.0	179.0
silty clay	3.3	4.0	15.5	290.0	290.0
silty clay	3.3	5.0	15.3	301.0	301.0
silty clay	3.3	6.0	14.2	310.0	310.0
clay	7.2	31.0	14.5	249.0	249.0
fine sand	2.8	0.0	20.6	301.0	301.0
silty clay	5.1	6.0	16.3	352.0	352.0
sandy gravel	18.9	0.0	22.5	498.0	498.0
sandy clay	3.0	18.0	17.4	689.0	689.0
sandy gravel	12.1	0.0	20.6	697.0	697.0
sandy gravelly clay	18.9	21.0	16.7	523.0	523.0
sandy clay	2.1	22.0	18.0	538.0	538.0
gravel	0.9	0.0	19.2	701.0	701.0
sandy clay	28.8	27.0	15.3	710.0	710.0
sandy clay	28.8	34.0	15.4	719.0	719.0
sandy clay	14.4	27.0	15.4	731.0	731.0
weathered bedrock	7.6	0.0	24.1	791.0	1186.0

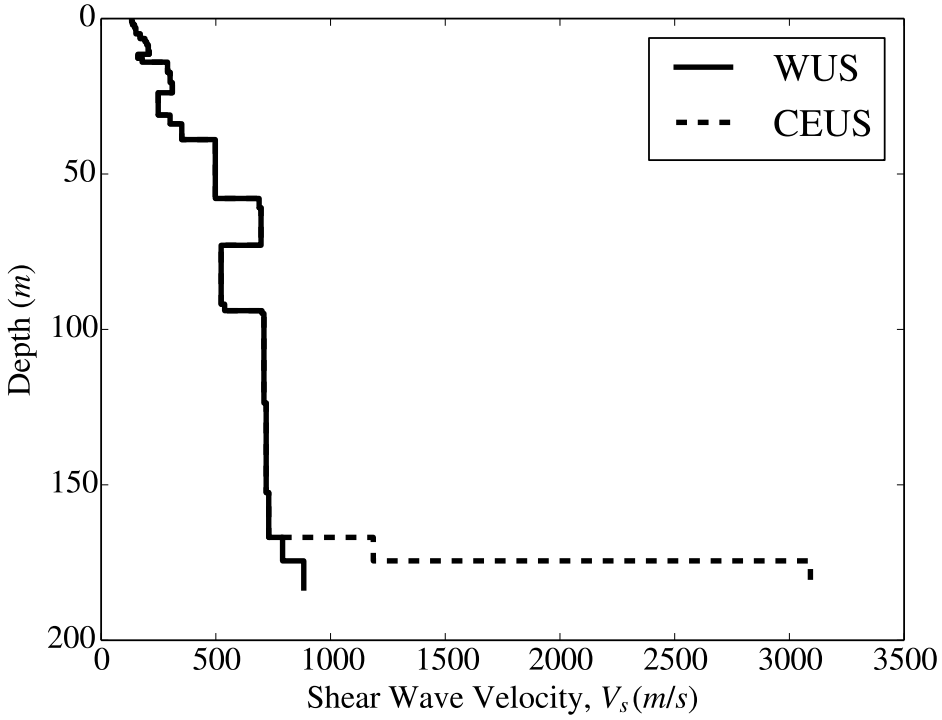


Figure E.6: Shear wave velocities of Profile005

E.7 Profile006 - Treasure Island-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty fine sand	1.1	9.0	15.8	151.0	151.0
silty fine sand	1.1	13.0	17.0	151.0	151.0
silty fine sand	1.1	20.0	18.9	150.0	150.0
silty fine sand	1.1	16.0	19.3	150.0	150.0
silty fine sand	0.9	14.0	17.0	151.0	151.0
silty fine sand	1.9	19.0	19.5	154.0	154.0
silty fine sand	0.9	4.0	17.7	160.0	160.0
silty fine sand	0.9	7.0	19.3	165.0	165.0
silty fine sand	1.9	11.0	19.1	169.0	169.0
silty fine sand	0.9	16.0	16.8	174.0	174.0
sandy clay	2.0	34.0	15.7	153.0	153.0
young bay mud	14.9	44.0	14.7	185.0	185.0
medium to dense sand	13.0	0.0	19.6	294.0	294.0
old bay mud	17.0	20.0	18.8	225.0	225.0
old bay mud	17.0	21.0	20.5	277.0	277.0
gravelly sand	4.9	0.0	19.2	449.0	449.0
old bay mud	10.0	20.0	19.5	325.0	325.0
weathered sandstone	3.7	0.0	24.0	610.0	914.0
weathered sandstone	3.7	0.0	23.0	669.0	1450.0
weathered sandstone	3.9	0.0	24.4	732.0	2073.0

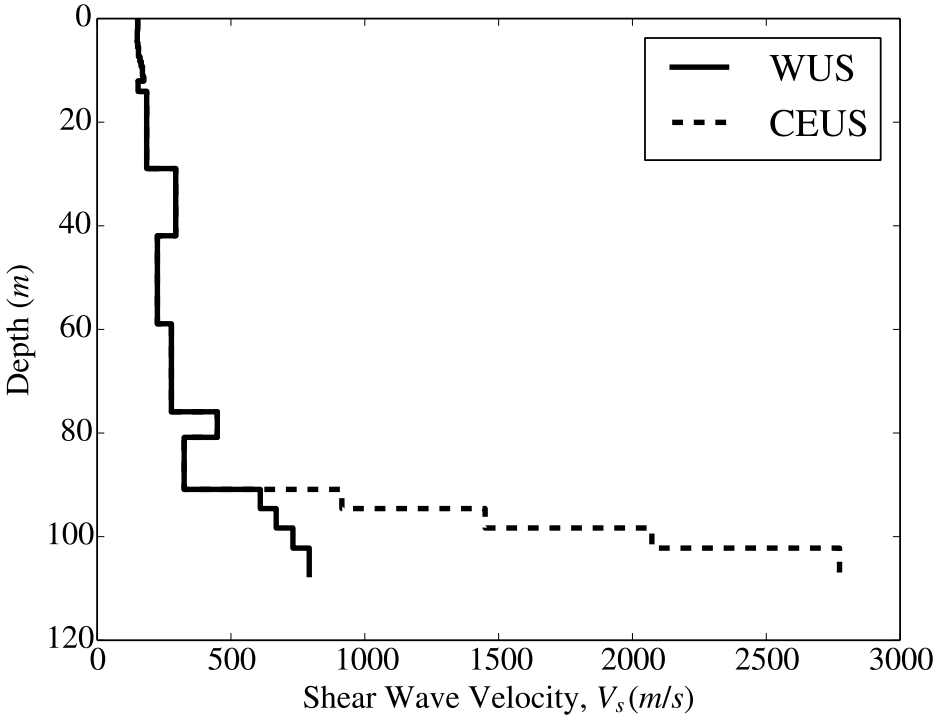


Figure E.7: Shear wave velocities of Profile006

E.8 Profile007 - SFOBB-1-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sand and gravel	1.0	0.0	21.5	109.0	109.0
sand and gravel	1.0	0.0	21.3	140.0	140.0
sand and gravel	1.0	0.0	22.7	152.0	152.0
sand and gravel	1.0	0.0	22.7	175.0	175.0
sand and gravel	1.0	0.0	22.5	124.0	124.0
sand and gravel	1.0	0.0	22.7	175.0	175.0
silt	1.1	17.0	16.8	149.0	149.0
sand with silt	1.0	0.0	19.5	152.0	152.0
sand with silt	1.9	0.0	20.9	167.0	167.0
sand with silt	1.0	0.0	18.7	182.0	182.0
sand with silt	1.0	0.0	18.9	201.0	201.0
old bay clay	13.2	21.0	19.7	247.0	247.0
old bay clay	13.2	20.0	19.2	258.0	258.0
old bay clay	13.2	18.0	19.1	270.0	270.0
old bay clay	13.2	20.0	20.0	285.0	285.0
old bay clay	13.2	20.0	19.4	296.0	296.0
old bay clay	13.2	17.0	19.7	312.0	312.0
alluvium	12.2	28.0	17.0	418.0	418.0
alluvium	12.2	14.0	16.0	487.0	487.0
alluvium	12.2	14.0	16.1	556.0	556.0
alluvium	12.2	14.0	17.0	609.0	609.0
weathered bedrock	3.0	0.0	24.2	700.0	1051.0
weathered bedrock	3.1	0.0	25.0	745.0	1416.0
weathered bedrock	3.0	0.0	23.9	792.0	1821.0
weathered bedrock	3.0	0.0	24.2	837.0	2261.0
weathered bedrock	3.0	0.0	24.3	867.0	2687.0

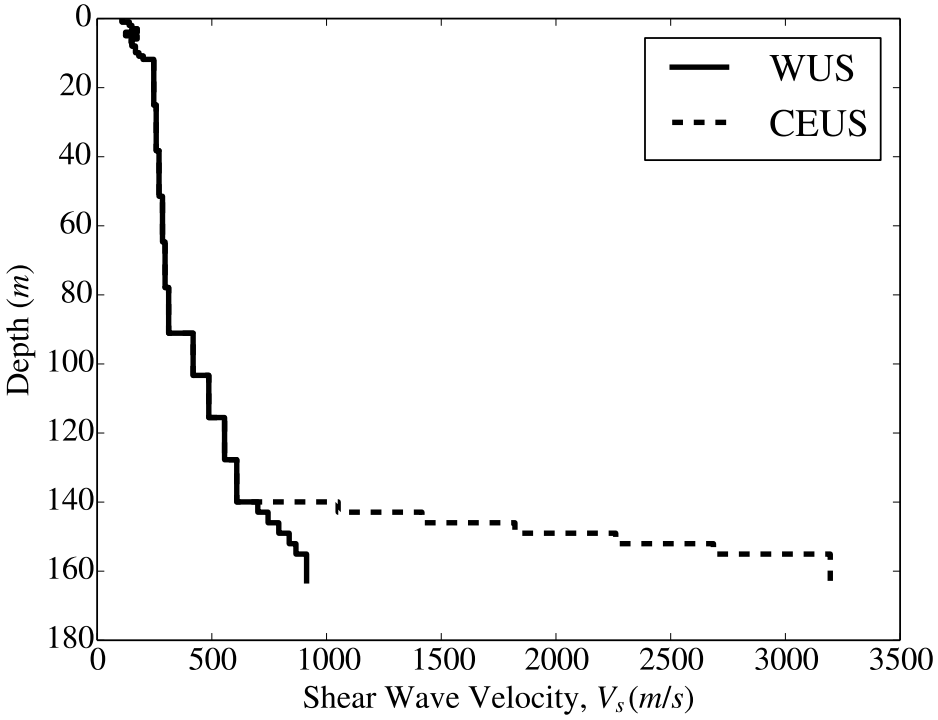


Figure E.8: Shear wave velocities of Profile007

E.9 Profile008 - SFOBB-2-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sand and gravel	1.0	0.0	20.8	152.0	152.0
sand and gravel	1.0	0.0	18.9	164.0	164.0
sand and gravel	1.0	0.0	22.6	169.0	169.0
sand and gravel	1.0	0.0	19.4	179.0	179.0
sand and gravel	1.0	0.0	23.6	140.0	140.0
sand and gravel	1.0	0.0	19.9	160.0	160.0
silt	0.8	19.0	14.2	140.0	140.0
sand with silt	0.9	0.0	17.9	146.0	146.0
sand with silt	0.9	0.0	20.8	153.0	153.0
sand with silt	0.9	0.0	18.5	160.0	160.0
old bay clay	13.2	21.0	19.7	247.0	247.0
old bay clay	13.2	21.0	19.5	258.0	258.0
old bay clay	13.2	20.0	19.3	270.0	270.0
old bay clay	13.2	18.0	19.0	285.0	285.0
old bay clay	13.2	21.0	19.0	296.0	296.0
old bay clay	13.2	20.0	18.7	312.0	312.0
alluvium	16.2	12.0	16.2	418.0	418.0
alluvium	16.2	34.0	16.8	510.0	510.0
alluvium	16.2	23.0	15.8	594.0	594.0
weathered bedrock	3.1	0.0	22.9	701.0	1052.0
weathered bedrock	3.1	0.0	23.5	746.0	1417.0
weathered bedrock	3.0	0.0	24.9	792.0	1821.0
weathered bedrock	3.0	0.0	25.0	838.0	2262.0
weathered bedrock	3.1	0.0	24.6	868.0	2691.0

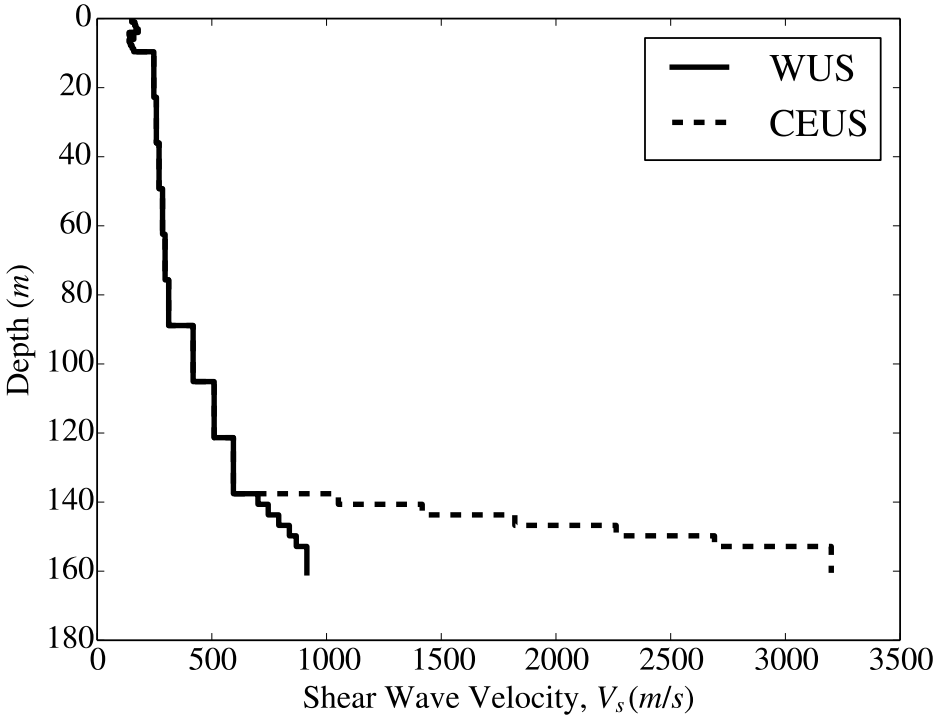


Figure E.9: Shear wave velocities of Profile008

E.10 Profile009 - POO7-2-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sand	1.0	0.0	21.1	126.0	126.0
sand	1.0	0.0	18.8	137.0	137.0
sand	1.0	0.0	19.9	149.0	149.0
sand	1.0	0.0	20.9	161.0	161.0
sand	1.0	0.0	17.7	176.0	176.0
sand	3.0	0.0	21.2	181.0	181.0
sand	1.0	0.0	19.0	196.0	196.0
sand	1.0	0.0	18.2	200.0	200.0
old bay clay	12.9	21.0	19.7	243.0	243.0
old bay clay	12.9	19.0	20.0	264.0	264.0
old bay clay	12.9	20.0	19.1	282.0	282.0
old bay clay	12.9	19.0	19.9	302.0	302.0
old bay clay	12.9	19.0	19.8	315.0	315.0
alluvium	15.7	14.0	17.0	395.0	395.0
alluvium	15.7	15.0	17.9	580.0	580.0
alluvium	15.7	24.0	18.3	707.0	707.0
weathered bedrock	4.5	0.0	23.0	792.0	1188.0
weathered bedrock	1.5	0.0	23.4	871.0	1886.0
weathered bedrock	1.5	0.0	23.0	832.0	2356.0

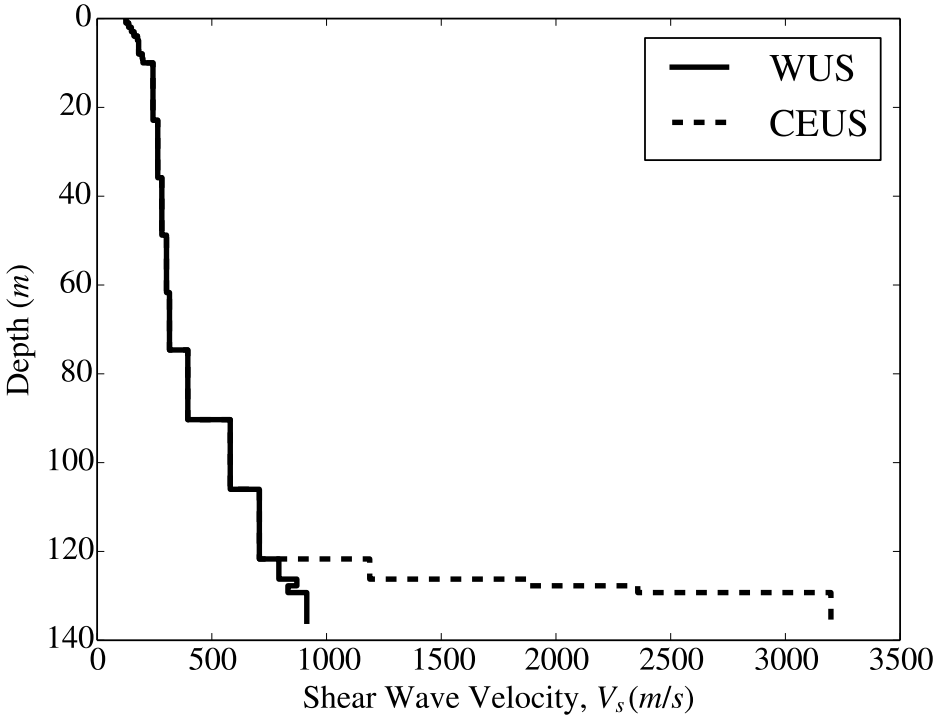


Figure E.10: Shear wave velocities of Profile009

E.11 Profile010 - POO7-3-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sand	1.0	0.0	17.6	141.0	141.0
sand	2.1	0.0	21.3	152.0	152.0
sand	1.0	0.0	20.9	164.0	164.0
sand	1.0	0.0	19.3	174.0	174.0
sand	1.0	0.0	22.0	183.0	183.0
sand	1.0	0.0	19.8	107.0	107.0
sand	1.0	0.0	21.2	125.0	125.0
sand	1.0	0.0	21.8	137.0	137.0
sand	1.0	0.0	18.4	152.0	152.0
old bay clay	13.5	19.0	19.7	250.0	250.0
old bay clay	13.5	19.0	19.4	268.0	268.0
old bay clay	13.5	20.0	19.9	286.0	286.0
old bay clay	13.5	20.0	20.2	311.0	311.0
old bay clay	13.5	22.0	19.7	335.0	335.0
old bay clay	13.5	19.0	19.8	350.0	350.0
alluvium	14.7	17.0	17.6	381.0	381.0
alluvium	14.7	26.0	17.4	556.0	556.0
alluvium	14.7	9.0	18.6	670.0	670.0
alluvium	14.7	25.0	18.8	747.0	747.0
weathered bedrock	3.0	0.0	22.6	701.0	1051.0
weathered bedrock	3.0	0.0	23.7	746.0	1418.0
weathered bedrock	3.1	0.0	24.6	791.0	1820.0
weathered bedrock	3.0	0.0	25.1	838.0	2262.0
weathered bedrock	3.0	0.0	25.1	868.0	2692.0

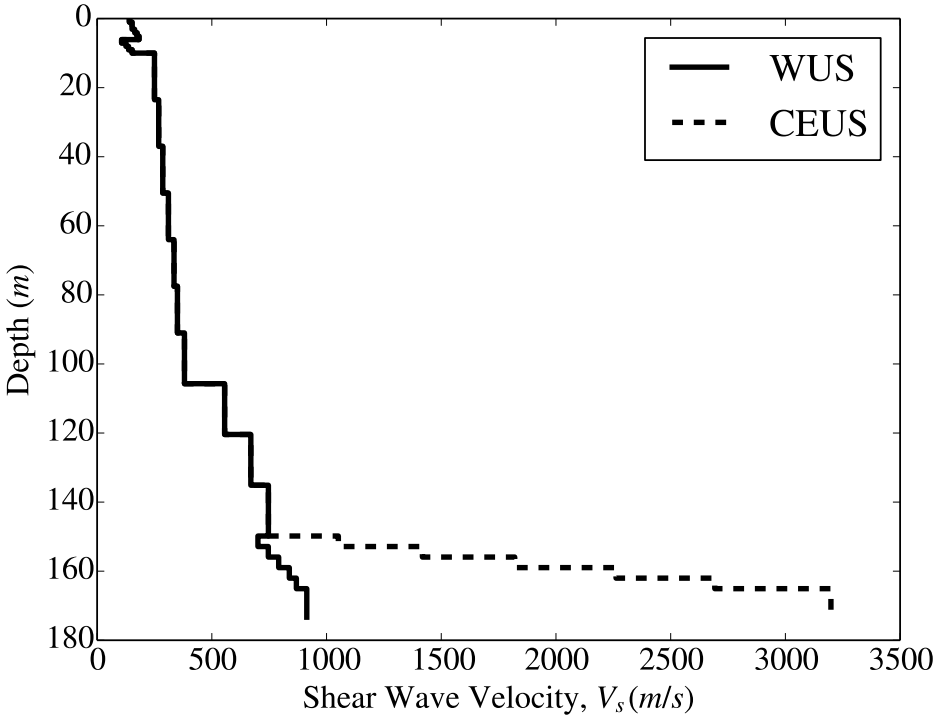


Figure E.11: Shear wave velocities of Profile010

E.12 Profile011 - Woodward Marine UC-B4-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sand	1.1	0.0	19.2	120.0	120.0
sand	1.1	0.0	20.7	130.0	130.0
sand	1.1	0.0	17.4	169.0	169.0
sand	1.1	0.0	17.8	219.0	219.0
silty sand with clay	1.2	28.0	17.6	159.0	159.0
silty sand with clay	1.2	7.0	17.6	168.0	168.0
silty sand with clay	1.2	6.0	17.3	181.0	181.0
sand	1.0	0.0	19.8	250.0	250.0
clay	0.9	27.0	16.7	171.0	171.0
silty clay	3.4	7.0	15.6	290.0	290.0
silty clay	3.4	5.0	17.2	302.0	302.0
silty clay	3.4	5.0	17.2	311.0	311.0
silty clay	3.4	4.0	15.7	238.0	238.0
silty clay	3.4	7.0	15.0	258.0	258.0
sand	3.0	0.0	18.6	300.0	300.0
silty clay	4.9	7.0	14.7	352.0	352.0
sandy gravel	19.0	0.0	22.9	498.0	498.0
sandy clay	2.9	15.0	15.7	688.0	688.0
sandy gravel	12.0	0.0	21.7	698.0	698.0
sandy gravelly clay	19.0	8.0	16.3	524.0	524.0
sandy clay	2.0	20.0	16.5	539.0	539.0
gravel	1.0	0.0	23.5	701.0	701.0
sandy gravelly clay	28.7	25.0	18.1	710.0	710.0
sandy gravelly clay	28.7	10.0	18.7	719.0	719.0
sandy gravelly clay	14.4	9.0	17.6	732.0	732.0
weathered bedrock	7.8	0.0	25.0	792.0	1189.0

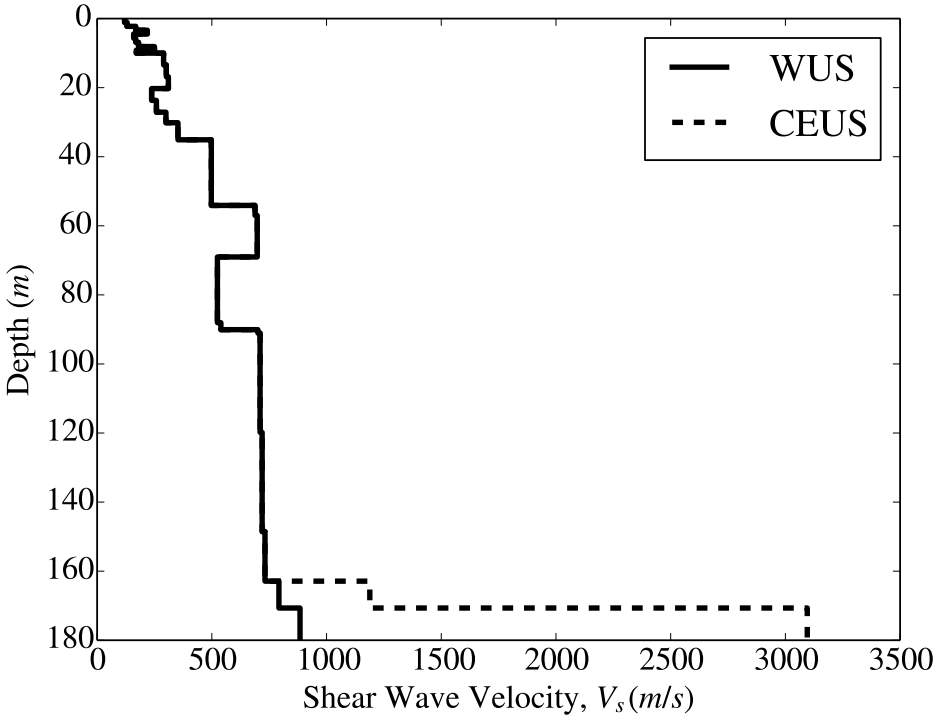


Figure E.12: Shear wave velocities of Profile011

E.13 Profile012 - Moss Landing UC-B1-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sand	1.0	0.0	21.2	120.0	120.0
sand	1.0	0.0	18.1	125.0	125.0
sand	1.0	0.0	20.8	129.0	129.0
sand	1.0	0.0	20.8	137.0	137.0
sand	1.0	0.0	18.4	141.0	141.0
sand and gravel	1.2	0.0	23.5	208.0	208.0
sand and gravel	1.2	0.0	21.6	218.0	218.0
sand and gravel	1.2	0.0	22.3	229.0	229.0
silty sand with clay	1.2	28.0	18.4	162.0	162.0
silty sand with clay	1.2	30.0	17.2	173.0	173.0
silty sand with clay	1.2	30.0	16.8	181.0	181.0
sand	1.0	0.0	21.1	250.0	250.0
silty clay	9.0	5.0	16.1	266.0	266.0
silty clay	9.0	5.0	14.3	270.0	270.0
sand	1.5	0.0	19.4	281.0	281.0
silty clay	6.5	5.0	14.8	345.0	345.0
sandy gravel	19.0	0.0	21.8	499.0	499.0
sandy clay	3.0	13.0	15.7	689.0	689.0
sandy gravel	11.9	0.0	21.2	699.0	699.0
sandy gravelly clay	19.0	27.0	18.6	525.0	525.0
sandy clay	2.0	12.0	15.8	540.0	540.0
gravel	1.0	0.0	19.8	702.0	702.0
sandy clay	28.8	19.0	17.3	711.0	711.0
sandy clay	28.8	12.0	16.5	720.0	720.0
sandy clay	14.4	19.0	18.0	732.0	732.0
weathered bedrock	7.7	0.0	24.5	793.0	1190.0

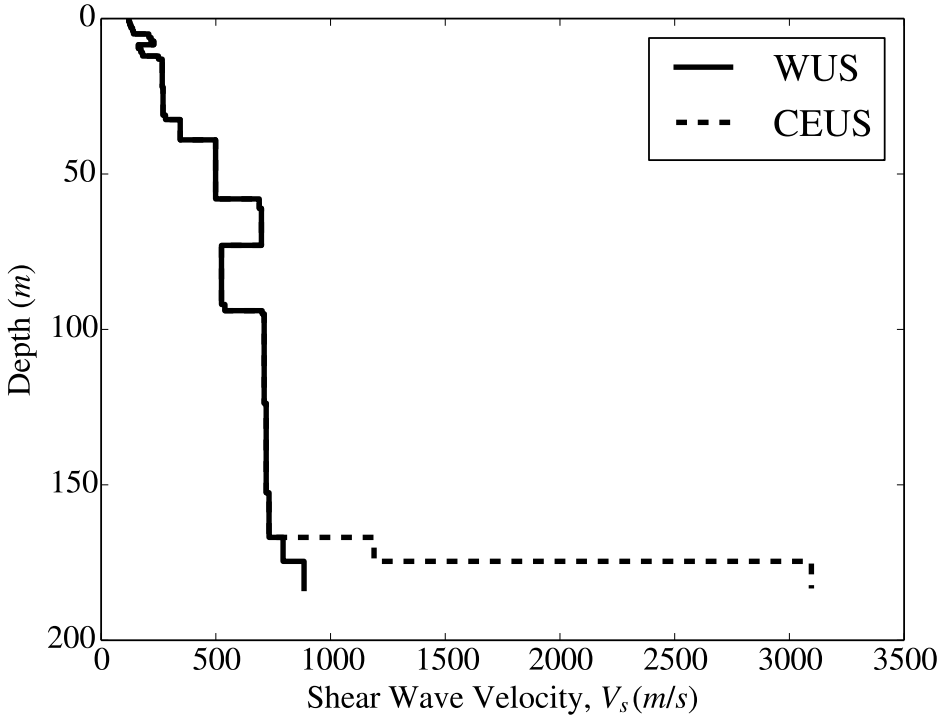


Figure E.13: Shear wave velocities of Profile012

E.14 Profile013 - Moss Landing UC-B2-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sand	1.0	0.0	19.9	153.0	153.0
sand	1.0	0.0	21.7	159.0	159.0
sand	1.0	0.0	20.4	167.0	167.0
sand	1.0	0.0	17.6	171.0	171.0
sand	1.0	0.0	17.9	176.0	176.0
sand and gravel	1.0	0.0	21.9	180.0	180.0
sand and gravel	1.0	0.0	22.3	183.0	183.0
sand and gravel	1.0	0.0	21.6	208.0	208.0
sand and gravel	1.0	0.0	23.4	219.0	219.0
sand and gravel	1.0	0.0	20.9	229.0	229.0
silty sand with clay	1.0	9.0	18.5	160.0	160.0
silty sand with clay	1.0	27.0	16.1	180.0	180.0
sand	1.0	0.0	17.7	250.0	250.0
silty clay	9.0	6.0	15.5	266.0	266.0
silty clay	9.0	6.0	15.6	269.0	269.0
clay	3.0	31.0	14.7	300.0	300.0
fine sand	4.9	0.0	20.9	353.0	353.0
silty clay	19.0	6.0	17.0	498.0	498.0
sandy gravel	3.1	0.0	21.9	688.0	688.0
sandy clay	12.0	19.0	17.5	698.0	698.0
sandy gravel	21.0	0.0	20.0	524.0	524.0
sandy clay	1.0	9.0	16.2	701.0	701.0
gravel	28.8	0.0	22.9	710.0	710.0
gravel	28.8	0.0	21.0	719.0	719.0
gravel	14.4	0.0	21.7	732.0	732.0
weathered bedrock	7.6	0.0	23.2	792.0	1188.0

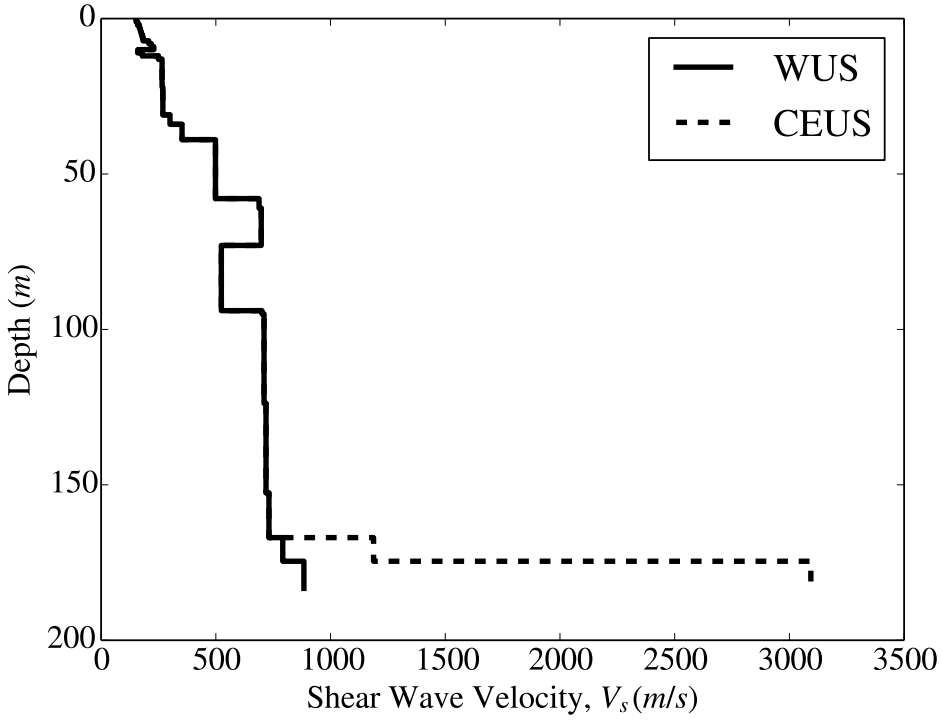


Figure E.14: Shear wave velocities of Profile013

E.15 Profile014 - Marine Lab. B1-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sand	1.0	0.0	18.5	137.0	137.0
sand	1.0	0.0	21.1	156.0	156.0
sand	1.0	0.0	19.3	166.0	166.0
sand	1.0	0.0	21.8	172.0	172.0
sand	1.0	0.0	21.8	178.0	178.0
sand	1.1	0.0	21.1	180.0	180.0
sand	1.1	0.0	19.6	189.0	189.0
sand	1.1	0.0	20.0	210.0	210.0
silty sand with clay	1.1	27.0	17.4	160.0	160.0
silty sand with clay	1.1	8.0	17.8	166.0	166.0
silty sand with clay	1.1	19.0	17.7	169.0	169.0
silty sand with clay	1.1	18.0	18.3	181.0	181.0
sand	1.0	0.0	21.8	250.0	250.0
silty clay	1.0	5.0	15.7	171.0	171.0
silty clay	15.1	5.0	17.1	284.0	284.0
clay	2.9	12.0	14.5	301.0	301.0
fine sand	5.1	0.0	19.4	353.0	353.0
silty clay	18.9	7.0	16.2	499.0	499.0
sandy gravel	3.0	0.0	20.7	689.0	689.0
sandy clay	12.0	24.0	15.7	698.0	698.0
sandy gravel	19.0	0.0	22.5	524.0	524.0
sandy gravelly clay	2.0	10.0	16.6	539.0	539.0
sandy clay	1.0	23.0	15.7	701.0	701.0
gravel	28.8	0.0	20.3	710.0	710.0
gravel	28.8	0.0	23.4	720.0	720.0
gravel	14.4	0.0	22.6	732.0	732.0
weathered bedrock	7.6	0.0	23.7	792.0	1189.0

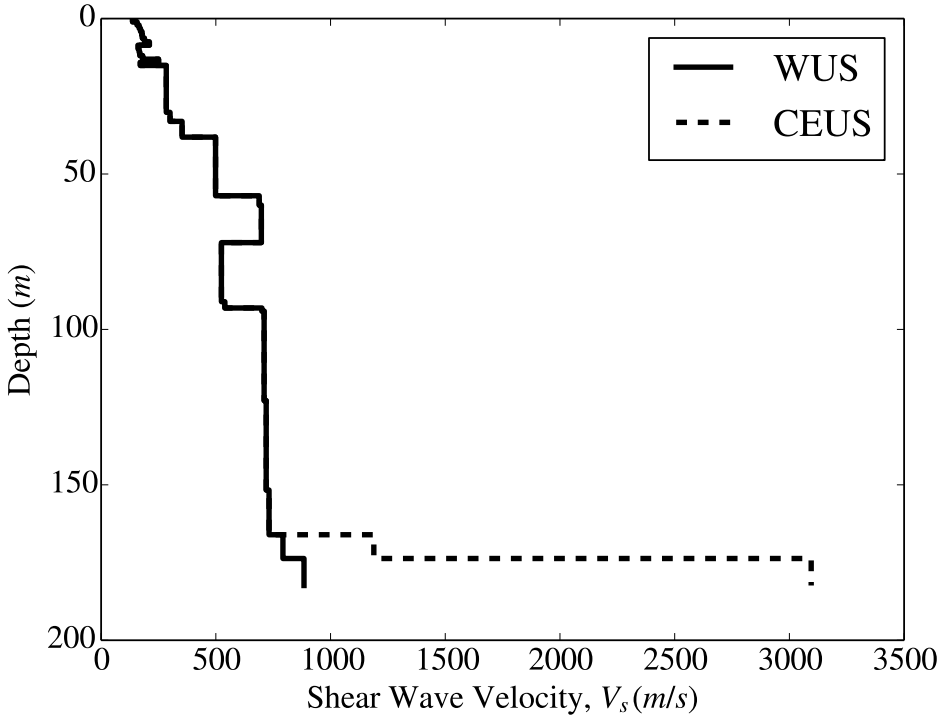


Figure E.15: Shear wave velocities of Profile014

E.16 Profile015 - Marine Lab. B2-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sand	1.1	0.0	21.2	115.0	115.0
sand	1.1	0.0	20.2	140.0	140.0
sand	1.1	0.0	20.5	143.0	143.0
sand	1.1	0.0	20.8	146.0	146.0
sand	1.1	0.0	18.5	149.0	149.0
sand	1.1	0.0	19.4	153.0	153.0
sand	1.1	0.0	19.3	219.0	219.0
sand	1.1	0.0	20.0	237.0	237.0
silty sand with clay	0.9	8.0	18.9	159.0	159.0
silty sand with clay	0.9	7.0	18.1	165.0	165.0
silty sand with clay	0.9	23.0	15.7	164.0	164.0
silty sand with clay	0.9	21.0	16.5	172.0	172.0
silty sand with clay	0.9	25.0	16.9	180.0	180.0
sand	0.8	0.0	18.7	250.0	250.0
silty clay	1.1	4.0	15.0	170.0	170.0
silty clay	4.0	5.0	16.5	294.0	294.0
silty clay	4.0	5.0	15.3	308.0	308.0
clay	6.9	29.0	15.2	248.0	248.0
fine sand	3.0	0.0	17.9	300.0	300.0
silty clay	5.1	7.0	15.7	352.0	352.0
sandy gravel	18.9	0.0	21.1	498.0	498.0
sandy clay	3.1	14.0	15.2	689.0	689.0
sandy gravel	12.0	0.0	22.5	698.0	698.0
sandy gravelly clay	19.0	21.0	17.8	524.0	524.0
sandy clay	2.0	9.0	16.6	540.0	540.0
gravel	1.0	0.0	18.9	701.0	701.0
sandy clay	28.8	31.0	16.7	710.0	710.0
sandy clay	28.8	23.0	15.1	720.0	720.0
sandy clay	14.4	35.0	15.8	731.0	731.0
weathered bedrock	7.6	0.0	23.0	793.0	1189.0

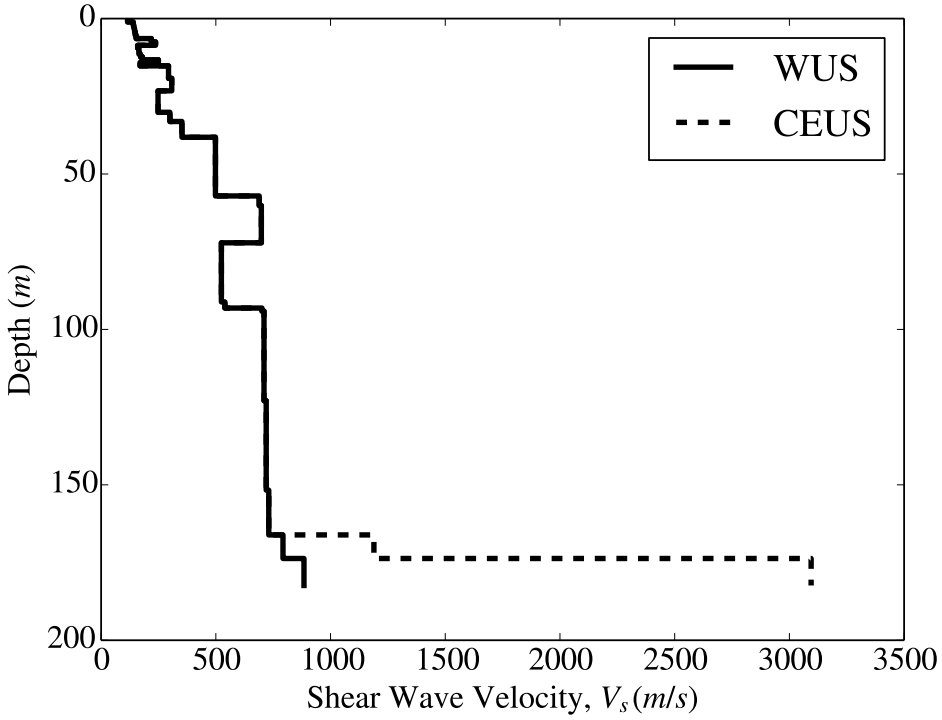


Figure E.16: Shear wave velocities of Profile015

E.17 Profile016 - Miller Farm CMF 3-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty sand	1.0	19.0	19.8	122.0	122.0
silty sand	1.0	5.0	17.9	131.0	131.0
silty sand	1.0	14.0	16.8	136.0	136.0
silty sand	1.0	9.0	19.7	141.0	141.0
silty sand	1.0	8.0	18.2	180.0	180.0
silty sand	1.0	13.0	20.0	189.0	189.0
silty sand	2.0	9.0	18.1	201.0	201.0
silty sand	1.0	6.0	17.9	211.0	211.0
sandy silt	1.0	11.0	17.2	198.0	198.0
sandy silt	1.0	17.0	17.5	204.0	204.0
sandy silt	2.0	12.0	17.1	210.0	210.0
sandy silt	1.0	17.0	17.1	216.0	216.0
sandy silt	1.0	12.0	16.3	226.0	226.0
sand	5.0	0.0	19.8	300.0	300.0
silt	1.9	13.0	14.5	259.0	259.0
clay	9.1	23.0	15.6	248.0	248.0
fine sand	2.9	0.0	20.3	300.0	300.0
silty clay	5.0	4.0	17.1	353.0	353.0
sandy gravel	19.0	0.0	21.0	498.0	498.0
sandy clay	3.0	9.0	16.3	689.0	689.0
sandy gravel	11.9	0.0	19.8	698.0	698.0
sandy gravelly clay	19.0	19.0	16.8	524.0	524.0
sandy clay	2.1	34.0	17.6	540.0	540.0
gravel	1.0	0.0	20.1	701.0	701.0
sandy clay	28.8	18.0	18.0	711.0	711.0
sandy clay	28.8	20.0	15.2	719.0	719.0
sandy clay	14.4	22.0	16.7	732.0	732.0
weather bedrock	7.6	0.0	22.3	792.0	1188.0

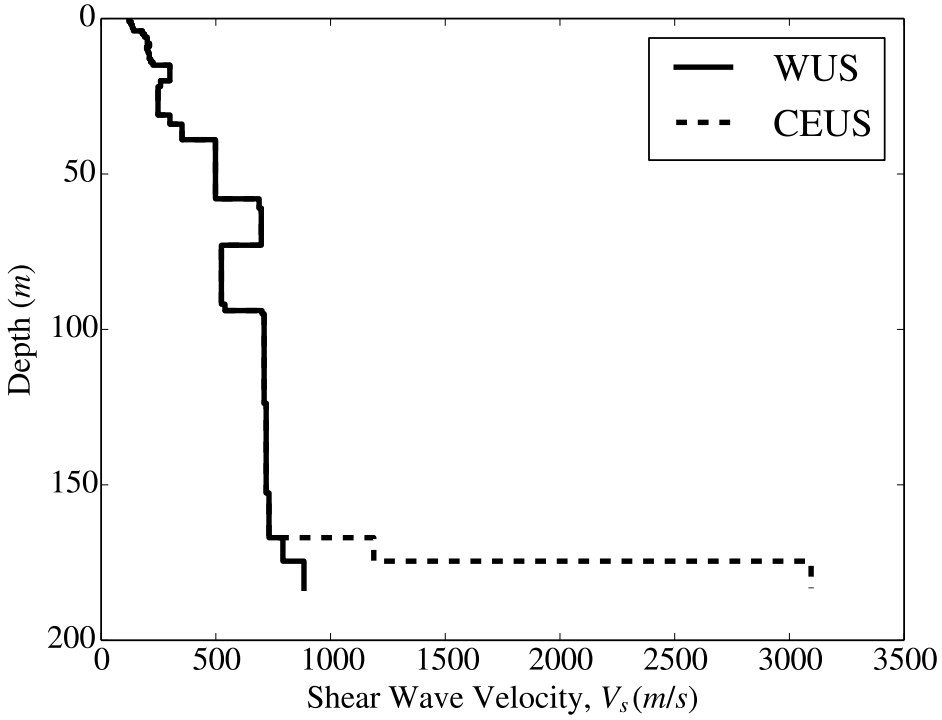


Figure E.17: Shear wave velocities of Profile016

E.18 Profile017 - Miller Farm CMF 5-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty sand	0.9	16.0	18.4	129.0	129.0
silty sand	0.9	20.0	20.2	139.0	139.0
silty sand	0.9	8.0	17.2	149.0	149.0
silty sand	0.9	18.0	17.5	161.0	161.0
silty sand	0.9	15.0	18.8	180.0	180.0
silty sand	0.9	19.0	20.3	202.0	202.0
silty sand	1.8	16.0	16.0	206.0	206.0
silty sand	0.9	15.0	16.7	213.0	213.0
silty sand	0.9	14.0	16.0	219.0	219.0
silty sand	1.8	7.0	18.1	226.0	226.0
silty sand	0.9	15.0	16.5	229.0	229.0
lean clay	1.0	19.0	17.2	189.0	189.0
sand	3.5	0.0	20.2	296.0	296.0
sand	3.5	0.0	19.6	309.0	309.0
silt	2.0	10.0	14.3	259.0	259.0
clay	9.0	7.0	17.0	248.0	248.0
fine sand	3.0	0.0	19.0	300.0	300.0
silty clay	4.9	5.0	16.5	353.0	353.0
sandy gravel	19.1	0.0	20.8	498.0	498.0
sandy clay	2.9	26.0	17.7	689.0	689.0
sandy gravel	12.0	0.0	20.9	698.0	698.0
sandy gravelly clay	19.0	16.0	16.2	524.0	524.0
sandy clay	2.0	29.0	16.6	539.0	539.0
gravel	1.0	0.0	22.8	701.0	701.0
sandy clay	28.8	13.0	16.2	710.0	710.0
sandy clay	28.8	12.0	16.3	719.0	719.0
sandy clay	14.4	9.0	16.5	731.0	731.0
weathered bedrock	7.6	0.0	22.7	793.0	1189.0

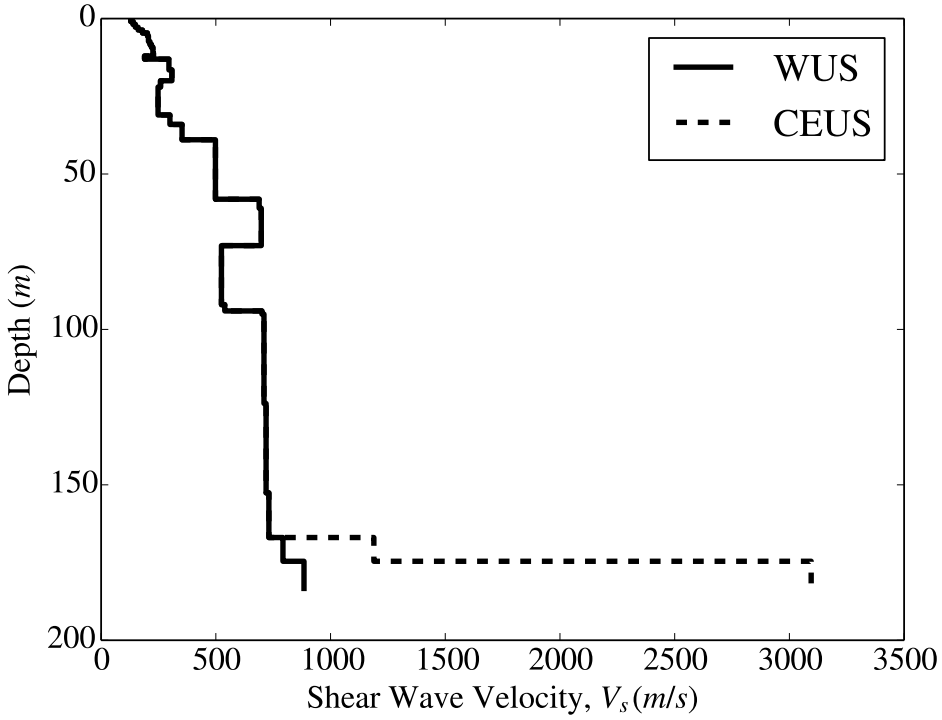


Figure E.18: Shear wave velocities of Profile017

E.19 Profile018 - Miller Farm CMF 8-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty sand	1.0	8.0	19.7	113.0	113.0
silty sand	1.0	17.0	20.1	123.0	123.0
silty sand	1.0	8.0	16.6	129.0	129.0
silty sand	1.0	15.0	18.0	135.0	135.0
silty sand	1.0	8.0	16.6	141.0	141.0
silty sand	1.0	15.0	20.1	180.0	180.0
silty sand	2.0	12.0	17.4	190.0	190.0
silty sand	1.0	7.0	15.9	198.0	198.0
silty sand	1.0	7.0	18.5	207.0	207.0
silty sand	1.0	11.0	17.1	217.0	217.0
silty sand	2.0	20.0	17.0	226.0	226.0
silty sand	1.0	6.0	19.9	241.0	241.0
lean clay	1.0	32.0	14.7	190.0	190.0
sand	4.9	0.0	21.8	300.0	300.0
silt	2.1	11.0	16.9	259.0	259.0
clay	9.0	32.0	14.2	248.0	248.0
fine sand	3.0	0.0	19.7	300.0	300.0
silty clay	4.9	6.0	15.4	353.0	353.0
sandy gravel	19.1	0.0	20.0	498.0	498.0
sandy clay	2.9	26.0	17.0	689.0	689.0
sandy gravel	12.0	0.0	23.3	698.0	698.0
sandy gravelly clay	19.0	26.0	16.0	524.0	524.0
sandy clay	2.1	23.0	17.3	539.0	539.0
gravel	1.0	0.0	19.6	701.0	701.0
sandy clay	28.8	31.0	15.2	710.0	710.0
sandy clay	28.8	29.0	16.7	719.0	719.0
sandy clay	14.4	12.0	17.5	732.0	732.0
weathered bedrock	7.6	0.0	23.1	793.0	1189.0

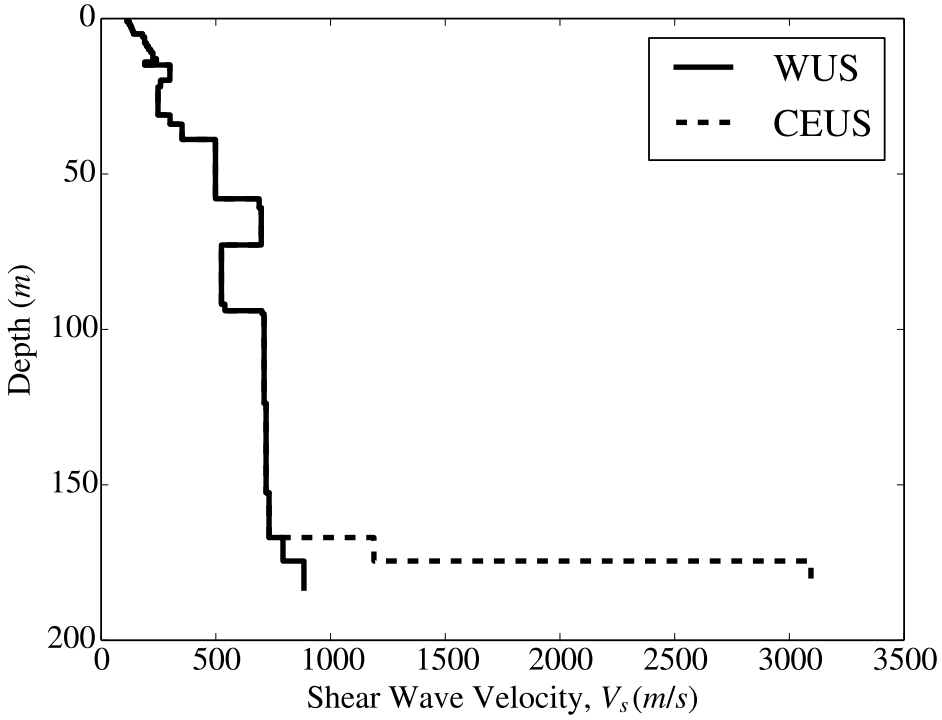


Figure E.19: Shear wave velocities of Profile018

E.20 Profile019 - Miller Farm CMF 10-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty sandy clay	1.0	15.0	17.3	137.0	137.0
silty sandy clay	1.0	8.0	15.5	143.0	143.0
silty sandy clay	1.0	10.0	17.8	153.0	153.0
silty sandy clay	1.0	17.0	17.7	162.0	162.0
silty sandy clay	1.0	20.0	17.2	168.0	168.0
silty sandy clay	1.0	15.0	15.2	174.0	174.0
silty sandy clay	1.0	21.0	16.6	180.0	180.0
silty sand	1.0	8.0	19.3	200.0	200.0
silty sand	1.0	10.0	20.0	213.0	213.0
silty sand	1.0	8.0	16.9	225.0	225.0
silty sand	1.0	6.0	18.5	240.0	240.0
lean clay	1.0	10.0	15.8	190.0	190.0
sand	4.0	0.0	19.0	295.0	295.0
sand	4.0	0.0	21.3	310.0	310.0
silt	2.0	5.0	14.6	259.0	259.0
clay	9.0	19.0	16.2	247.0	247.0
fine sand	3.0	0.0	20.6	300.0	300.0
silty clay	4.9	6.0	15.3	353.0	353.0
sandy gravel	19.0	0.0	22.4	498.0	498.0
sandy clay	3.0	14.0	16.5	688.0	688.0
sandy gravel	12.0	0.0	22.2	698.0	698.0
sandy gravelly clay	19.0	10.0	17.3	524.0	524.0
sandy clay	2.0	25.0	16.6	539.0	539.0
gravel	1.0	0.0	21.7	701.0	701.0
sandy clay	28.8	34.0	16.6	710.0	710.0
sandy clay	28.8	29.0	16.5	719.0	719.0
sandy clay	14.4	25.0	17.9	731.0	731.0
weathered bedrock	7.5	0.0	22.8	792.0	1188.0

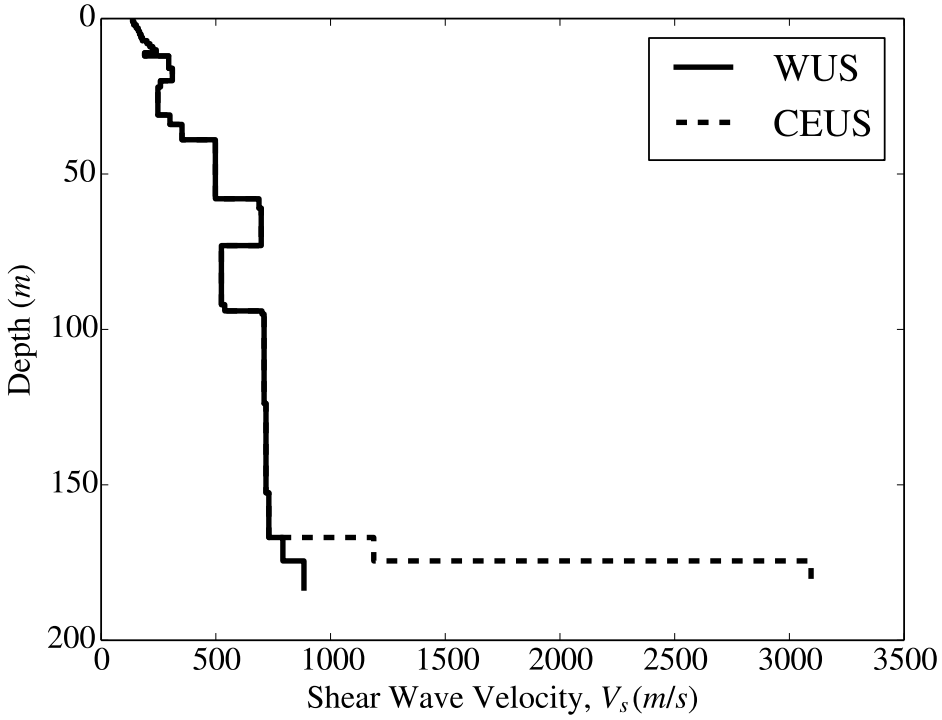


Figure E.20: Shear wave velocities of Profile019

E.21 Profile020 - Richmond City Hall-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sandy clay	1.5	31.0	16.5	79.0	79.0
silty clay	1.0	4.0	14.4	84.0	84.0
silty clay	1.0	6.0	14.6	100.0	100.0
silty sand	1.3	9.0	17.5	107.0	107.0
silty sand	1.3	12.0	19.9	125.0	125.0
silty sand	1.3	5.0	19.4	157.0	157.0
silty sand	1.3	17.0	16.2	175.0	175.0
old bay clay	3.8	19.0	19.3	249.0	249.0
old bay clay	3.8	20.0	19.9	327.0	327.0
weathered bedrock	3.0	0.0	22.6	731.0	1097.0
weathered bedrock	3.1	0.0	22.8	792.0	1717.0
weathered bedrock	3.1	0.0	22.5	853.0	2418.0

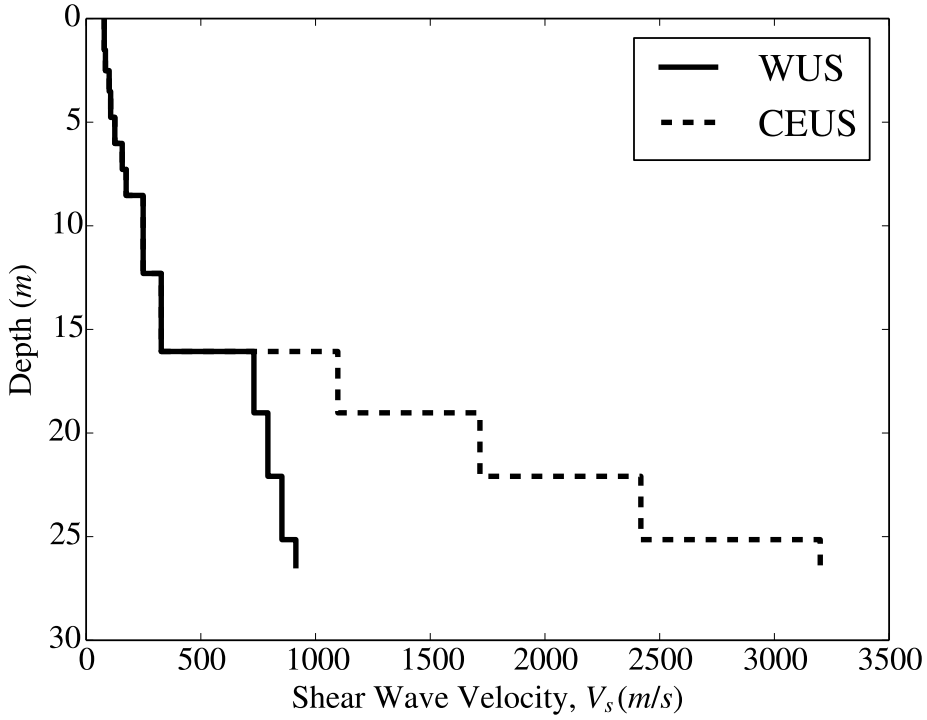


Figure E.21: Shear wave velocities of Profile020

E.22 Profile021 - Port of Richmond POR-2-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sandy clay	1.0	25.0	16.0	84.0	84.0
sandy clay	1.0	29.0	17.0	100.0	100.0
sandy clay	1.0	32.0	15.2	112.0	112.0
sandy clay	1.0	25.0	15.2	121.0	121.0
silt	1.0	14.0	16.2	121.0	121.0
silt	1.0	9.0	16.7	131.0	131.0
silt	1.0	4.0	14.3	144.0	144.0
silt	1.0	18.0	14.8	152.0	152.0
old bay clay	4.0	21.0	19.4	200.0	200.0
old bay clay	4.0	19.0	20.8	295.0	295.0
weathered bedrock	3.0	0.0	24.2	732.0	1098.0
weathered bedrock	3.0	0.0	24.5	793.0	1718.0
weathered bedrock	3.0	0.0	24.1	853.0	2418.0

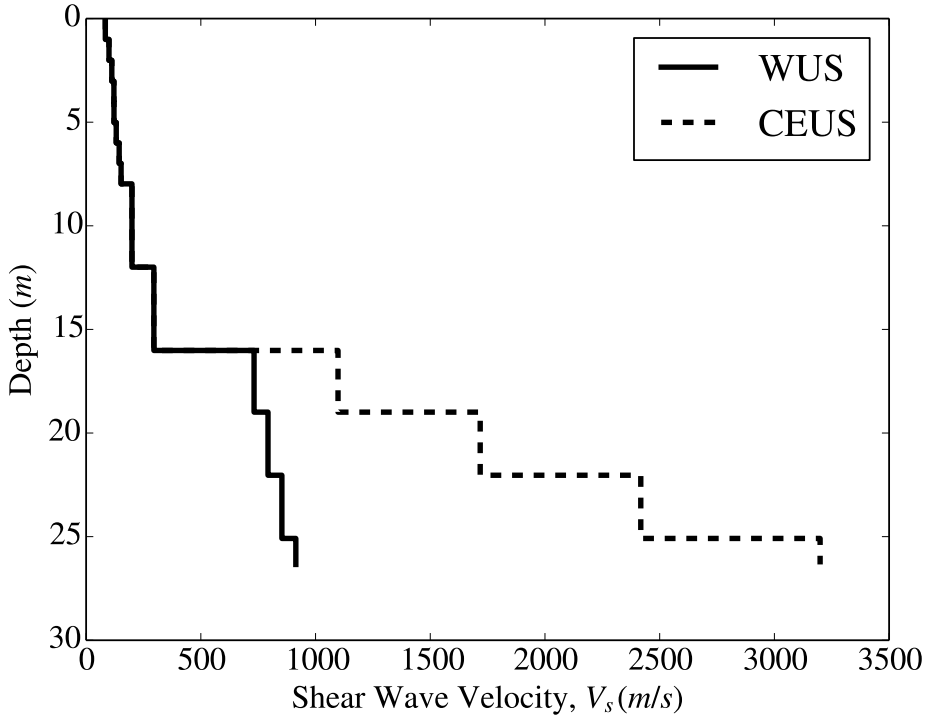


Figure E.22: Shear wave velocities of Profile021

E.23 Profile022 - Port of Richmond POR-3-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sandy clay	1.0	17.0	15.3	84.0	84.0
sandy clay	1.0	27.0	15.2	96.0	96.0
sandy clay	1.0	12.0	15.9	107.0	107.0
sandy clay	1.0	31.0	15.9	114.0	114.0
sandy clay	1.0	22.0	15.9	121.0	121.0
silty sand	1.0	11.0	16.4	122.0	122.0
silty sand	1.0	16.0	19.7	152.0	152.0
old bay clay	4.5	20.0	19.6	202.0	202.0
old bay clay	4.5	19.0	19.6	322.0	322.0
weathered bedrock	3.0	0.0	23.0	731.0	1097.0
weathered bedrock	3.0	0.0	23.1	792.0	1715.0
weathered bedrock	3.0	0.0	23.7	852.0	2415.0

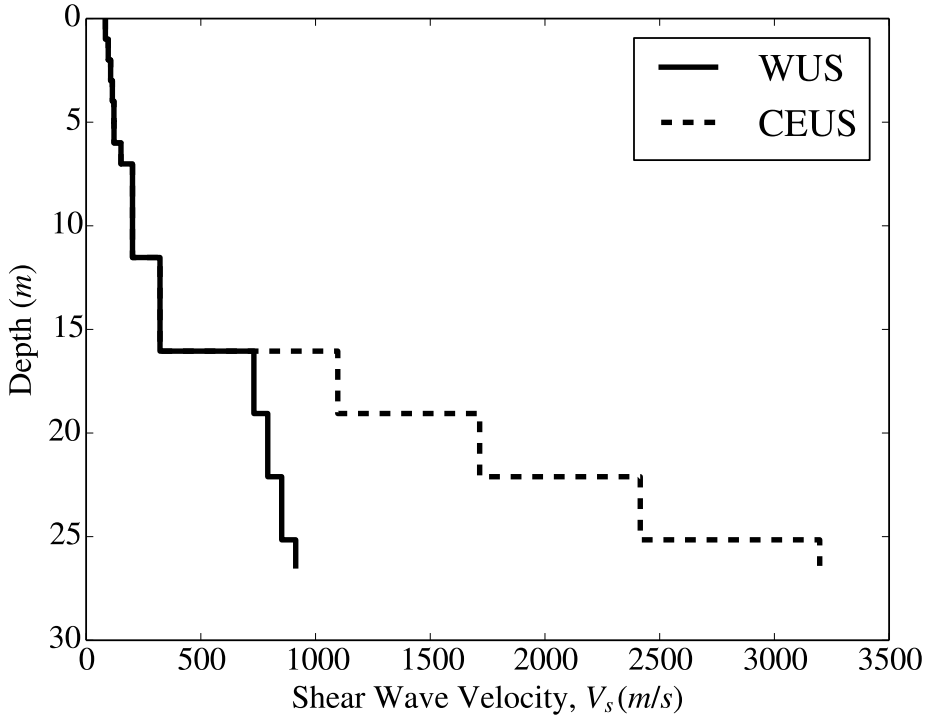


Figure E.23: Shear wave velocities of Profile022

E.24 Profile023 - Port of Richmond POR-4-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sandy clay	1.0	23.0	17.6	81.0	81.0
sandy clay	1.0	34.0	16.5	90.0	90.0
sandy clay	1.0	32.0	17.6	95.0	95.0
sandy clay	1.0	19.0	16.4	98.0	98.0
sandy clay	1.0	27.0	17.4	101.0	101.0
silty sand	1.0	10.0	19.1	122.0	122.0
silty sand	1.0	17.0	16.0	152.0	152.0
old bay clay	4.5	20.0	19.6	209.0	209.0
old bay clay	4.5	20.0	20.2	322.0	322.0
weathered bedrock	3.0	0.0	22.4	731.0	1096.0
weathered bedrock	3.1	0.0	24.8	792.0	1715.0
weathered bedrock	3.1	0.0	24.4	853.0	2416.0

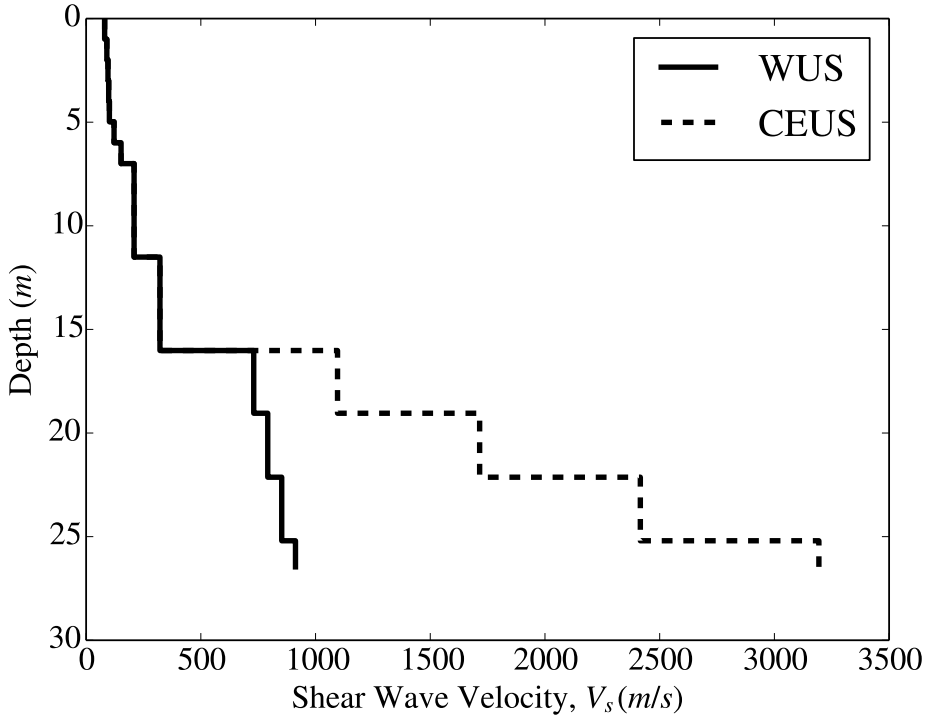


Figure E.24: Shear wave velocities of Profile023

E.25 Profile024 - Miller Farm-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silt	1.1	11.0	15.3	152.0	152.0
silt	1.1	5.0	14.8	164.0	164.0
silt	1.1	9.0	15.7	173.0	173.0
silt	1.1	7.0	14.8	180.0	180.0
silty sand	0.9	15.0	17.0	148.0	148.0
silty sand	1.9	6.0	18.5	179.0	179.0
silty sand	0.9	9.0	16.2	159.0	159.0
silty sand	0.9	9.0	19.4	198.0	198.0
silty sand	0.9	11.0	17.8	219.0	219.0
silty sand	0.9	19.0	16.4	250.0	250.0
silty sand	0.9	10.0	16.1	274.0	274.0
lean clay	1.0	34.0	14.9	334.0	334.0
sand	0.4	0.0	17.5	365.0	365.0
sandy gravel	4.3	0.0	21.3	411.0	411.0
sandy gravel	4.3	0.0	22.6	468.0	468.0
clay	9.0	23.0	14.6	247.0	247.0
fine sand	3.0	0.0	20.3	300.0	300.0
silty clay	5.1	5.0	16.9	352.0	352.0
sandy gravel	19.0	0.0	19.4	498.0	498.0
sandy clay	3.0	33.0	15.4	689.0	689.0
sandy gravel	12.0	0.0	21.7	698.0	698.0
sandy gravelly clay	19.0	27.0	18.4	524.0	524.0
sandy clay	2.1	17.0	15.1	540.0	540.0
gravel	0.9	0.0	20.4	701.0	701.0
sandy clay	28.8	23.0	17.2	710.0	710.0
sandy clay	28.8	7.0	16.4	720.0	720.0
sandy clay	14.4	16.0	15.3	731.0	731.0
weathered bedrock	7.6	0.0	24.3	792.0	1189.0

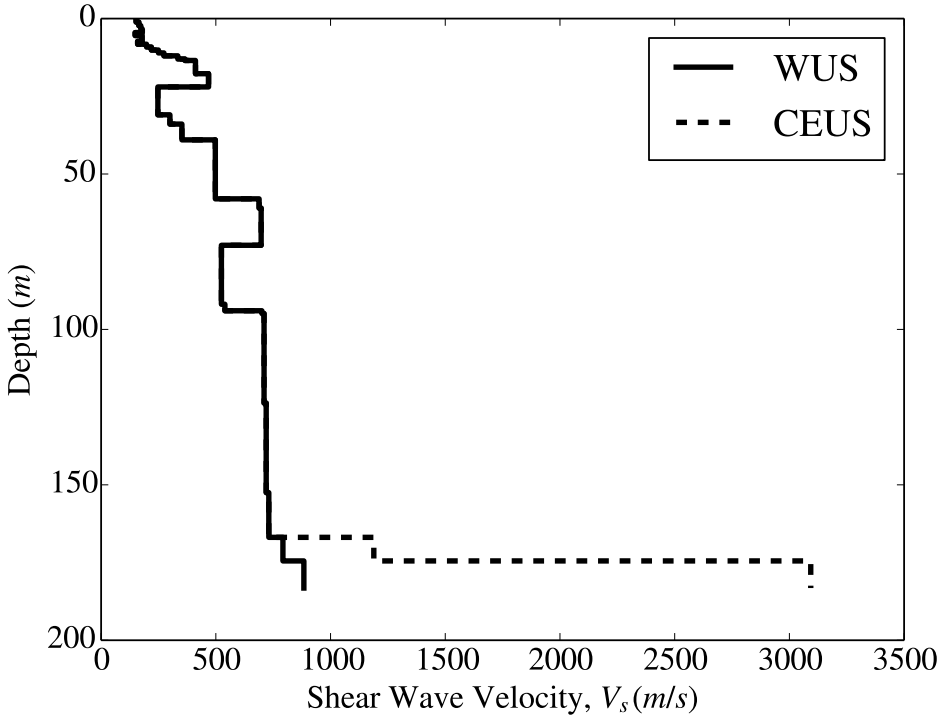


Figure E.25: Shear wave velocities of Profile024

E.26 Profile025 - Farris Farm-Loma Prieta Earthquake (1989)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sandy silt	1.0	13.0	18.2	153.0	153.0
sandy silt	1.0	4.0	17.4	165.0	165.0
sandy silt	1.0	13.0	17.9	172.0	172.0
sandy silt	1.0	17.0	17.4	180.0	180.0
sand	1.0	0.0	17.9	149.0	149.0
sand	1.0	0.0	21.1	164.0	164.0
sand	1.0	0.0	20.1	198.0	198.0
sand	1.0	0.0	18.9	219.0	219.0
sand	2.0	0.0	19.7	250.0	250.0
sand	1.0	0.0	20.5	274.0	274.0
lean clay	1.0	22.0	14.5	320.0	320.0
lean clay	1.0	23.0	16.0	380.0	380.0
sandy silt	4.0	10.0	15.3	350.0	350.0
sandy gravel	5.0	0.0	19.6	437.0	437.0
clay	9.0	19.0	17.0	247.0	247.0
fine sand	3.0	0.0	20.5	300.0	300.0
silty clay	5.0	7.0	15.1	352.0	352.0
sandy gravel	18.9	0.0	20.2	498.0	498.0
sandy clay	3.1	10.0	16.6	688.0	688.0
sandy gravel	12.0	0.0	19.7	698.0	698.0
sandy gravelly clay	18.9	23.0	16.6	524.0	524.0
sandy clay	2.1	8.0	16.1	539.0	539.0
gravel	1.0	0.0	22.0	701.0	701.0
sandy clay	28.8	10.0	16.9	709.0	709.0
sandy clay	28.8	23.0	15.5	719.0	719.0
sandy clay	14.4	34.0	18.1	730.0	730.0
weathered bedrock	7.5	0.0	23.2	792.0	1188.0

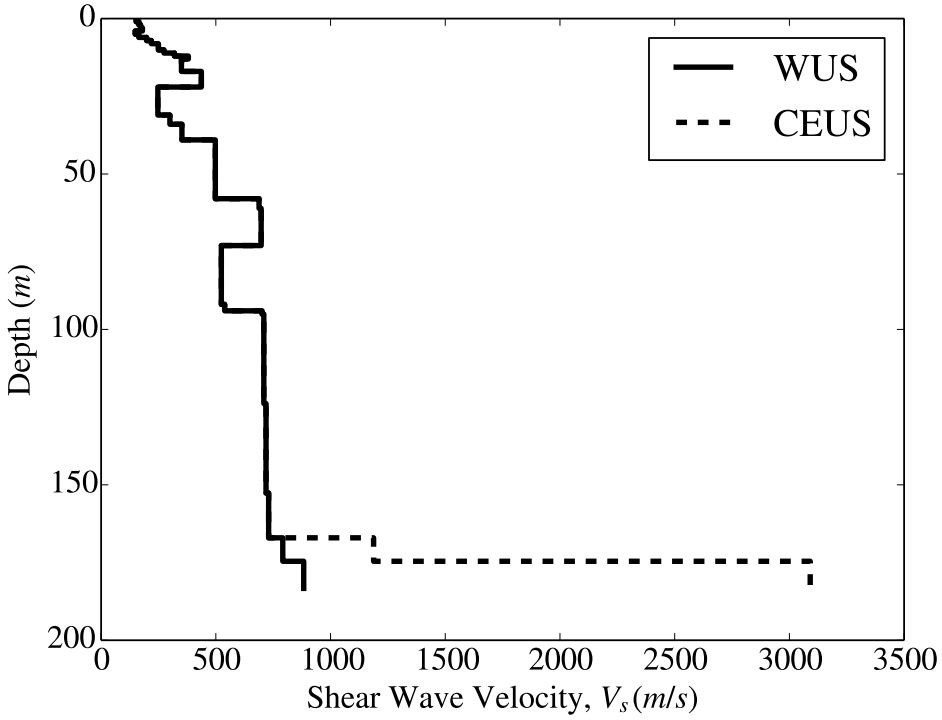


Figure E.26: Shear wave velocities of Profile025

E.27 Profile026 - Wildlife Site-Elmore Ranch (1987)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silt	1.1	12.0	16.9	90.0	90.0
silt	1.1	13.0	14.3	92.0	92.0
silty sand	1.1	17.0	16.2	108.0	108.0
silty sand	1.1	6.0	18.8	113.0	113.0
silty sand	1.1	16.0	15.9	126.0	126.0
silty sand	1.1	11.0	17.9	140.0	140.0
silty clay	0.9	7.0	15.5	152.0	152.0

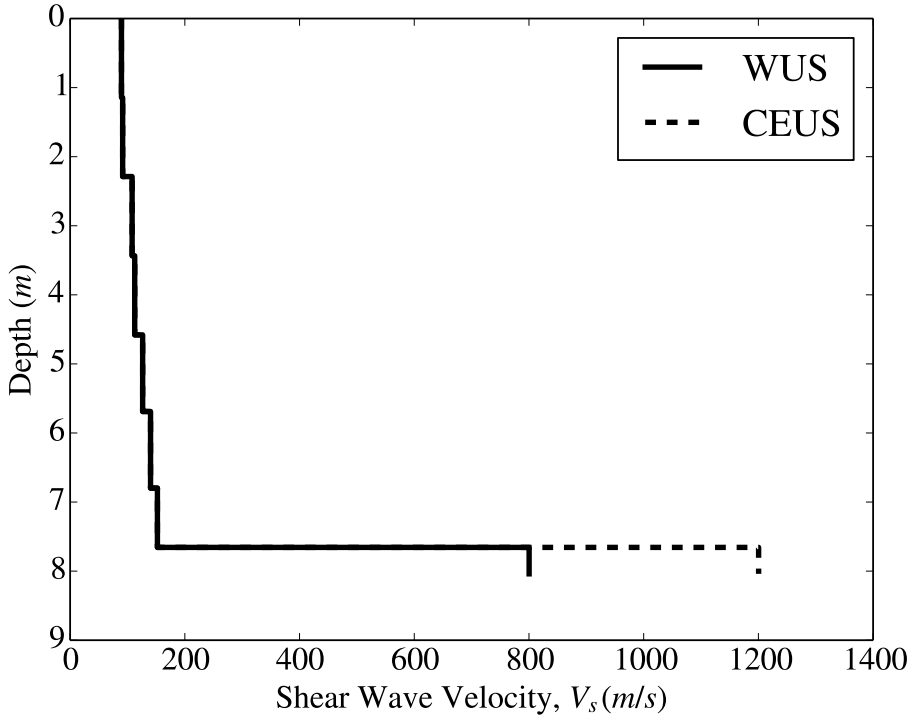


Figure E.27: Shear wave velocities of Profile026

E.28 Profile027 - Wildlife Site-Superstition Hills (1987)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silt	1.1	12.0	15.5	89.0	89.0
silt	1.1	15.0	17.1	92.0	92.0
silty sand	1.1	5.0	19.8	107.0	107.0
silty sand	1.1	19.0	17.9	111.0	111.0
silty sand	1.1	13.0	16.4	126.0	126.0
silty sand	1.1	5.0	19.8	140.0	140.0
silty clay	0.8	5.0	15.5	153.0	153.0

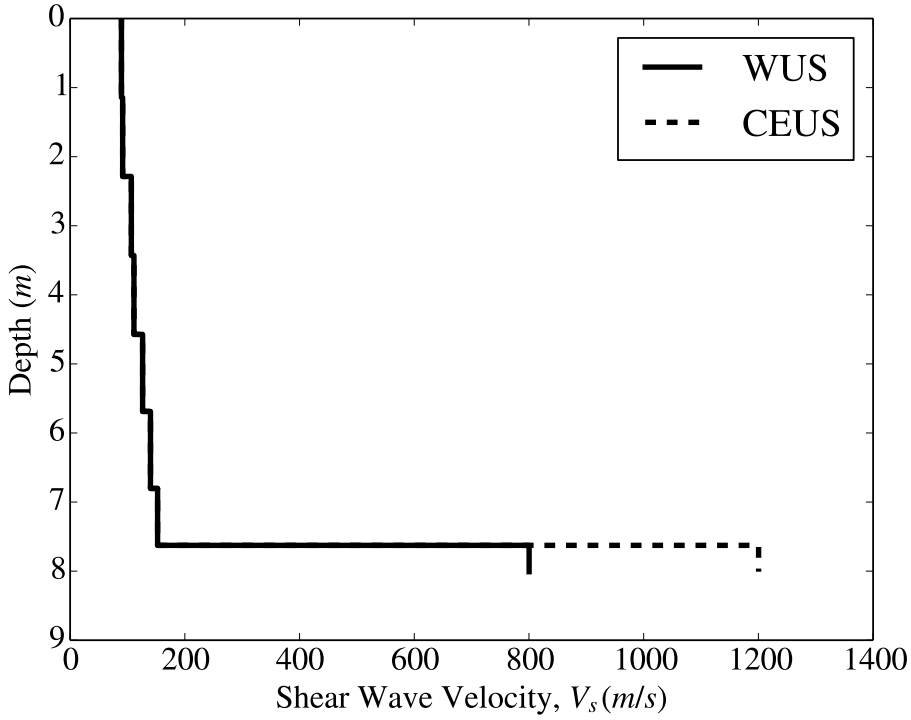


Figure E.28: Shear wave velocities of Profile027

E.29 Profile028 - Radio Tower B1-Superstition Hills (1987)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty clay	1.0	6.0	15.4	130.0	130.0
silty clay	1.0	6.0	14.4	144.0	144.0
silty clay	1.0	5.0	16.2	152.0	152.0
sand	1.0	0.0	20.7	125.0	125.0
sand	1.0	0.0	19.4	152.0	152.0
sand	1.0	0.0	20.2	175.0	175.0
silty clay	0.9	4.0	16.4	175.0	175.0
silty clay	1.8	4.0	14.8	192.0	192.0
silty clay	0.9	6.0	15.6	210.0	210.0
silty clay	1.8	6.0	16.0	226.0	226.0
silty clay	0.9	5.0	16.1	228.0	228.0
sand	1.1	0.0	17.8	198.0	198.0
silty clay	12.2	5.0	16.2	276.0	276.0
silty clay	12.2	4.0	16.0	283.0	283.0
silty clay	12.2	7.0	17.0	290.0	290.0
silty clay	12.2	4.0	16.4	300.0	300.0
silty clay	12.2	5.0	16.2	309.0	309.0
silty clay	12.2	4.0	14.5	320.0	320.0
silty clay	12.2	5.0	15.5	355.0	355.0
silty clay	12.2	7.0	15.6	373.0	373.0
silty clay	12.2	4.0	16.6	387.0	387.0
silty clay	12.2	4.0	16.3	401.0	401.0
silty clay	12.2	5.0	14.5	417.0	417.0
silty clay	12.2	5.0	15.2	441.0	441.0
clayey silt with sand	12.8	6.0	15.4	449.0	449.0
clayey silt with sand	12.8	6.0	15.8	457.0	457.0
clayey silt with sand	12.8	6.0	16.9	465.0	465.0
clayey silt with sand	12.8	7.0	15.9	472.0	472.0
clayey silt with sand	12.8	6.0	17.4	478.0	478.0
clayey silt with sand	12.8	4.0	15.3	487.0	487.0
clayey silt with sand	12.8	5.0	15.1	503.0	503.0
weathered bedrock	9.0	0.0	22.4	685.0	1028.0

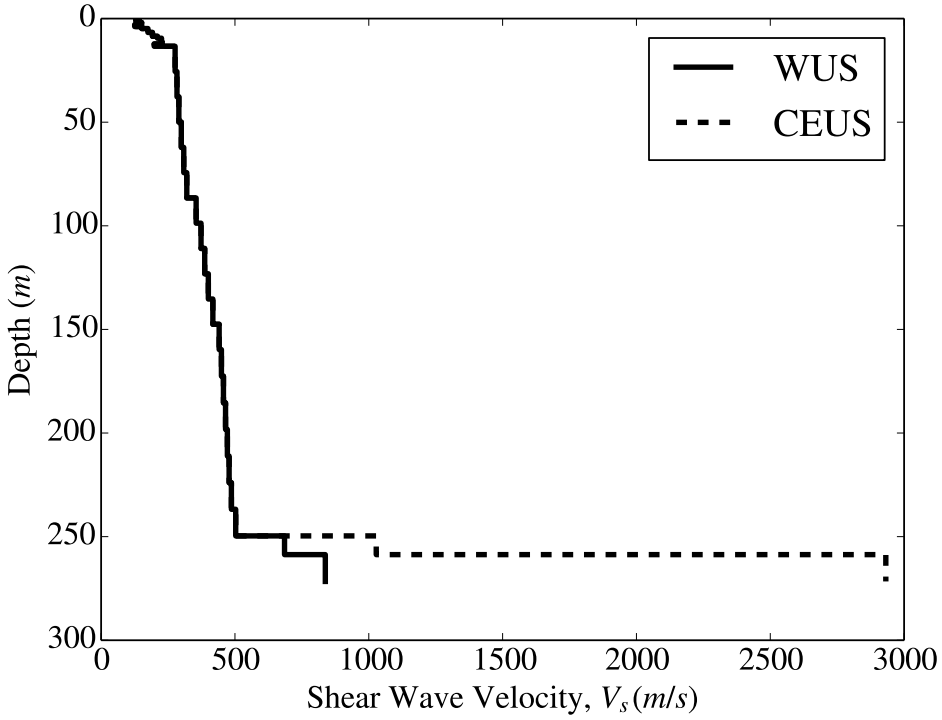


Figure E.29: Shear wave velocities of Profile028

E.30 Profile029 - Radio Tower B2-Superstition Hills (1987)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty clay	1.8	6.0	14.9	140.0	140.0
sand	1.1	0.0	21.6	151.0	151.0
silty clay	2.0	5.0	17.0	176.0	176.0
silty clay	2.0	7.0	17.3	191.0	191.0
silty clay	1.0	7.0	16.0	199.0	199.0
silty clay	1.0	4.0	14.6	206.0	206.0
silty clay	1.0	6.0	17.2	213.0	213.0
silty clay	2.0	5.0	16.0	221.0	221.0
silty clay	1.0	7.0	15.2	229.0	229.0
sand	1.0	0.0	18.9	175.0	175.0
sand	1.0	0.0	18.5	225.0	225.0
silty clay	13.2	5.0	14.9	277.0	277.0
silty clay	13.2	6.0	14.9	286.0	286.0
silty clay	13.2	5.0	16.4	295.0	295.0
silty clay	13.2	4.0	14.2	304.0	304.0
silty clay	13.2	5.0	14.5	311.0	311.0
silty clay	13.2	7.0	16.5	335.0	335.0
silty clay	13.2	5.0	14.3	364.0	364.0
silty clay	13.2	6.0	15.0	382.0	382.0
silty clay	13.2	6.0	15.7	401.0	401.0
silty clay	13.2	7.0	14.6	418.0	418.0
silty clay	13.2	5.0	15.0	441.0	441.0
clayey silt with sand	12.8	5.0	16.3	449.0	449.0
clayey silt with sand	12.8	7.0	17.5	457.0	457.0
clayey silt with sand	12.8	6.0	16.4	465.0	465.0
clayey silt with sand	12.8	7.0	17.0	471.0	471.0
clayey silt with sand	12.8	6.0	17.2	478.0	478.0
clayey silt with sand	12.8	4.0	16.8	486.0	486.0
clayey silt with sand	12.8	5.0	16.9	502.0	502.0
weathered bedrock	9.0	0.0	22.5	685.0	1027.0

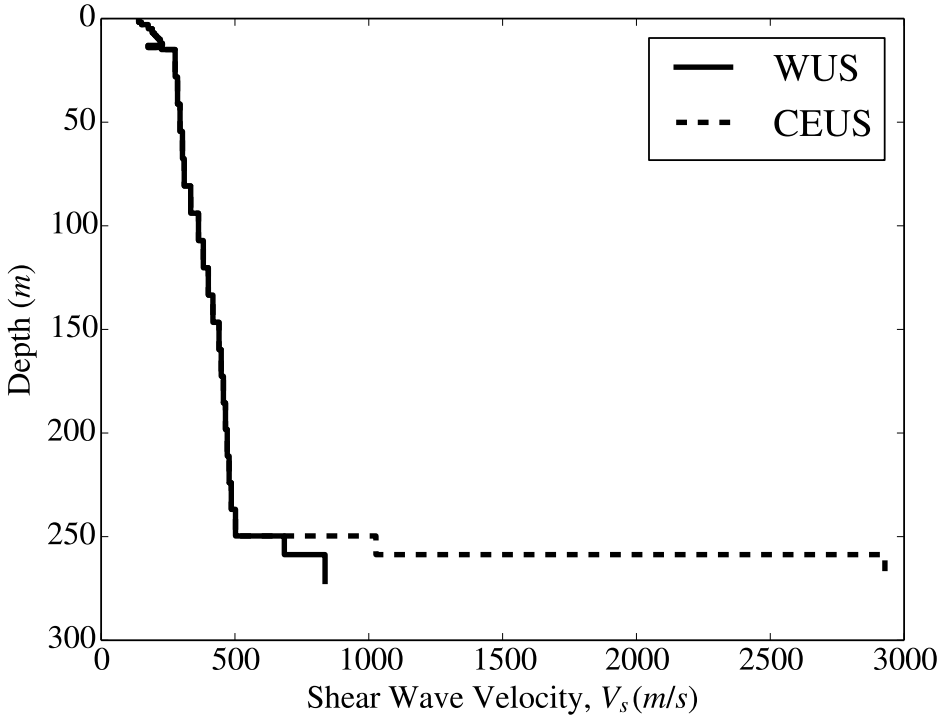


Figure E.30: Shear wave velocities of Profile029

E.31 Profile030 - McKim Ranch A-Superstition Hills (1987)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty sand	1.0	16.0	20.4	156.0	156.0
silty sand	1.0	8.0	17.3	166.0	166.0
silty sand	2.0	15.0	19.4	172.0	172.0
silty sand	1.0	5.0	17.9	180.0	180.0
silty clay	1.0	5.0	14.5	168.0	168.0
silty clay	1.0	6.0	16.7	177.0	177.0
silty clay	1.0	4.0	14.5	184.0	184.0
silty clay	1.0	5.0	17.1	192.0	192.0
silty clay	1.0	6.0	15.1	202.0	202.0
silty clay	1.0	6.0	15.2	213.0	213.0
silty clay	4.5	6.0	16.4	186.0	186.0
silty clay	4.5	7.0	16.9	217.0	217.0
silty clay	12.7	6.0	14.8	276.0	276.0
silty clay	12.7	4.0	15.0	285.0	285.0
silty clay	12.7	6.0	14.4	294.0	294.0
silty clay	12.7	6.0	17.1	304.0	304.0
silty clay	12.7	6.0	16.7	315.0	315.0
silty clay	12.7	5.0	15.0	355.0	355.0
silty clay	12.7	6.0	15.2	369.0	369.0
silty clay	12.7	5.0	15.8	382.0	382.0
silty clay	12.7	5.0	17.0	401.0	401.0
silty clay	12.7	6.0	14.2	417.0	417.0
silty clay	12.7	6.0	16.8	441.0	441.0
clayey silt with sand	12.2	5.0	15.3	449.0	449.0
clayey silt with sand	12.2	5.0	17.8	454.0	454.0
clayey silt with sand	12.2	6.0	15.1	460.0	460.0
clayey silt with sand	12.2	4.0	17.8	467.0	467.0
clayey silt with sand	12.2	7.0	16.9	476.0	476.0
clayey silt with sand	12.2	6.0	17.9	487.0	487.0
clayey silt with sand	12.2	4.0	15.3	502.0	502.0
weathered bedrock	9.1	0.0	24.5	686.0	1028.0

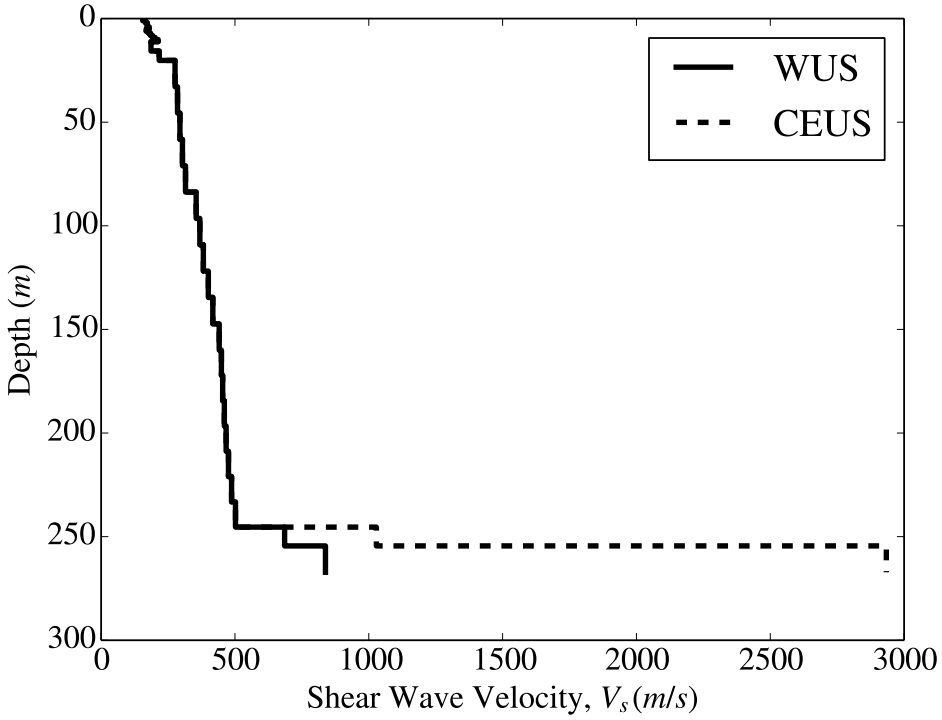


Figure E.31: Shear wave velocities of Profile030

E.32 Profile031 - Kornbloom-Superstition Hills (1987)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty clay	1.7	6.0	16.8	131.0	131.0
silt	1.1	13.0	16.7	124.0	124.0
silt	1.1	15.0	16.4	133.0	133.0
silt	1.1	4.0	16.5	144.0	144.0
silt	1.1	5.0	14.8	153.0	153.0
silty clay	1.1	6.0	16.9	175.0	175.0
silty clay	1.1	5.0	16.9	198.0	198.0
silt	0.9	16.0	15.1	175.0	175.0
silt	0.9	8.0	16.3	225.0	225.0
sand	0.9	0.0	17.9	225.0	225.0
sand	0.9	0.0	18.2	249.0	249.0
silt	3.1	6.0	17.1	265.0	265.0
silt	3.1	17.0	15.8	308.0	308.0
silty clay	12.9	5.0	16.1	276.0	276.0
silty clay	12.9	6.0	15.5	285.0	285.0
silty clay	12.9	5.0	16.8	295.0	295.0
silty clay	12.9	6.0	16.0	304.0	304.0
silty clay	12.9	4.0	16.1	315.0	315.0
silty clay	12.9	7.0	14.4	350.0	350.0
silty clay	12.9	7.0	15.6	364.0	364.0
silty clay	12.9	6.0	16.6	382.0	382.0
silty clay	12.9	5.0	16.0	401.0	401.0
silty clay	12.9	5.0	14.5	418.0	418.0
silty clay	12.9	5.0	15.7	441.0	441.0
clayey silt with sand	12.8	5.0	15.0	449.0	449.0
clayey silt with sand	12.8	6.0	16.2	454.0	454.0
clayey silt with sand	12.8	5.0	17.1	460.0	460.0
clayey silt with sand	12.8	7.0	16.7	467.0	467.0
clayey silt with sand	12.8	4.0	16.4	476.0	476.0
clayey silt with sand	12.8	4.0	17.3	484.0	484.0
clayey silt with sand	12.8	5.0	15.6	496.0	496.0
weathered bedrock	9.1	0.0	24.7	686.0	1028.0

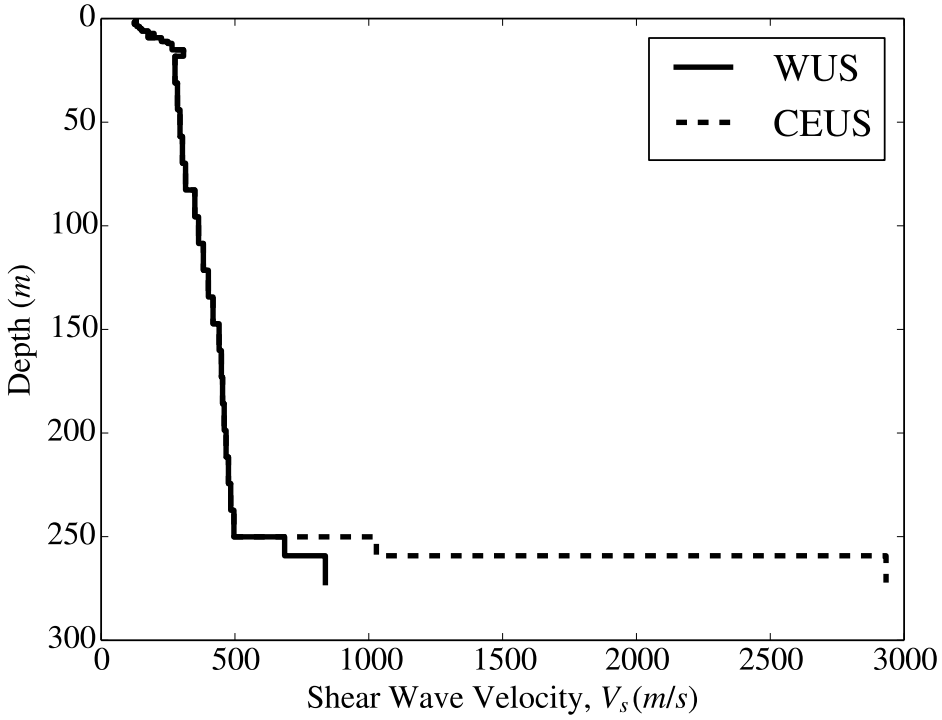


Figure E.32: Shear wave velocities of Profile031

E.33 Profile032 - River Park A & C-Superstition Hills (1987)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silt and sandy silt	1.8	7.0	15.3	130.0	130.0
silty clay	1.6	5.0	14.8	150.0	150.0
sand	1.3	0.0	18.5	200.0	200.0
sand	1.3	0.0	20.8	208.0	208.0
sand	1.0	0.0	17.7	213.0	213.0
sand	1.0	0.0	20.4	220.0	220.0
sand	1.0	0.0	21.8	226.0	226.0
sand	1.0	0.0	20.1	231.0	231.0
sand	1.0	0.0	20.0	233.0	233.0
sand	1.0	0.0	18.1	236.0	236.0
silty clay	12.3	6.0	14.5	276.0	276.0
silty clay	12.3	4.0	16.0	283.0	283.0
silty clay	12.3	6.0	16.1	290.0	290.0
silty clay	12.3	5.0	17.0	299.0	299.0
silty clay	12.3	5.0	15.6	307.0	307.0
silty clay	12.3	6.0	15.9	315.0	315.0
silty clay	12.3	5.0	15.7	355.0	355.0
silty clay	12.3	7.0	16.4	368.0	368.0
silty clay	12.3	4.0	16.1	382.0	382.0
silty clay	12.3	4.0	14.4	401.0	401.0
silty clay	12.3	5.0	15.9	417.0	417.0
silty clay	12.3	5.0	15.1	441.0	441.0
clayey silt with sand	12.8	6.0	18.0	449.0	449.0
clayey silt with sand	12.8	5.0	17.2	457.0	457.0
clayey silt with sand	12.8	6.0	16.5	465.0	465.0
clayey silt with sand	12.8	6.0	15.7	472.0	472.0
clayey silt with sand	12.8	6.0	17.5	478.0	478.0
clayey silt with sand	12.8	4.0	15.9	486.0	486.0
clayey silt with sand	12.8	5.0	16.6	502.0	502.0
weathered bedrock	9.1	0.0	25.1	685.0	1028.0

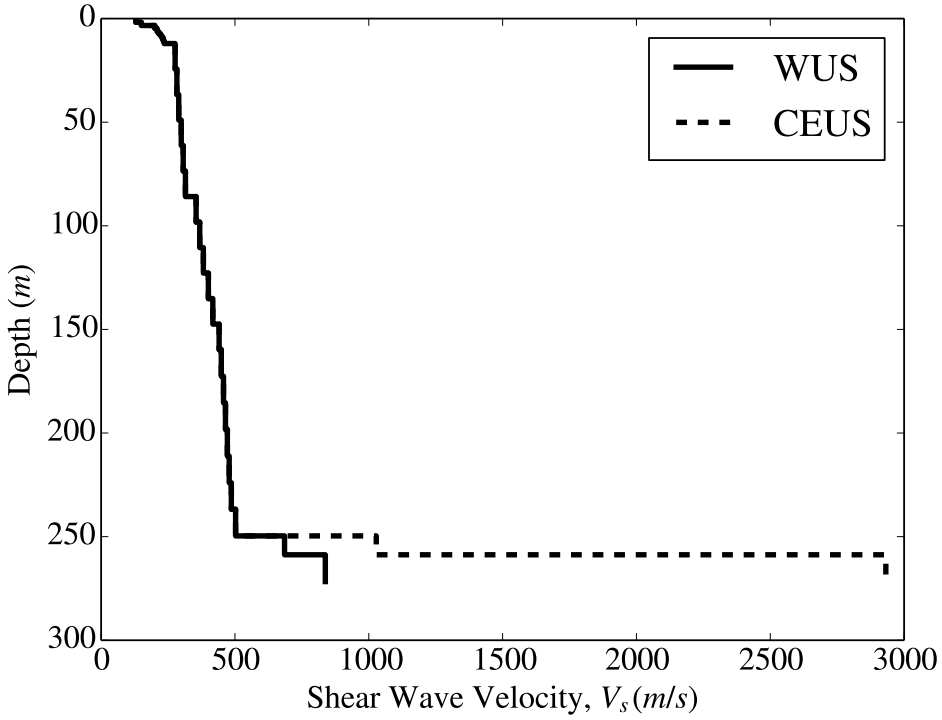


Figure E.33: Shear wave velocities of Profile032

E.34 Profile033 - Heber Road A1-Superstition Hills (1987)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
fill sand	1.0	0.0	19.9	129.0	129.0
fill sand	1.0	0.0	20.9	159.0	159.0
fill sand	1.0	0.0	20.9	183.0	183.0
fill sand	1.0	0.0	21.6	205.0	205.0
fill sand	1.0	0.0	21.4	228.0	228.0
clay	1.4	17.0	15.2	138.0	138.0
sand	1.4	0.0	17.6	200.0	200.0
clay	0.7	18.0	17.3	229.0	229.0
sand	0.7	0.0	19.7	258.0	258.0
clay	0.5	14.0	16.5	273.0	273.0
sand	0.9	0.0	18.4	305.0	305.0
silty clay	12.5	5.0	17.3	276.0	276.0
silty clay	12.5	5.0	16.5	283.0	283.0
silty clay	12.5	5.0	15.6	290.0	290.0
silty clay	12.5	4.0	16.5	299.0	299.0
silty clay	12.5	6.0	14.4	306.0	306.0
silty clay	12.5	5.0	16.3	315.0	315.0
silty clay	12.5	6.0	17.2	355.0	355.0
silty clay	12.5	5.0	16.4	368.0	368.0
silty clay	12.5	6.0	16.3	382.0	382.0
silty clay	12.5	6.0	16.9	400.0	400.0
silty clay	12.5	4.0	17.1	417.0	417.0
silty clay	12.5	5.0	16.5	440.0	440.0
clayey silt with sand	12.9	7.0	17.5	449.0	449.0
clayey silt with sand	12.9	5.0	15.8	457.0	457.0
clayey silt with sand	12.9	6.0	18.0	464.0	464.0
clayey silt with sand	12.9	5.0	18.0	471.0	471.0
clayey silt with sand	12.9	6.0	17.6	478.0	478.0
clayey silt with sand	12.9	6.0	14.9	486.0	486.0
clayey silt with sand	12.9	5.0	18.0	502.0	502.0
weathered bedrock	9.1	0.0	22.6	731.0	1096.0
weathered bedrock	9.2	0.0	24.1	822.0	2055.0

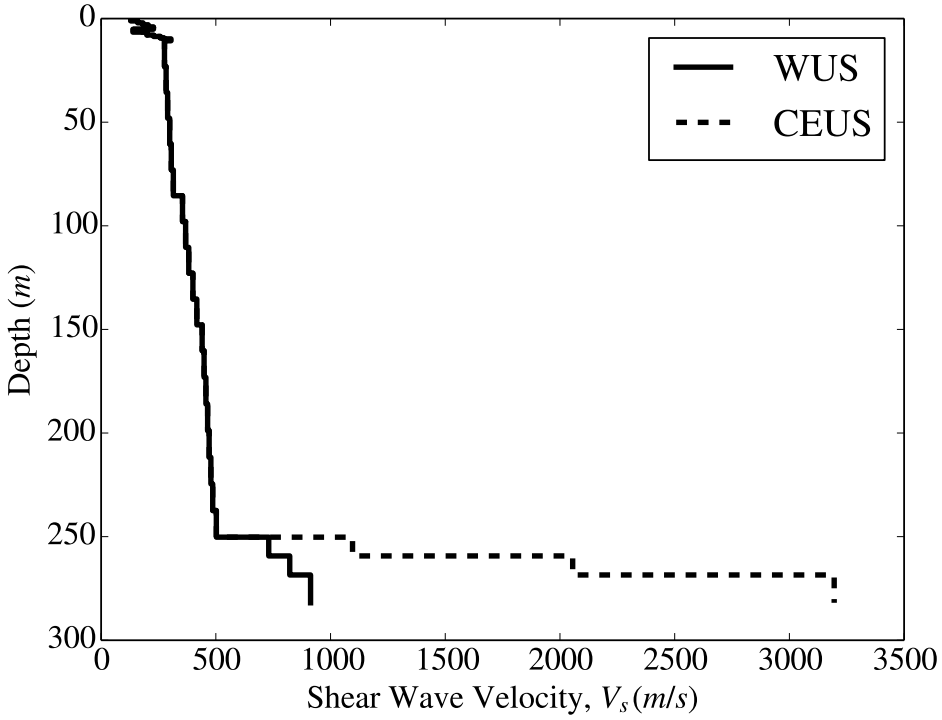


Figure E.34: Shear wave velocities of Profile033

E.35 Profile034 - Heber Road A2-Superstition Hills (1987)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
fill sand	1.0	0.0	20.5	145.0	145.0
fill sand	1.0	0.0	19.7	167.0	167.0
fill sand	1.0	0.0	21.9	183.0	183.0
fill sand	1.0	0.0	21.2	205.0	205.0
fill sand	1.0	0.0	19.8	228.0	228.0
fill sand	1.0	0.0	21.7	137.0	137.0
clay	1.1	26.0	16.8	200.0	200.0
sand	1.4	0.0	19.0	245.0	245.0
clay	0.6	32.0	14.5	274.0	274.0
sand	0.7	0.0	20.0	305.0	305.0
clay	1.2	18.0	16.7	274.0	274.0
clay	1.2	12.0	16.0	319.0	319.0
sand	1.5	0.0	18.0	305.0	305.0
clay	12.2	34.0	15.3	276.0	276.0
clay	12.2	26.0	16.0	283.0	283.0
clay	12.2	10.0	16.5	290.0	290.0
clay	12.2	16.0	16.1	299.0	299.0
clay	12.2	18.0	16.2	309.0	309.0
clay	12.2	23.0	15.4	320.0	320.0
clay	12.2	30.0	16.4	355.0	355.0
clay	12.2	11.0	15.0	369.0	369.0
clay	12.2	10.0	16.1	382.0	382.0
clay	12.2	30.0	15.0	401.0	401.0
clay	12.2	15.0	15.5	418.0	418.0
clay	12.2	10.0	16.0	441.0	441.0
silty clay	13.5	5.0	16.2	452.0	452.0
silty clay	13.5	6.0	16.8	460.0	460.0
silty clay	13.5	5.0	14.8	467.0	467.0
silty clay	13.5	5.0	14.4	474.0	474.0
silty clay	13.5	4.0	15.2	480.0	480.0
silty clay	13.5	5.0	16.1	502.0	502.0
weathered bedrock	13.7	0.0	23.0	731.0	1097.0

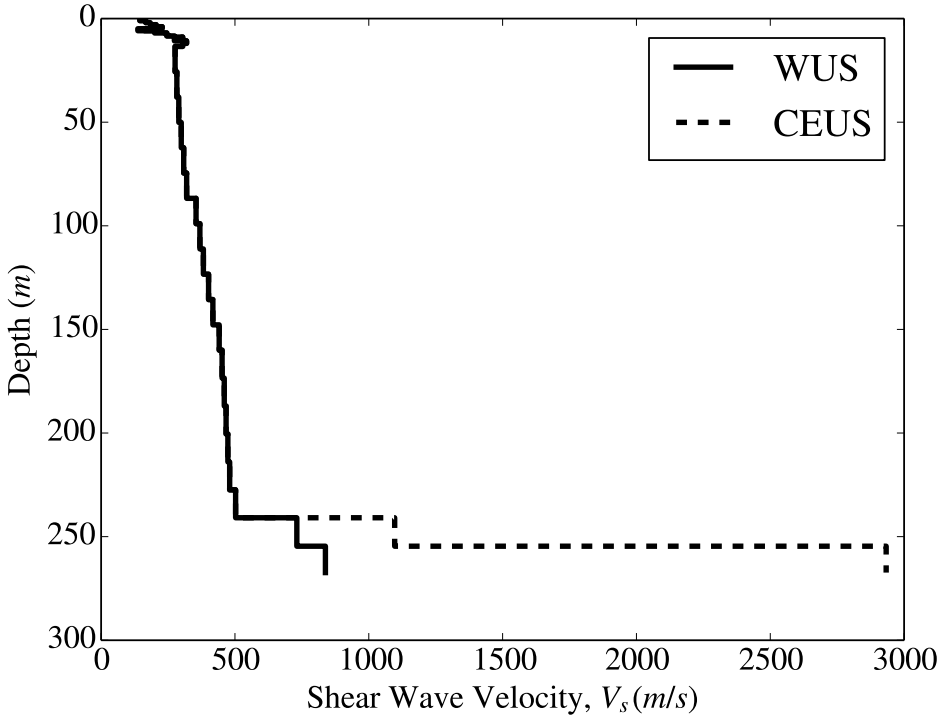


Figure E.35: Shear wave velocities of Profile034

E.36 Profile035 - Heber Road A3-Superstition Hills (1987)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
fill sand	1.0	0.0	21.8	137.0	137.0
fill sand	1.0	0.0	21.3	167.0	167.0
fill sand	1.0	0.0	20.7	183.0	183.0
fill sand	1.0	0.0	19.7	205.0	205.0
fill sand	1.0	0.0	21.3	229.0	229.0
clay	1.4	11.0	14.2	239.0	239.0
sand	0.5	0.0	21.5	259.0	259.0
clay	1.2	9.0	15.1	274.0	274.0
sand	1.4	0.0	20.4	305.0	305.0
clay	1.3	33.0	15.8	274.0	274.0
clay	1.3	27.0	16.1	319.0	319.0
sand	1.4	0.0	19.6	305.0	305.0
silty clay	12.2	7.0	16.8	277.0	277.0
silty clay	12.2	5.0	15.4	283.0	283.0
silty clay	12.2	5.0	15.7	290.0	290.0
silty clay	12.2	7.0	14.4	300.0	300.0
silty clay	12.2	7.0	15.5	309.0	309.0
silty clay	12.2	5.0	14.3	320.0	320.0
silty clay	12.2	7.0	14.2	355.0	355.0
silty clay	12.2	7.0	14.9	369.0	369.0
silty clay	12.2	6.0	14.5	382.0	382.0
silty clay	12.2	7.0	15.3	401.0	401.0
silty clay	12.2	4.0	15.1	417.0	417.0
silty clay	12.2	6.0	15.5	441.0	441.0
clayey silt with sand	12.8	6.0	15.0	449.0	449.0
clayey silt with sand	12.8	5.0	16.5	457.0	457.0
clayey silt with sand	12.8	6.0	15.1	465.0	465.0
clayey silt with sand	12.8	5.0	15.3	471.0	471.0
clayey silt with sand	12.8	6.0	17.2	478.0	478.0
clayey silt with sand	12.8	6.0	15.3	486.0	486.0
clayey silt with sand	12.8	5.0	15.2	502.0	502.0
weathered bedrock	9.0	0.0	22.3	731.0	1096.0
weathered bedrock	9.2	0.0	24.0	822.0	2055.0

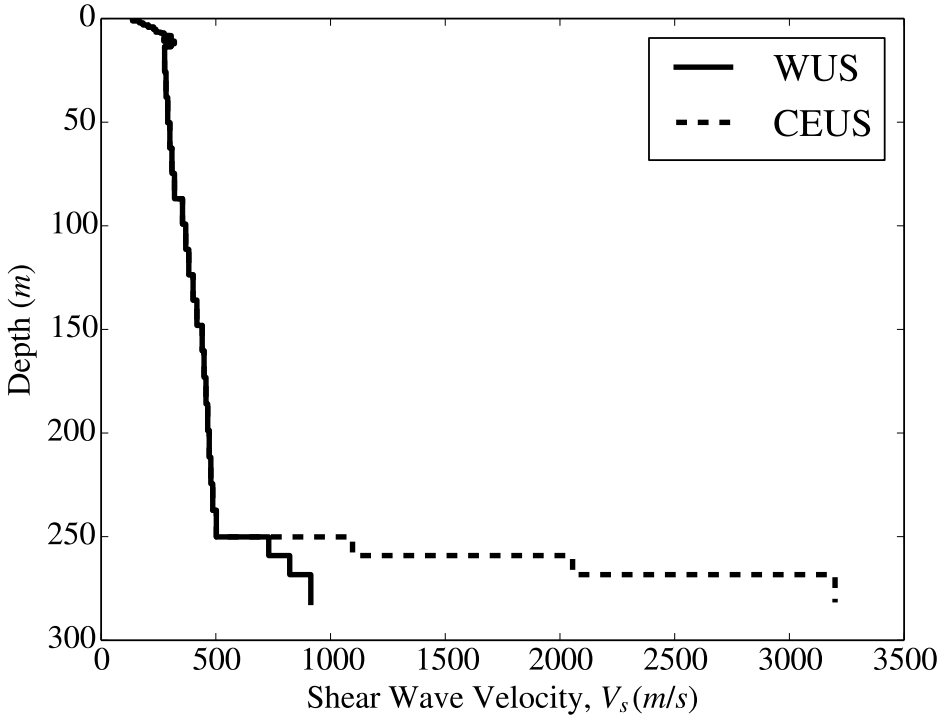


Figure E.36: Shear wave velocities of Profile035

E.37 Profile036 - Kornbloom-Westmorland (1981)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty clay	1.6	5.0	16.3	131.0	131.0
silt	1.1	4.0	15.5	125.0	125.0
silt	1.1	8.0	14.9	139.0	139.0
silt	1.1	7.0	15.8	143.0	143.0
silt	1.1	20.0	14.3	153.0	153.0
silty clay	1.1	5.0	14.8	176.0	176.0
silty clay	1.1	5.0	16.5	198.0	198.0
silt	1.8	7.0	17.0	200.0	200.0
sand	1.0	0.0	18.7	226.0	226.0
sand	1.0	0.0	19.9	250.0	250.0
silt	5.9	18.0	14.5	286.0	286.0
silty clay	12.9	6.0	16.6	276.0	276.0
silty clay	12.9	6.0	14.8	286.0	286.0
silty clay	12.9	6.0	16.3	295.0	295.0
silty clay	12.9	5.0	15.2	304.0	304.0
silty clay	12.9	5.0	15.0	316.0	316.0
silty clay	12.9	4.0	14.6	350.0	350.0
silty clay	12.9	5.0	17.2	364.0	364.0
silty clay	12.9	4.0	15.9	382.0	382.0
silty clay	12.9	7.0	15.4	401.0	401.0
silty clay	12.9	5.0	17.1	418.0	418.0
silty clay	12.9	7.0	15.0	441.0	441.0
clayey silt with sand	12.9	7.0	15.3	450.0	450.0
clayey silt with sand	12.9	6.0	17.1	454.0	454.0
clayey silt with sand	12.9	4.0	16.6	461.0	461.0
clayey silt with sand	12.9	5.0	16.6	467.0	467.0
clayey silt with sand	12.9	6.0	16.0	476.0	476.0
clayey silt with sand	12.9	5.0	17.7	484.0	484.0
clayey silt with sand	12.9	5.0	16.8	497.0	497.0
weathered bedrock	9.1	0.0	24.9	686.0	1028.0

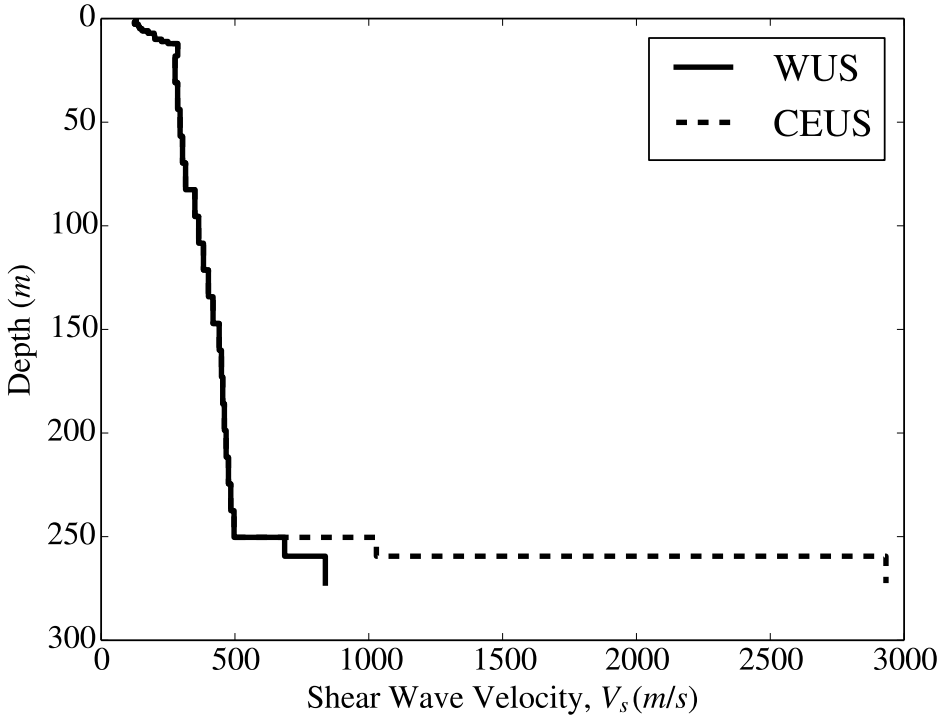


Figure E.37: Shear wave velocities of Profile036

E.38 Profile037 - McKim Ranch-Westmorland (1981)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty sand	1.0	14.0	17.1	152.0	152.0
silty sand	1.0	16.0	17.3	159.0	159.0
silty sand	2.0	17.0	16.4	169.0	169.0
silty sand	1.0	8.0	18.6	181.0	181.0
silty clay	1.0	6.0	14.6	165.0	165.0
silty clay	2.0	6.0	14.4	178.0	178.0
silty clay	1.0	6.0	17.1	188.0	188.0
silty clay	1.0	4.0	16.6	201.0	201.0
silty clay	1.0	5.0	16.8	213.0	213.0
silty clay	3.1	6.0	16.5	181.0	181.0
silty clay	3.1	7.0	15.7	203.0	203.0
silty clay	3.1	4.0	16.9	223.0	223.0
silty clay	12.7	6.0	15.6	276.0	276.0
silty clay	12.7	5.0	16.5	285.0	285.0
silty clay	12.7	5.0	15.3	295.0	295.0
silty clay	12.7	6.0	15.3	304.0	304.0
silty clay	12.7	4.0	17.1	315.0	315.0
silty clay	12.7	6.0	16.0	356.0	356.0
silty clay	12.7	7.0	17.1	369.0	369.0
silty clay	12.7	4.0	14.8	383.0	383.0
silty clay	12.7	5.0	15.8	401.0	401.0
silty clay	12.7	5.0	17.0	418.0	418.0
silty clay	12.7	6.0	14.4	441.0	441.0
clayey silt with sand	14.2	6.0	16.5	450.0	450.0
clayey silt with sand	14.2	7.0	17.8	457.0	457.0
clayey silt with sand	14.2	4.0	16.6	465.0	465.0
clayey silt with sand	14.2	5.0	15.2	469.0	469.0
clayey silt with sand	14.2	6.0	16.2	482.0	482.0
clayey silt with sand	14.2	7.0	17.5	503.0	503.0
weathered bedrock	9.0	0.0	23.4	687.0	1030.0

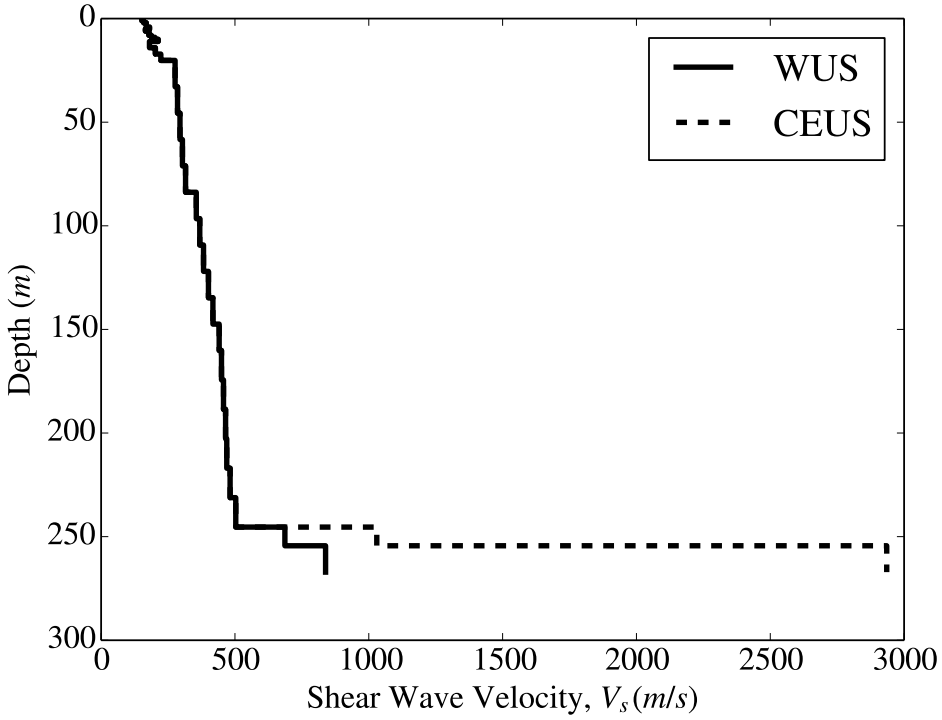


Figure E.38: Shear wave velocities of Profile037

E.39 Profile038 - Radio Tower B1-Westmorland (1981)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty clay	1.0	6.0	15.4	129.0	129.0
silty clay	1.0	7.0	15.2	144.0	144.0
silty clay	1.0	5.0	15.2	153.0	153.0
sand	1.0	0.0	21.2	125.0	125.0
sand	1.0	0.0	21.8	152.0	152.0
sand	1.0	0.0	20.7	175.0	175.0
silty clay	0.9	4.0	15.2	175.0	175.0
silty clay	1.9	6.0	15.4	192.0	192.0
silty clay	0.9	5.0	16.6	210.0	210.0
silty clay	1.9	7.0	14.6	226.0	226.0
silty clay	0.9	5.0	16.0	229.0	229.0
sand	0.9	0.0	19.4	198.0	198.0
silty clay	13.3	4.0	14.7	277.0	277.0
silty clay	13.3	5.0	14.4	286.0	286.0
silty clay	13.3	5.0	15.0	295.0	295.0
silty clay	13.3	5.0	16.6	302.0	302.0
silty clay	13.3	7.0	16.6	309.0	309.0
silty clay	13.3	4.0	17.2	336.0	336.0
silty clay	13.3	6.0	16.7	364.0	364.0
silty clay	13.3	6.0	14.8	382.0	382.0
silty clay	13.3	7.0	17.1	396.0	396.0
silty clay	13.3	7.0	17.2	411.0	411.0
silty clay	13.3	5.0	14.4	441.0	441.0
clayey silt with sand	12.8	7.0	17.8	449.0	449.0
clayey silt with sand	12.8	4.0	16.9	457.0	457.0
clayey silt with sand	12.8	6.0	16.9	464.0	464.0
clayey silt with sand	12.8	7.0	17.2	471.0	471.0
clayey silt with sand	12.8	5.0	16.8	478.0	478.0
clayey silt with sand	12.8	5.0	16.1	486.0	486.0
clayey silt with sand	12.8	5.0	16.6	502.0	502.0
weathered bedrock	9.2	0.0	22.4	685.0	1028.0

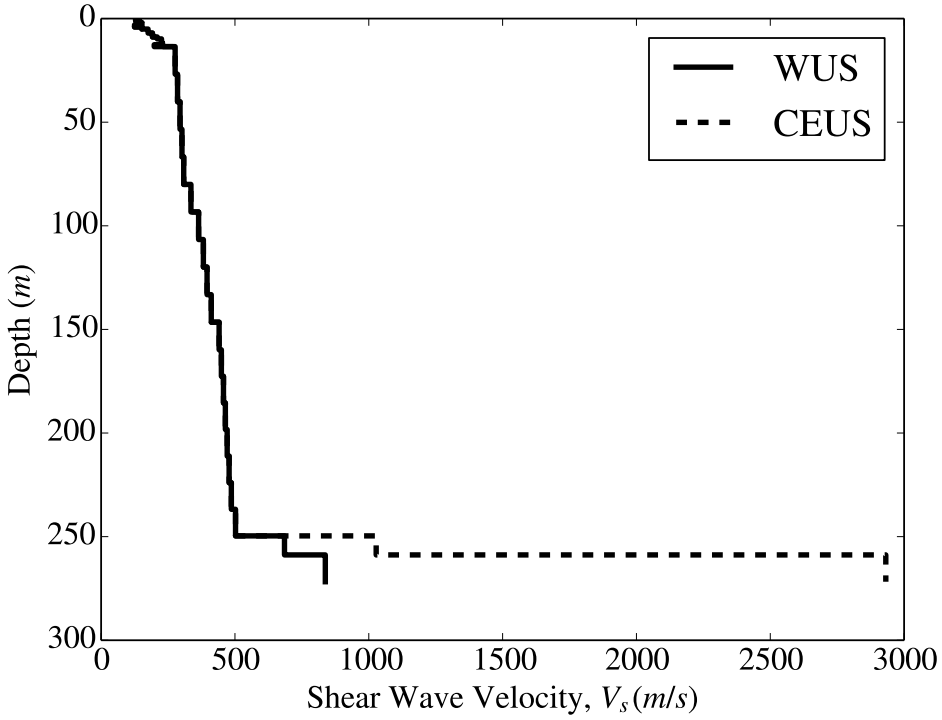


Figure E.39: Shear wave velocities of Profile038

E.40 Profile039 - Radio Tower B2-Westmorland (1981)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty clay	0.9	7.0	17.3	132.0	132.0
silty clay	0.9	6.0	15.0	152.0	152.0
sand	1.1	0.0	19.4	150.0	150.0
silty clay	1.0	5.0	16.8	175.0	175.0
silty clay	1.0	6.0	15.8	182.0	182.0
silty clay	2.0	7.0	14.8	190.0	190.0
silty clay	1.0	4.0	14.5	198.0	198.0
silty clay	1.0	5.0	14.5	205.0	205.0
silty clay	2.0	6.0	16.9	213.0	213.0
silty clay	1.0	5.0	17.0	221.0	221.0
silty clay	1.0	6.0	15.3	228.0	228.0
sand	1.0	0.0	19.0	174.0	174.0
sand	1.0	0.0	17.4	226.0	226.0
silty clay	13.2	7.0	16.2	276.0	276.0
silty clay	13.2	6.0	14.6	286.0	286.0
silty clay	13.2	6.0	16.4	295.0	295.0
silty clay	13.2	5.0	15.7	304.0	304.0
silty clay	13.2	5.0	14.9	310.0	310.0
silty clay	13.2	5.0	16.2	335.0	335.0
silty clay	13.2	5.0	16.4	364.0	364.0
silty clay	13.2	7.0	15.4	382.0	382.0
silty clay	13.2	6.0	16.4	396.0	396.0
silty clay	13.2	4.0	17.3	411.0	411.0
silty clay	13.2	6.0	14.2	441.0	441.0
clayey silt with sand	12.9	6.0	15.8	449.0	449.0
clayey silt with sand	12.9	6.0	15.2	457.0	457.0
clayey silt with sand	12.9	6.0	15.8	465.0	465.0
clayey silt with sand	12.9	6.0	17.4	472.0	472.0
clayey silt with sand	12.9	7.0	17.6	478.0	478.0
clayey silt with sand	12.9	6.0	17.4	487.0	487.0
clayey silt with sand	12.9	6.0	15.8	503.0	503.0
weathered bedrock	8.9	0.0	24.2	685.0	1028.0

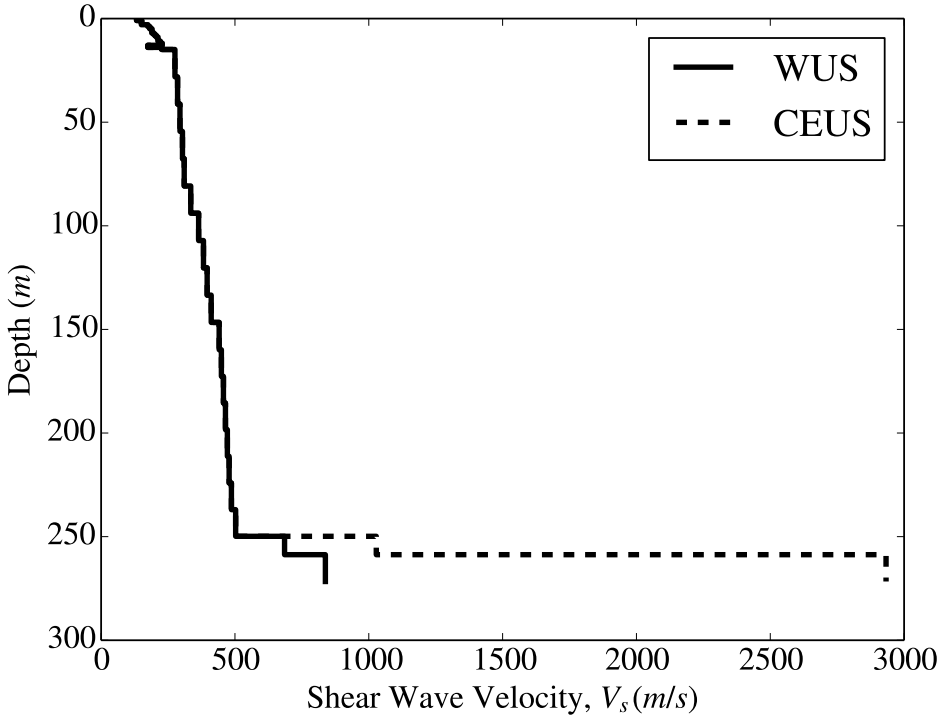


Figure E.40: Shear wave velocities of Profile039

E.41 Profile040 - River Park A & C-Westmorland (1981)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silt	1.0	17.0	16.5	121.0	121.0
silt	1.0	19.0	16.4	138.0	138.0
silty clay	1.4	6.0	15.9	150.0	150.0
sand	1.4	0.0	21.1	197.0	197.0
sand	1.4	0.0	21.3	207.0	207.0
sand	1.0	0.0	18.8	216.0	216.0
sand	1.0	0.0	19.4	223.0	223.0
sand	2.0	0.0	18.0	229.0	229.0
sand	1.0	0.0	17.7	232.0	232.0
sand	1.0	0.0	19.5	236.0	236.0
silty clay	12.3	6.0	14.4	276.0	276.0
silty clay	12.3	6.0	14.9	282.0	282.0
silty clay	12.3	5.0	16.2	290.0	290.0
silty clay	12.3	5.0	14.6	299.0	299.0
silty clay	12.3	5.0	16.8	306.0	306.0
silty clay	12.3	7.0	17.0	315.0	315.0
silty clay	12.3	4.0	16.8	355.0	355.0
silty clay	12.3	7.0	15.3	369.0	369.0
silty clay	12.3	6.0	16.7	382.0	382.0
silty clay	12.3	4.0	14.9	401.0	401.0
silty clay	12.3	7.0	17.0	417.0	417.0
silty clay	12.3	5.0	14.3	441.0	441.0
clayey silt with sand	12.9	7.0	16.1	449.0	449.0
clayey silt with sand	12.9	6.0	15.5	457.0	457.0
clayey silt with sand	12.9	4.0	17.9	464.0	464.0
clayey silt with sand	12.9	5.0	16.0	472.0	472.0
clayey silt with sand	12.9	6.0	17.5	479.0	479.0
clayey silt with sand	12.9	4.0	18.0	486.0	486.0
clayey silt with sand	12.9	4.0	16.5	502.0	502.0
weathered bedrock	9.0	0.0	22.3	686.0	1028.0

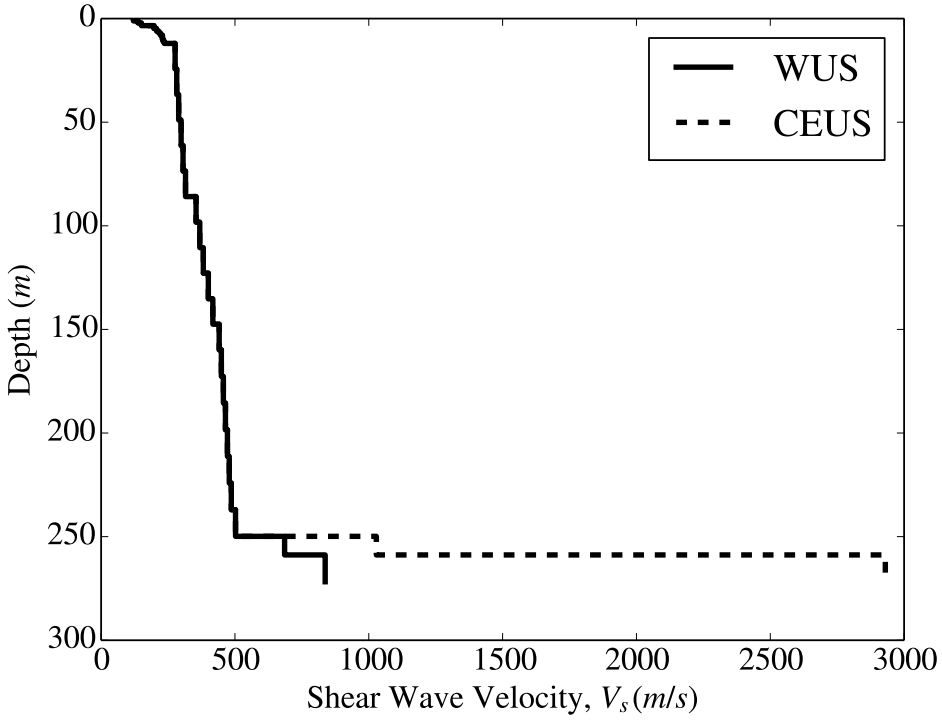


Figure E.41: Shear wave velocities of Profile040

E.42 Profile041 - Wildlife-Westmorland (1981)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silt	1.2	11.0	16.1	89.0	89.0
silt	1.2	13.0	15.5	90.0	90.0
silty sand	1.2	14.0	17.0	109.0	109.0
silty sand	1.2	19.0	18.0	114.0	114.0
silty sand	1.2	10.0	20.4	127.0	127.0
silty sand	1.2	12.0	18.7	140.0	140.0
silty clay	1.0	5.0	15.0	152.0	152.0
silty clay	1.0	4.0	14.7	161.0	161.0
silty clay	1.0	4.0	14.8	170.0	170.0
silty clay	1.0	7.0	14.9	180.0	180.0
silty clay	1.0	5.0	17.2	189.0	189.0
silty clay	1.0	6.0	16.9	201.0	201.0
silty clay	6.1	5.0	14.5	280.0	280.0
silt	13.4	17.0	16.6	274.0	274.0
silt	13.4	12.0	14.4	283.0	283.0
silt	13.4	12.0	15.9	293.0	293.0
silt	13.4	6.0	17.1	309.0	309.0
silt	13.4	15.0	16.5	320.0	320.0
silty clay	12.7	6.0	14.2	351.0	351.0
silty clay	12.7	4.0	14.3	373.0	373.0
silty clay	25.4	6.0	16.0	396.0	396.0
silty clay	12.7	6.0	16.5	427.0	427.0
silty clay	12.7	6.0	15.3	457.0	457.0
clayey silt with sand	25.7	6.0	18.0	457.0	457.0
clayey silt with sand	12.8	7.0	17.5	467.0	467.0
clayey silt with sand	12.8	7.0	18.1	475.0	475.0
clayey silt with sand	25.7	5.0	17.3	488.0	488.0
clayey silt with sand	12.8	6.0	17.0	503.0	503.0
weathered bedrock	9.2	0.0	22.0	685.0	1028.0

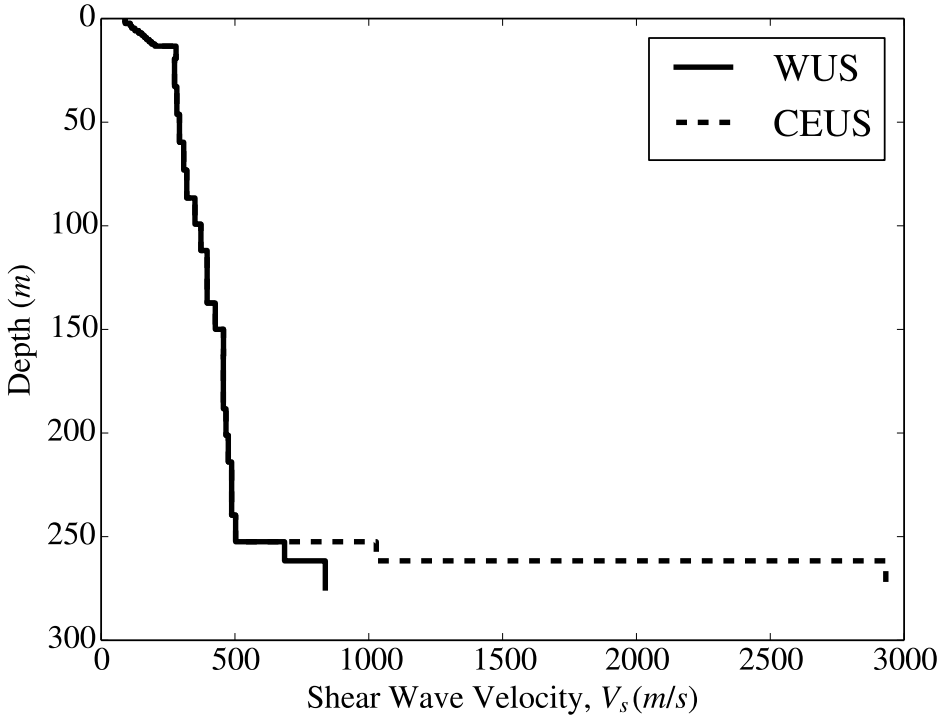


Figure E.42: Shear wave velocities of Profile041

E.43 Profile042 - Heber Road A1-Imperial Valley (1979)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sand	1.0	0.0	18.8	137.0	137.0
sand	1.0	0.0	17.4	167.0	167.0
sand	1.0	0.0	21.8	182.0	182.0
sand	1.0	0.0	20.4	205.0	205.0
sand	1.0	0.0	18.1	228.0	228.0
clayey silt	1.3	7.0	15.6	138.0	138.0
sand	1.5	0.0	18.8	200.0	200.0
clayey silt	0.8	6.0	17.2	228.0	228.0
sand	0.6	0.0	18.1	258.0	258.0
clayey silt	0.3	6.0	16.8	274.0	274.0
sand	1.1	0.0	20.6	304.0	304.0
silty clay	9.9	4.0	16.0	274.0	274.0
silty clay	9.9	7.0	16.8	281.0	281.0

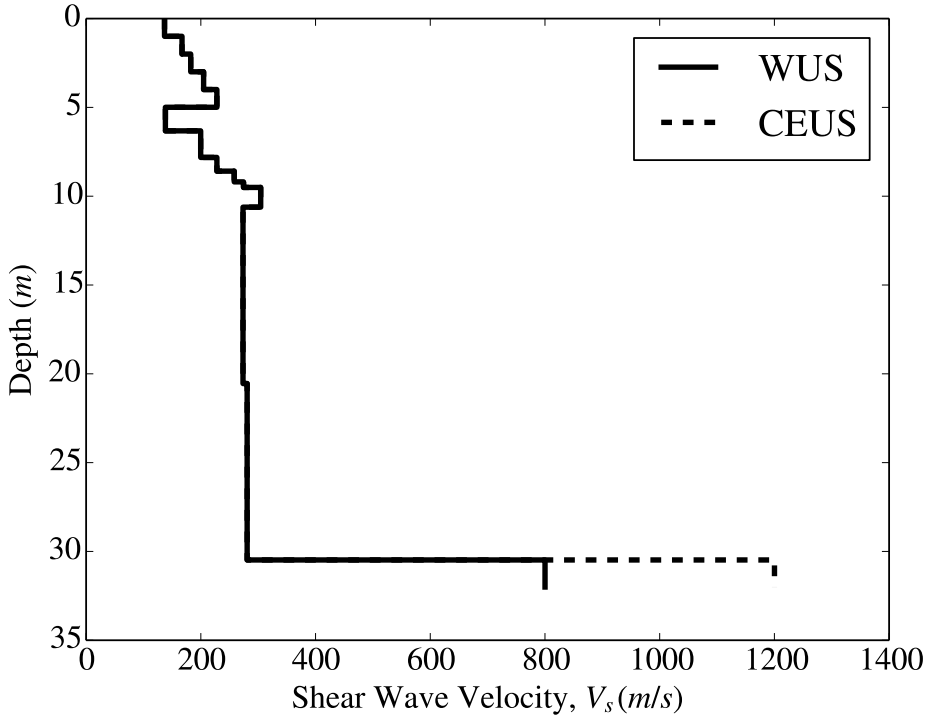


Figure E.43: Shear wave velocities of Profile042

E.44 Profile043 - Heber Road A2-Imperial Valley (1979)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sand	1.0	0.0	21.3	145.0	145.0
sand	1.0	0.0	20.2	168.0	168.0
sand	1.0	0.0	19.1	183.0	183.0
sand	1.0	0.0	21.6	205.0	205.0
sand	1.0	0.0	18.7	228.0	228.0
clayey silt	0.8	6.0	16.5	137.0	137.0
sand	1.2	0.0	18.2	200.0	200.0
clayey silt	1.2	7.0	16.0	239.0	239.0
sand	0.2	0.0	17.4	259.0	259.0
clayey silt	0.4	4.0	14.9	274.0	274.0
sand	0.7	0.0	22.0	305.0	305.0
clayey silt	1.3	6.0	14.6	273.0	273.0
clayey silt	1.3	7.0	14.9	320.0	320.0
sand	1.5	0.0	21.7	305.0	305.0
silty clay	8.5	7.0	14.6	274.0	274.0
silty clay	8.5	5.0	17.0	278.0	278.0

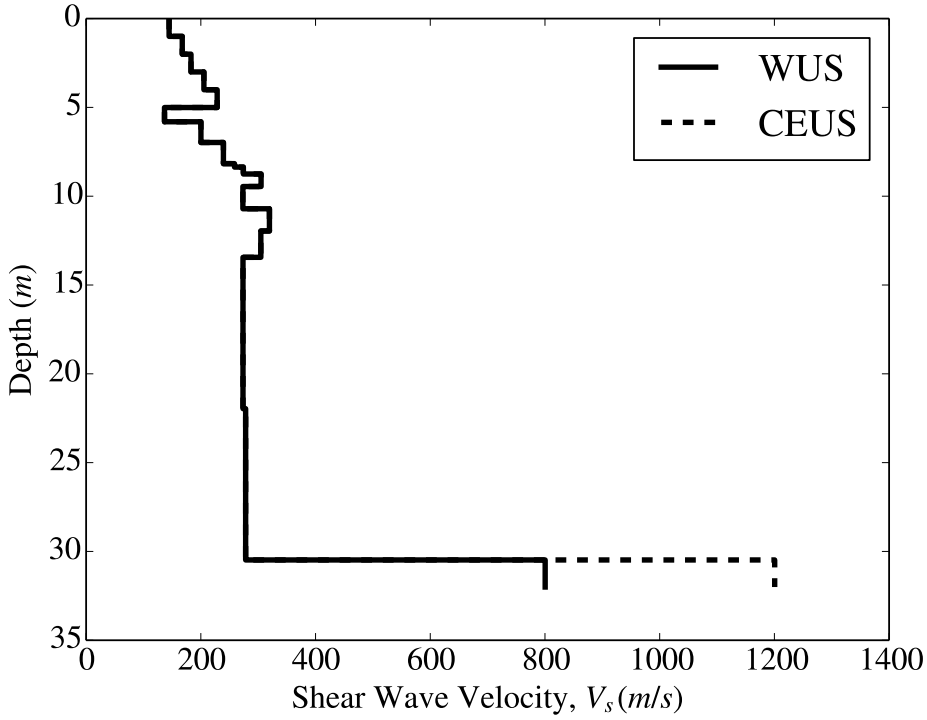


Figure E.44: Shear wave velocities of Profile043

E.45 Profile044 - Heber Road A3-Imperial Valley (1979)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
sand	1.0	0.0	19.1	137.0	137.0
sand	1.0	0.0	18.8	167.0	167.0
sand	1.0	0.0	19.9	182.0	182.0
sand	1.0	0.0	20.5	205.0	205.0
sand	1.0	0.0	19.4	228.0	228.0
clayey silt	1.5	4.0	17.1	239.0	239.0
sand	0.5	0.0	18.3	258.0	258.0
clayey silt	1.2	5.0	16.4	274.0	274.0
sand	1.3	0.0	20.9	305.0	305.0
clayey silt	1.3	7.0	17.0	274.0	274.0
clayey silt	1.3	7.0	14.5	320.0	320.0
sand	1.5	0.0	21.5	304.0	304.0
silty clay	8.5	5.0	15.8	273.0	273.0
silty clay	8.5	7.0	17.1	278.0	278.0

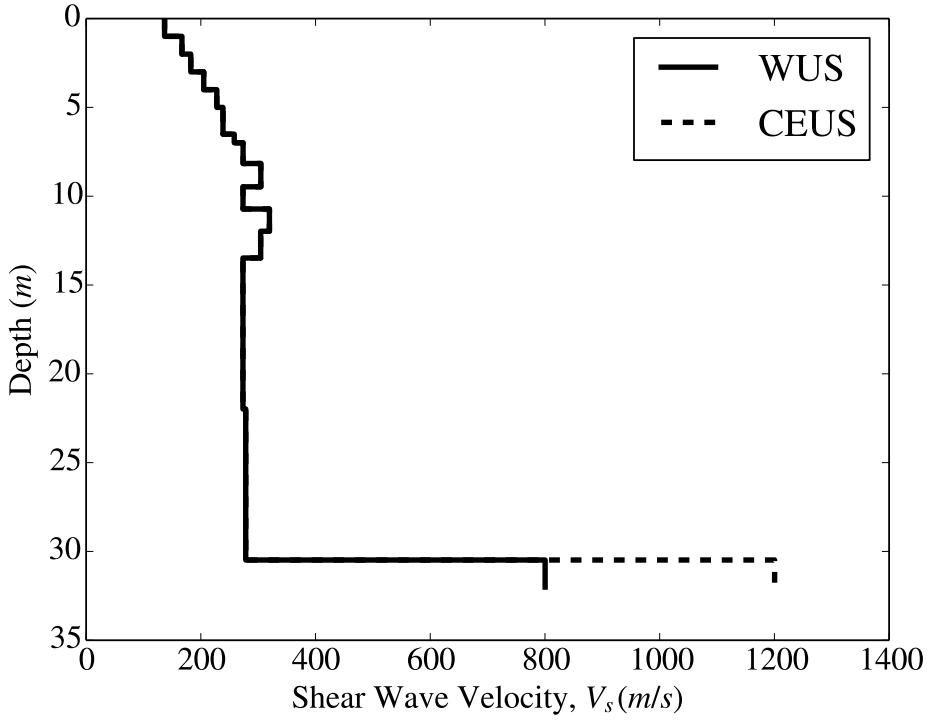


Figure E.45: Shear wave velocities of Profile044

E.46 Profile045 - Kornbloom-Imperial Valley (1979)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty clay	1.6	6.0	15.6	131.0	131.0
silt	1.1	18.0	16.0	124.0	124.0
silt	2.2	14.0	14.2	139.0	139.0
silt	1.1	5.0	17.2	152.0	152.0
silty clay	1.1	7.0	14.3	175.0	175.0
silty clay	1.1	5.0	15.0	198.0	198.0
silt	0.9	16.0	15.2	175.0	175.0
silt	0.9	18.0	14.5	225.0	225.0
sand	0.9	0.0	18.4	225.0	225.0
sand	0.9	0.0	18.4	250.0	250.0
silt	3.1	7.0	15.5	265.0	265.0
silt	3.1	8.0	16.1	308.0	308.0
silty clay	12.9	5.0	14.2	276.0	276.0
silty clay	12.9	5.0	17.0	285.0	285.0
silty clay	12.9	5.0	15.2	295.0	295.0
silty clay	12.9	6.0	15.2	304.0	304.0
silty clay	12.9	6.0	15.1	315.0	315.0
silty clay	12.9	7.0	15.6	350.0	350.0
silty clay	12.9	5.0	15.2	364.0	364.0
silty clay	12.9	5.0	15.8	382.0	382.0
silty clay	12.9	5.0	15.9	401.0	401.0
silty clay	12.9	6.0	15.4	418.0	418.0
silty clay	12.9	6.0	16.3	441.0	441.0
clayey silt with sand	12.9	5.0	17.7	449.0	449.0
clayey silt with sand	12.9	5.0	16.9	454.0	454.0
clayey silt with sand	12.9	7.0	15.9	460.0	460.0
clayey silt with sand	12.9	6.0	16.9	467.0	467.0
clayey silt with sand	12.9	5.0	17.1	476.0	476.0
clayey silt with sand	12.9	5.0	15.8	485.0	485.0
clayey silt with sand	12.9	6.0	16.9	497.0	497.0
weathered bedrock	9.0	0.0	23.2	685.0	1027.0

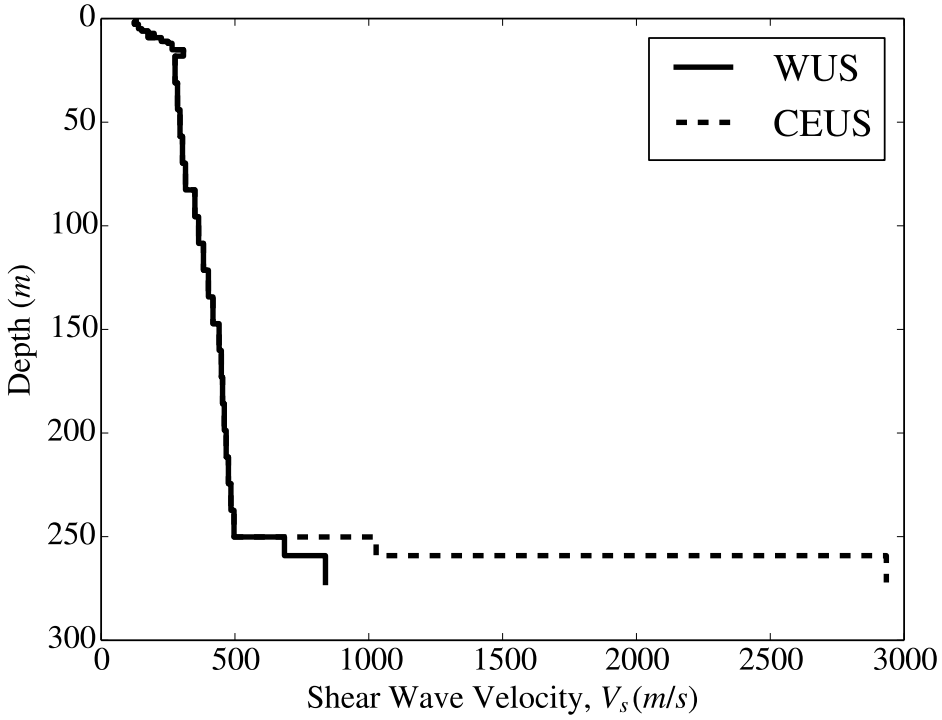


Figure E.46: Shear wave velocities of Profile045

E.47 Profile046 - Radio Tower B1-Imperial Valley (1979)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty clay	1.0	4.0	16.3	129.0	129.0
silty clay	1.0	4.0	15.2	142.0	142.0
silty clay	1.0	6.0	17.1	153.0	153.0
sand	1.0	0.0	20.9	125.0	125.0
sand	1.0	0.0	18.2	152.0	152.0
sand	1.0	0.0	17.5	175.0	175.0
silty clay	0.9	6.0	16.7	175.0	175.0
silty clay	1.9	4.0	16.9	192.0	192.0
silty clay	0.9	4.0	15.4	210.0	210.0
silty clay	1.9	4.0	16.1	225.0	225.0
silty clay	0.9	5.0	16.8	229.0	229.0
sand	0.9	0.0	21.3	198.0	198.0
silty clay	13.3	5.0	14.9	277.0	277.0
silty clay	13.3	5.0	15.0	285.0	285.0
silty clay	13.3	4.0	16.9	295.0	295.0
silty clay	13.3	7.0	14.8	302.0	302.0
silty clay	13.3	5.0	16.7	309.0	309.0
silty clay	13.3	5.0	16.9	335.0	335.0
silty clay	13.3	7.0	17.2	364.0	364.0
silty clay	13.3	4.0	16.2	382.0	382.0
silty clay	13.3	7.0	15.2	396.0	396.0
silty clay	13.3	5.0	16.0	411.0	411.0
silty clay	13.3	7.0	17.2	441.0	441.0
clayey silt with sand	12.8	4.0	16.9	449.0	449.0
clayey silt with sand	12.8	5.0	17.0	457.0	457.0
clayey silt with sand	12.8	5.0	16.3	464.0	464.0
clayey silt with sand	12.8	5.0	18.0	471.0	471.0
clayey silt with sand	12.8	7.0	17.6	478.0	478.0
clayey silt with sand	12.8	6.0	16.9	487.0	487.0
clayey silt with sand	12.8	6.0	15.8	503.0	503.0
weathered bedrock	9.2	0.0	22.2	685.0	1027.0

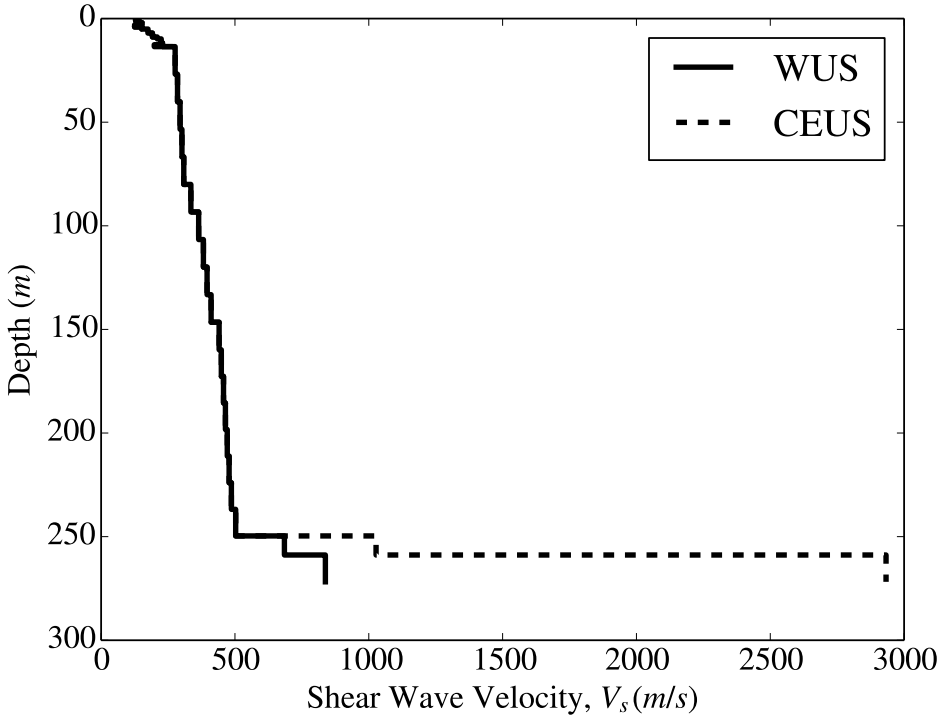


Figure E.47: Shear wave velocities of Profile046

E.48 Profile047 - Radio Tower B2-Imperial Valley (1979)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silty clay	0.9	6.0	16.7	132.0	132.0
silty clay	0.9	4.0	17.0	153.0	153.0
sand	1.1	0.0	18.6	150.0	150.0
silty clay	0.9	7.0	15.2	175.0	175.0
silty clay	1.8	5.0	14.5	181.0	181.0
silty clay	0.9	5.0	16.5	189.0	189.0
silty clay	1.8	6.0	14.9	197.0	197.0
silty clay	0.9	5.0	16.1	206.0	206.0
silty clay	0.9	4.0	15.0	214.0	214.0
silty clay	1.8	6.0	16.6	221.0	221.0
silty clay	0.9	6.0	17.1	228.0	228.0
sand	0.9	0.0	17.6	175.0	175.0
sand	0.9	0.0	18.0	225.0	225.0
silty clay	13.2	6.0	16.4	276.0	276.0
silty clay	13.2	5.0	14.7	285.0	285.0
silty clay	13.2	4.0	17.1	295.0	295.0
silty clay	13.2	7.0	16.9	304.0	304.0
silty clay	13.2	5.0	17.1	311.0	311.0
silty clay	13.2	4.0	14.6	335.0	335.0
silty clay	13.2	6.0	17.3	364.0	364.0
silty clay	13.2	7.0	14.2	382.0	382.0
silty clay	13.2	4.0	16.9	396.0	396.0
silty clay	13.2	7.0	14.3	411.0	411.0
silty clay	13.2	5.0	14.2	441.0	441.0
clayey silt with sand	12.8	7.0	16.1	449.0	449.0
clayey silt with sand	12.8	5.0	17.8	457.0	457.0
clayey silt with sand	12.8	6.0	16.3	465.0	465.0
clayey silt with sand	12.8	4.0	15.9	472.0	472.0
clayey silt with sand	12.8	4.0	16.5	478.0	478.0
clayey silt with sand	12.8	7.0	16.6	487.0	487.0
clayey silt with sand	12.8	5.0	15.1	503.0	503.0
weathered bedrock	9.0	0.0	23.6	686.0	1029.0

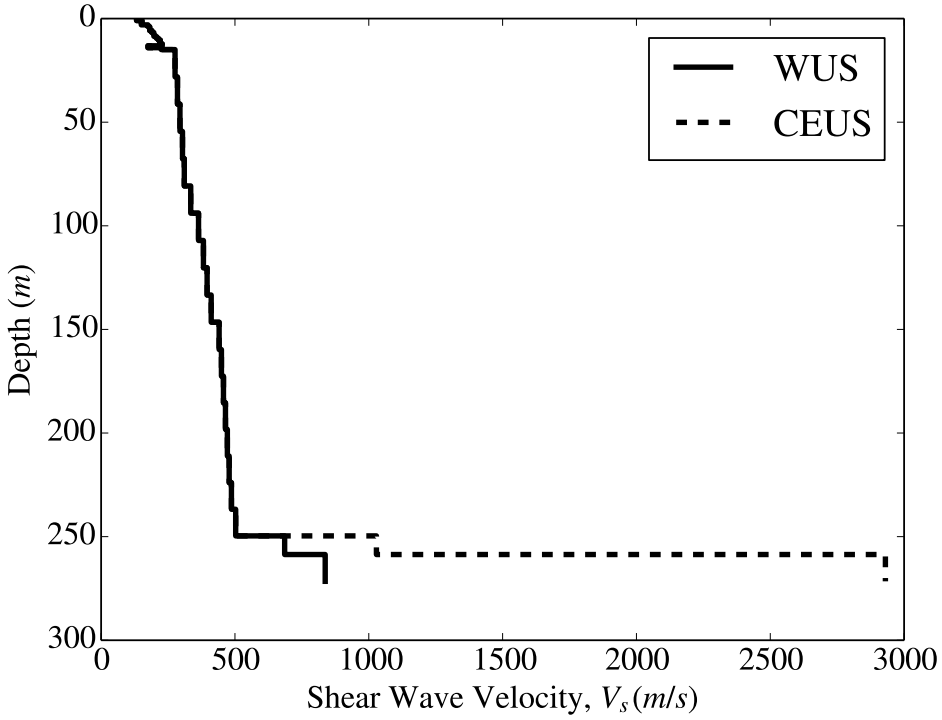


Figure E.48: Shear wave velocities of Profile047

E.49 Profile048 - River Park A & C-Imperial Valley (1979)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silt and sandy silt	1.0	7.0	18.2	126.0	126.0
silt and sandy silt	1.0	6.0	16.6	140.0	140.0
silty clay	1.6	5.0	17.0	150.0	150.0
sand	1.3	0.0	21.0	201.0	201.0
sand	1.3	0.0	21.3	211.0	211.0
sand	1.9	0.0	17.5	217.0	217.0
sand	1.9	0.0	17.3	227.0	227.0
sand	1.0	0.0	20.6	233.0	233.0
sand	1.0	0.0	17.8	237.0	237.0
silty clay	12.3	4.0	15.3	277.0	277.0
silty clay	12.3	6.0	14.2	283.0	283.0
silty clay	12.3	6.0	15.1	291.0	291.0
silty clay	12.3	7.0	14.3	300.0	300.0
silty clay	12.3	4.0	14.7	307.0	307.0
silty clay	12.3	5.0	14.4	315.0	315.0
silty clay	12.3	5.0	17.2	355.0	355.0
silty clay	12.3	4.0	14.6	369.0	369.0
silty clay	12.3	6.0	14.5	382.0	382.0
silty clay	12.3	6.0	15.4	401.0	401.0
silty clay	12.3	6.0	16.2	418.0	418.0
silty clay	12.3	6.0	15.1	441.0	441.0
clayey silt with sand	12.9	5.0	17.7	450.0	450.0
clayey silt with sand	12.9	4.0	16.3	457.0	457.0
clayey silt with sand	12.9	7.0	16.4	465.0	465.0
clayey silt with sand	12.9	4.0	15.2	472.0	472.0
clayey silt with sand	12.9	5.0	15.7	479.0	479.0
clayey silt with sand	12.9	7.0	15.1	487.0	487.0
clayey silt with sand	12.9	4.0	15.4	503.0	503.0
weathered bedrock	9.1	0.0	22.5	686.0	1028.0

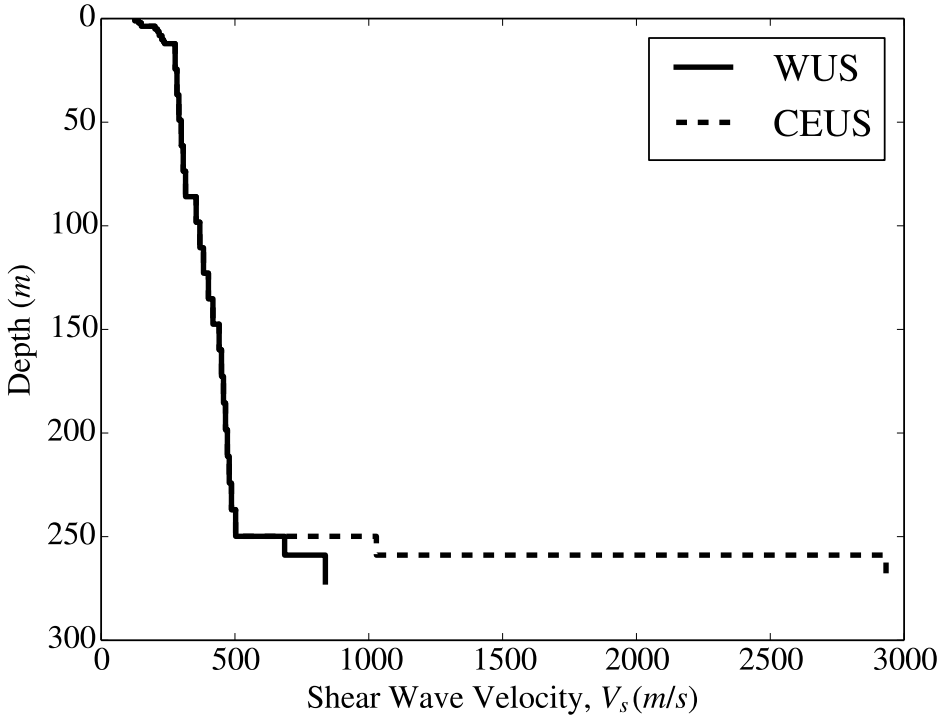


Figure E.49: Shear wave velocities of Profile048

E.50 Profile049 - Wildlife Site-Imperial Valley (1979)

Layer Name	Thickness	PI	Unit Weight	$V_{s,WUS}$	$V_{s,CEUS}$
	(m)		(kN/m^3)	(m/s)	(m/s)
silt	1.2	8.0	15.3	90.0	90.0
silt	1.2	16.0	17.0	98.0	98.0
silty sand	1.2	20.0	20.3	107.0	107.0
silty sand	1.2	7.0	20.2	113.0	113.0
silty sand	1.2	5.0	19.7	127.0	127.0
silty sand	1.2	8.0	18.2	140.0	140.0
silty clay	1.0	7.0	14.6	151.0	151.0
silty clay	1.0	7.0	14.7	162.0	162.0
silty clay	1.0	5.0	15.6	169.0	169.0
silty clay	1.0	5.0	15.9	179.0	179.0
silty clay	1.0	5.0	15.9	189.0	189.0
silty clay	1.0	5.0	15.7	201.0	201.0
silt	6.1	19.0	15.0	280.0	280.0
silty clay	13.4	7.0	15.7	274.0	274.0
silty clay	13.4	4.0	16.7	284.0	284.0
silty clay	13.4	6.0	15.2	293.0	293.0
silty clay	13.4	5.0	15.3	308.0	308.0
silty clay	13.4	5.0	16.4	320.0	320.0
silty clay	12.7	5.0	16.4	351.0	351.0
silty clay	12.7	6.0	17.0	373.0	373.0
silty clay	25.4	7.0	15.0	396.0	396.0
silty clay	12.7	5.0	14.6	427.0	427.0
silty clay	12.7	7.0	15.8	457.0	457.0
clayey silt with sand	25.7	5.0	15.5	457.0	457.0
clayey silt with sand	12.8	7.0	15.9	466.0	466.0
clayey silt with sand	12.8	4.0	16.9	475.0	475.0
clayey silt with sand	25.7	7.0	16.4	488.0	488.0
clayey silt with sand	12.8	5.0	15.4	503.0	503.0
weathered bedrock	9.2	0.0	23.5	686.0	1028.0

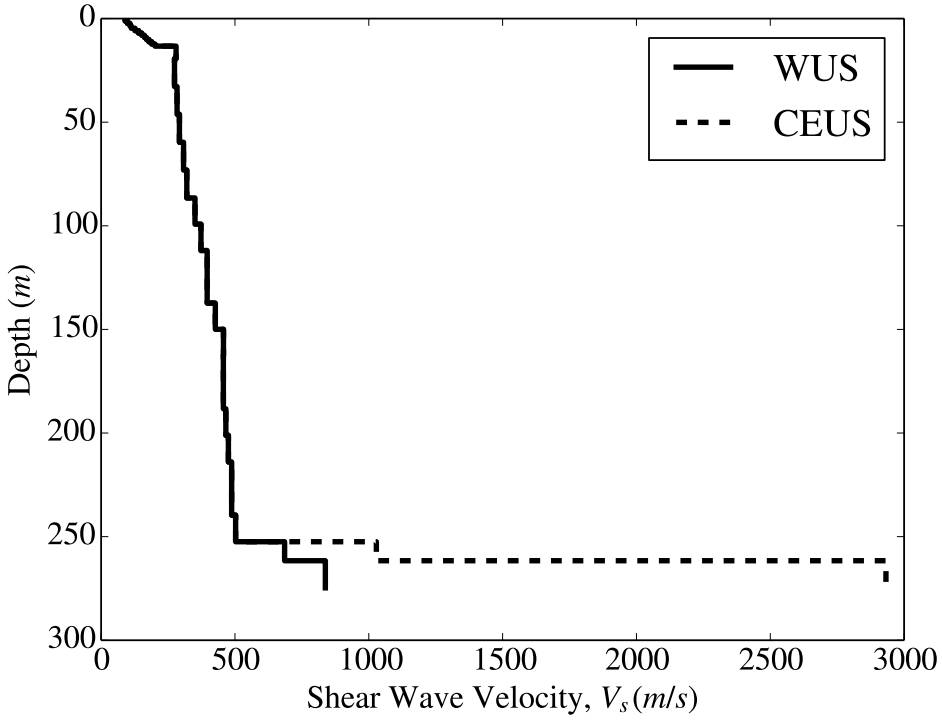


Figure E.50: Shear wave velocities of Profile049

E.51 Profile Summary Table

	Location	Earthquake	Total Depth (m)	Category
Profile000	Balboa Boulevard	Northridge Earthquake (1994)	87.1	A
Profile001	Malden Street	Northridge Earthquake (1994)	87.4	A
Profile002	Wynne Avenue	Northridge Earthquake (1994)	87.1	A
Profile003	MBARI NO:3 EB-1	Loma Prieta Earthquake (1989)	174.6	B
Profile004	MBARI NO:3 EB-5	Loma Prieta Earthquake (1989)	174.6	B
Profile005	MBARI NO:3 UC-B-10	Loma Prieta Earthquake (1989)	174.5	B
Profile006	Treasure Island	Loma Prieta Earthquake (1989)	102.2	C
Profile007	SFOBB-1	Loma Prieta Earthquake (1989)	155.1	D
Profile008	SFOBB-2	Loma Prieta Earthquake (1989)	152.8	D
Profile009	POO7-2	Loma Prieta Earthquake (1989)	129.3	D
Profile010	POO7-3	Loma Prieta Earthquake (1989)	165.1	D
Profile011	Woodward Marine UC-B4	Loma Prieta Earthquake (1989)	170.7	B
Profile012	Moss Landing UC-B1	Loma Prieta Earthquake (1989)	174.6	B
Profile013	Moss Landing UC-B2	Loma Prieta Earthquake (1989)	174.6	B
Profile014	Marine Lab. B1	Loma Prieta Earthquake (1989)	173.7	B
Profile015	Marine Lab. B2	Loma Prieta Earthquake (1989)	173.7	B
Profile016	Miller Farm CMF 3	Loma Prieta Earthquake (1989)	174.6	B
Profile017	Miller Farm CMF 5	Loma Prieta Earthquake (1989)	174.6	B
Profile018	Miller Farm CMF 8	Loma Prieta Earthquake (1989)	174.5	B
Profile019	Miller Farm CMF 10	Loma Prieta Earthquake (1989)	174.5	B
Profile020	Richmond City Hall	Loma Prieta Earthquake (1989)	25.1	E
Profile021	Port of Richmond POR-2	Loma Prieta Earthquake (1989)	25.1	E
Profile022	Port of Richmond POR-3	Loma Prieta Earthquake (1989)	25.2	E
Profile023	Port of Richmond POR-4	Loma Prieta Earthquake (1989)	25.2	E
Profile024	Miller Farm	Loma Prieta Earthquake (1989)	174.5	B
Profile025	Farris Farm	Loma Prieta Earthquake (1989)	174.6	B
Profile026	Wildlife Site	Elmore Ranch (1987)	7.7	F
Profile027	Wildlife Site	Superstition Hills (1987)	7.6	F
Profile028	Radio Tower B1	Superstition Hills (1987)	258.7	G
Profile029	Radio Tower B2	Superstition Hills (1987)	258.7	G
Profile030	McKim Ranch A	Superstition Hills (1987)	254.5	G
Profile031	Kornbloom	Superstition Hills (1987)	259.2	G
Profile032	River Park A & C	Superstition Hills (1987)	258.8	G
Profile033	Heber Road A1	Superstition Hills (1987)	268.5	G
Profile034	Heber Road A2	Superstition Hills (1987)	254.6	G
Profile035	Heber Road A3	Superstition Hills (1987)	268.3	G
Profile036	Kornbloom	Westmorland (1981)	259.4	G
Profile037	McKim Ranch	Westmorland (1981)	254.4	G
Profile038	Radio Tower B1	Westmorland (1981)	258.8	G
Profile039	Radio Tower B2	Westmorland (1981)	258.7	G
Profile040	River Park A & C	Westmorland (1981)	258.8	G

	Location	Earthquake	Total Depth (m)	Category
Profile041	Wildlife	Westmorland (1981)	261.7	G
Profile042	Heber Road A1	Imperial Valley (1979)	30.5	H
Profile043	Heber Road A2	Imperial Valley (1979)	30.5	H
Profile044	Heber Road A3	Imperial Valley (1979)	30.5	H
Profile045	Kornbloom	Imperial Valley (1979)	259.2	G
Profile046	Radio Tower B1	Imperial Valley (1979)	258.9	G
Profile047	Radio Tower B2	Imperial Valley (1979)	258.6	G
Profile048	River Park A & C	Imperial Valley (1979)	258.9	G
Profile049	Wildlife Site	Imperial Valley (1979)	261.6	G

Appendix F

Earthquake Motions

F.1 NGA Earthquake Database

The following figures show the WUS earthquake motions used in the equivalent-linear site response analyses. More details on each motion are given in the electronic supplements of Chapters 4 and 5.

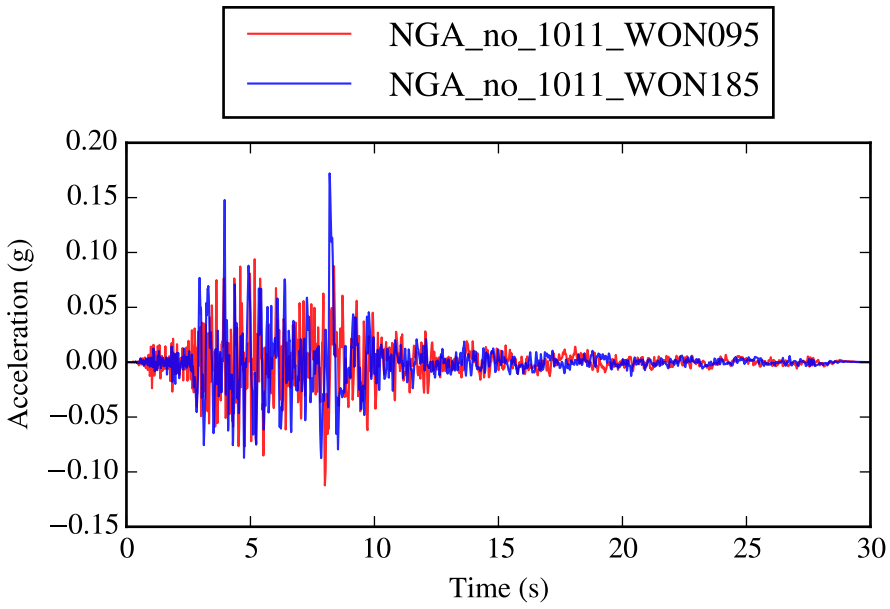


Figure F.1: WUS 1: *1011_WON095* and *1011_WON185*.

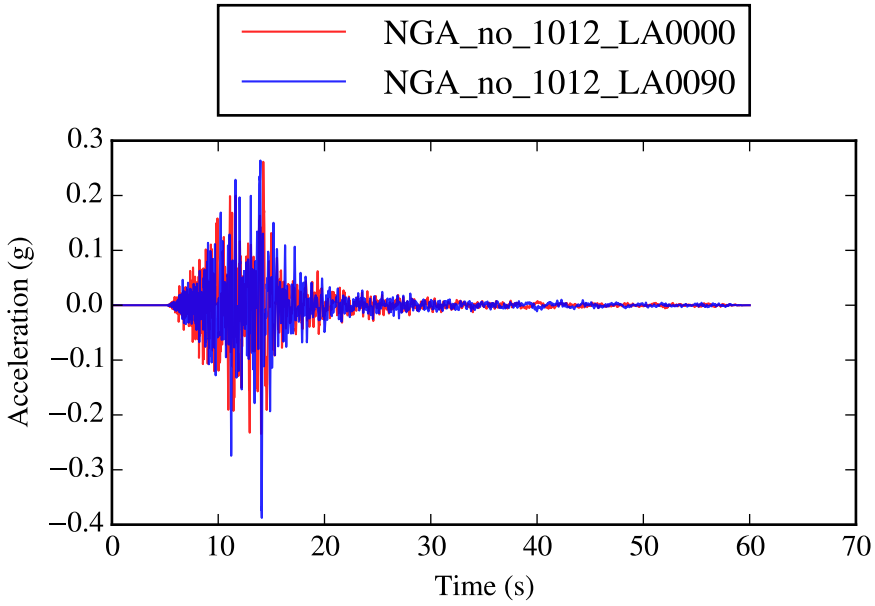


Figure F.2: WUS 2: *1012_LA0000* and *1012_LA0090*.

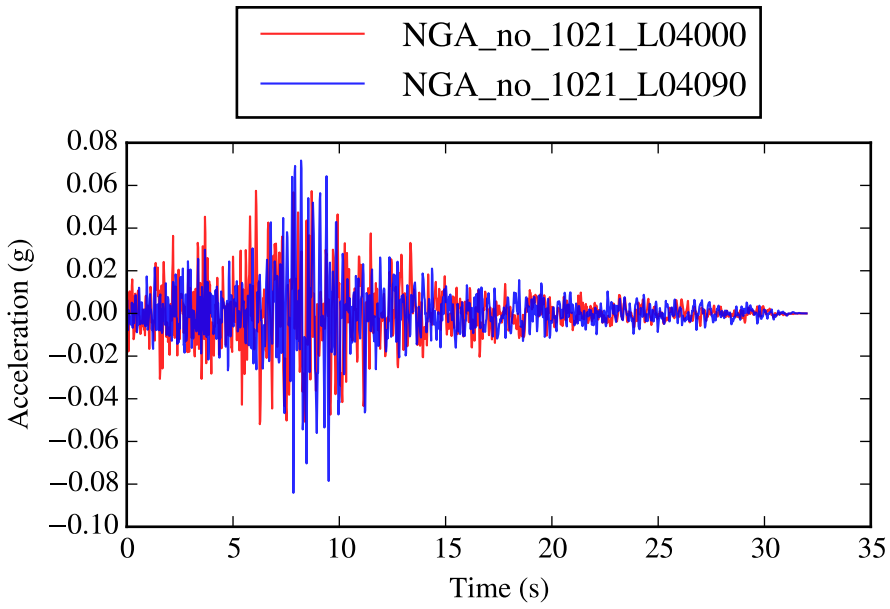


Figure F.3: WUS 3: *1021_L04000* and *1021_L04090*.

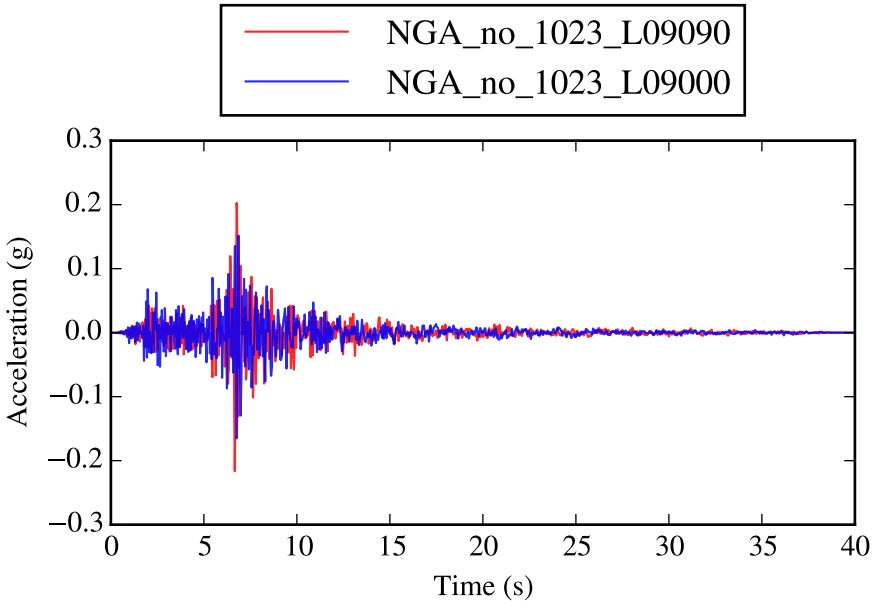


Figure F.4: WUS 4: *1023_L09090* and *1023_L09000*.

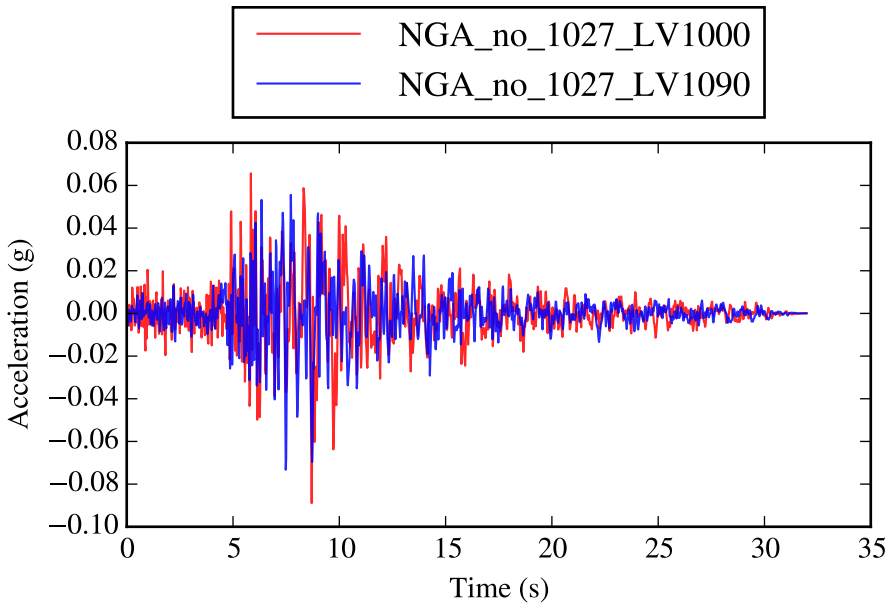


Figure F.5: WUS 5: *1027_LV1000* and *1027_LV1090*.

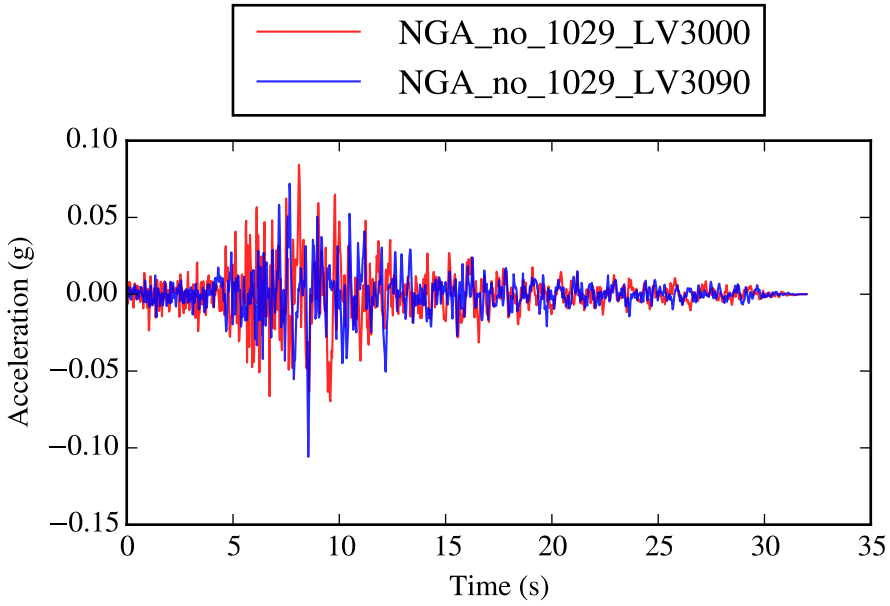


Figure F.6: WUS 6: *1029_LV3000* and *1029_LV3090*.

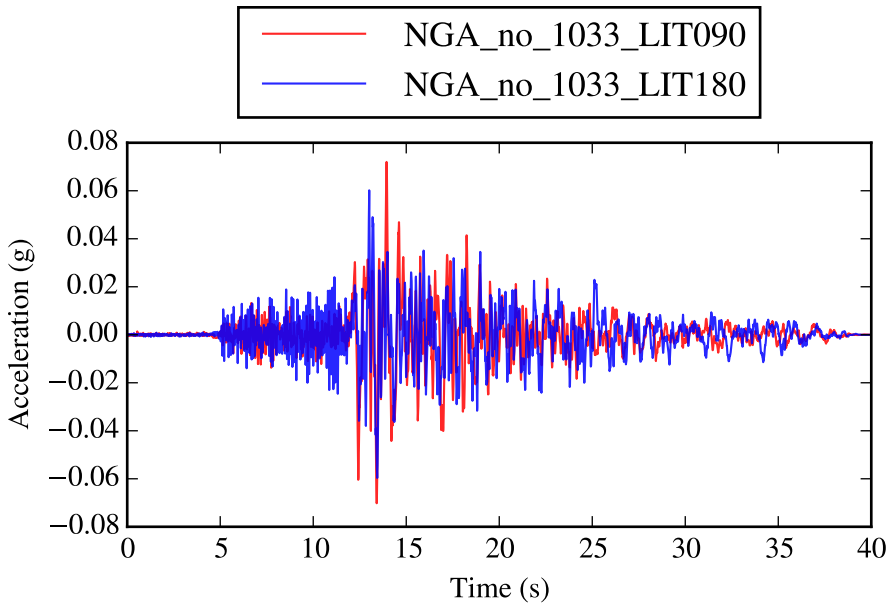


Figure F.7: WUS 7: *1033_LIT090* and *1033_LIT180*.

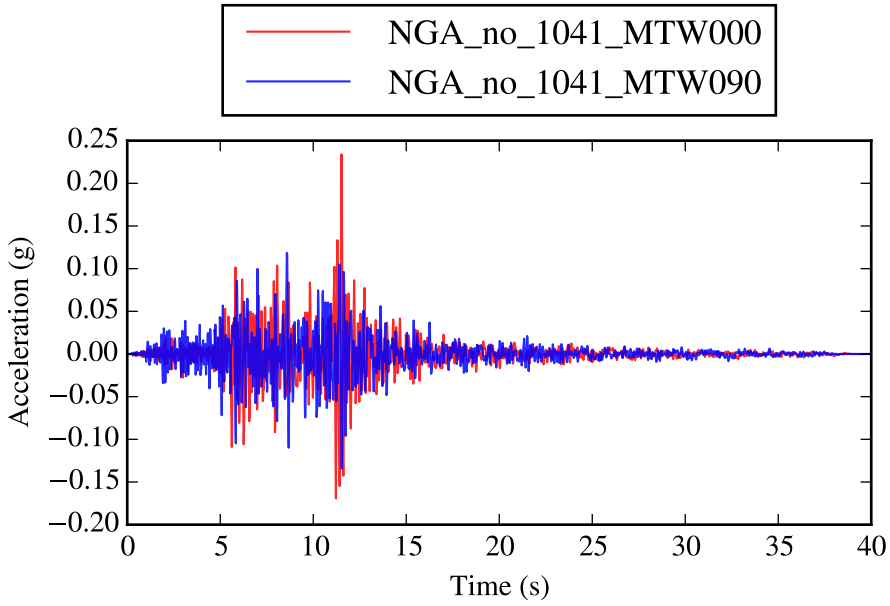


Figure F.8: WUS 8: *1041_MTW000* and *1041_MTW090*.

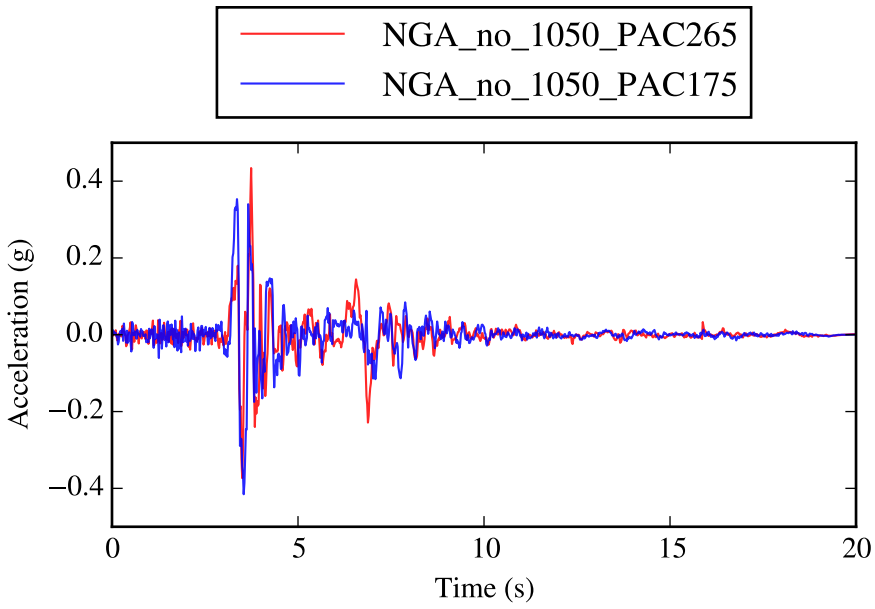


Figure F.9: WUS 9: *1050_PAC265* and *1050_PAC175*.

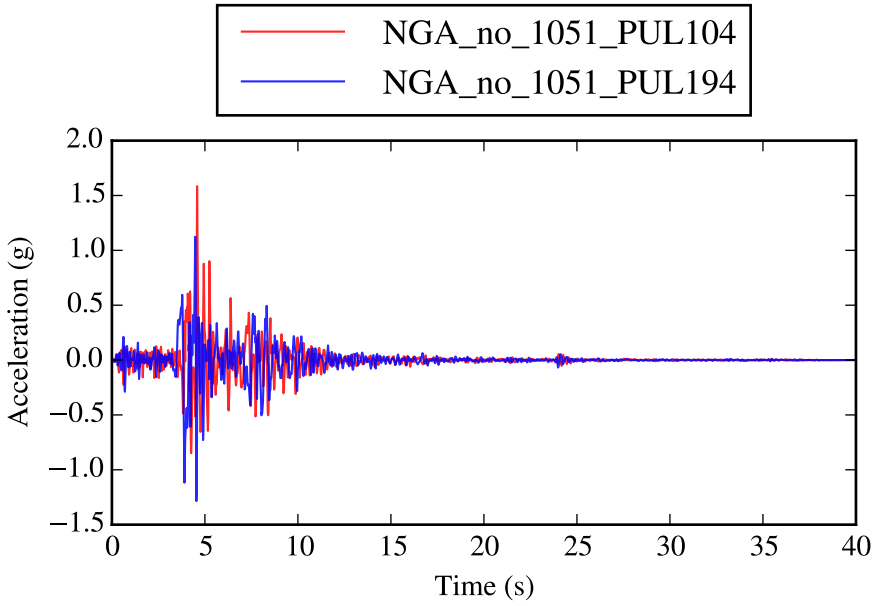


Figure F.10: WUS 10: *1051_PUL104* and *1051_PUL194*.

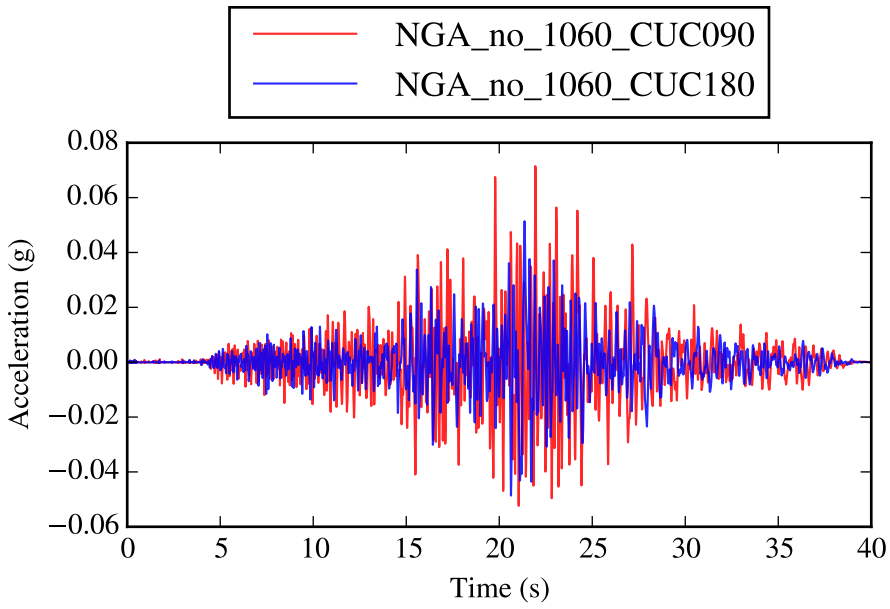


Figure F.11: WUS 11: *1060_CUC090* and *1060_CUC180*.

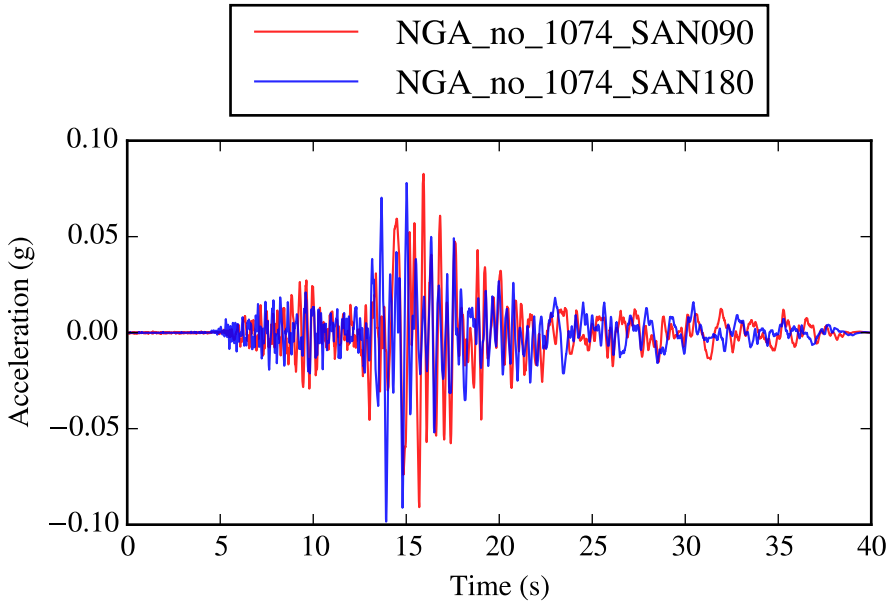


Figure F.12: WUS 12: *1074_SAN090* and *1074_SAN180*.

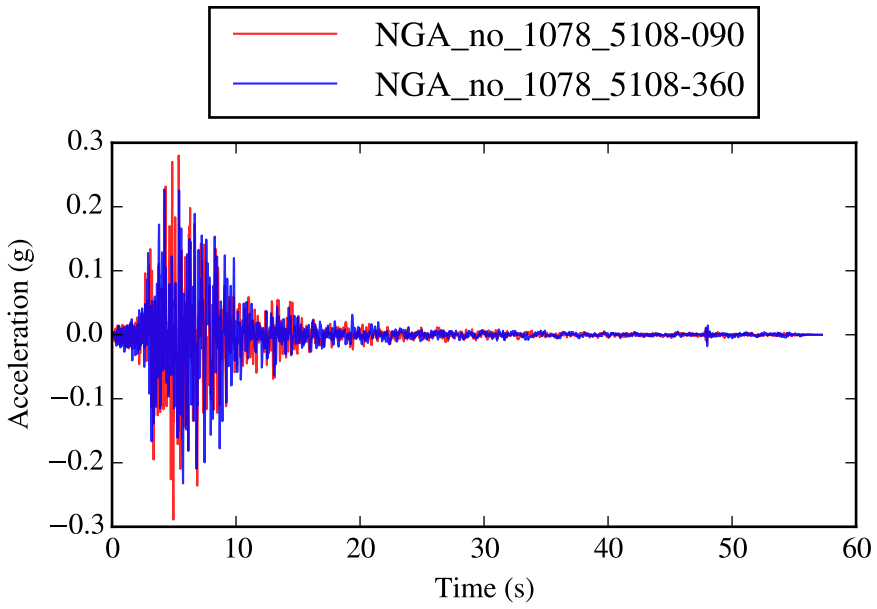


Figure F.13: WUS 13: *1078_5108-090* and *1078_5108-360*.

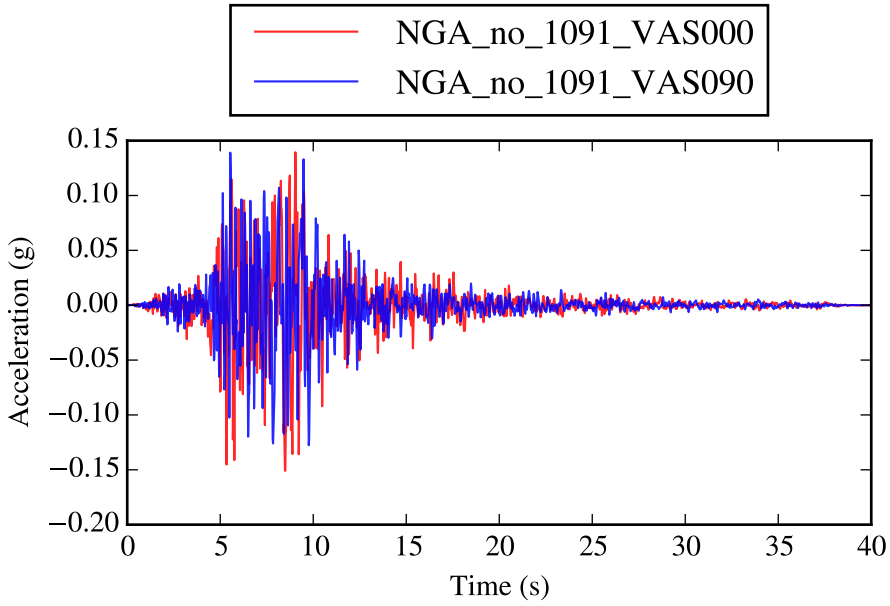


Figure F.14: WUS 14: *1091_VAS000* and *1091_VAS090*.

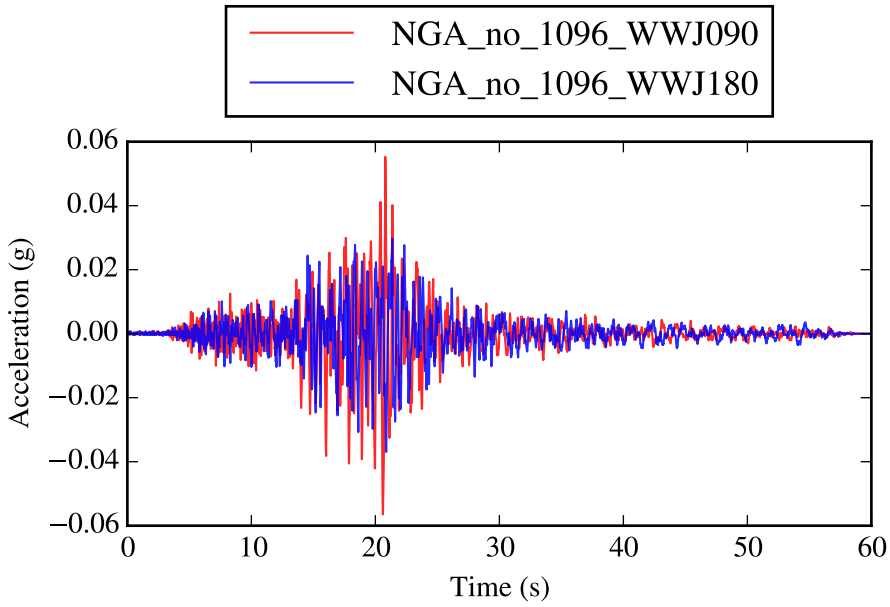


Figure F.15: WUS 15: *1096_WWJ090* and *1096_WWJ180*.

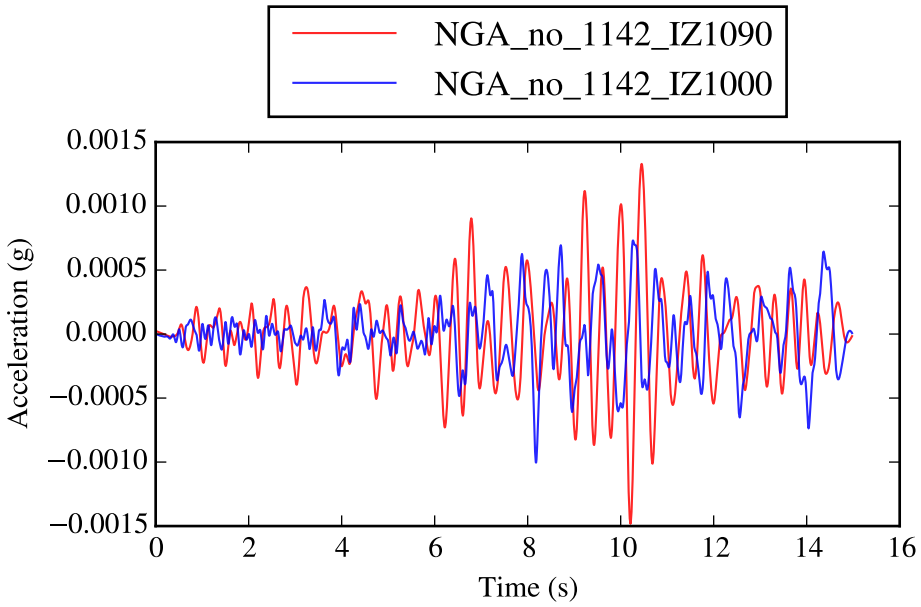


Figure F.16: WUS 16: *1142_IZ1090* and *1142_IZ1000*.

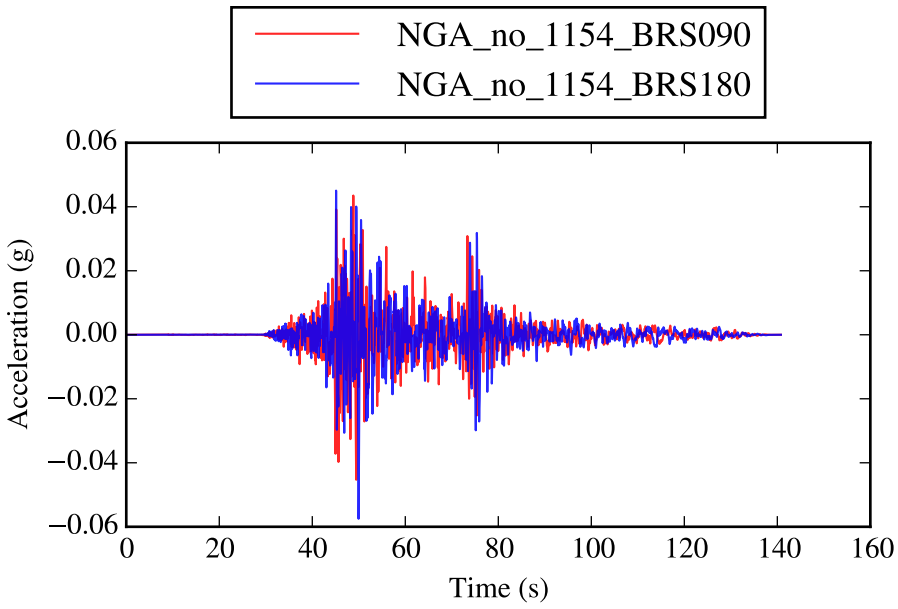


Figure F.17: WUS 17: *1154_BRS090* and *1154_BRS180*.

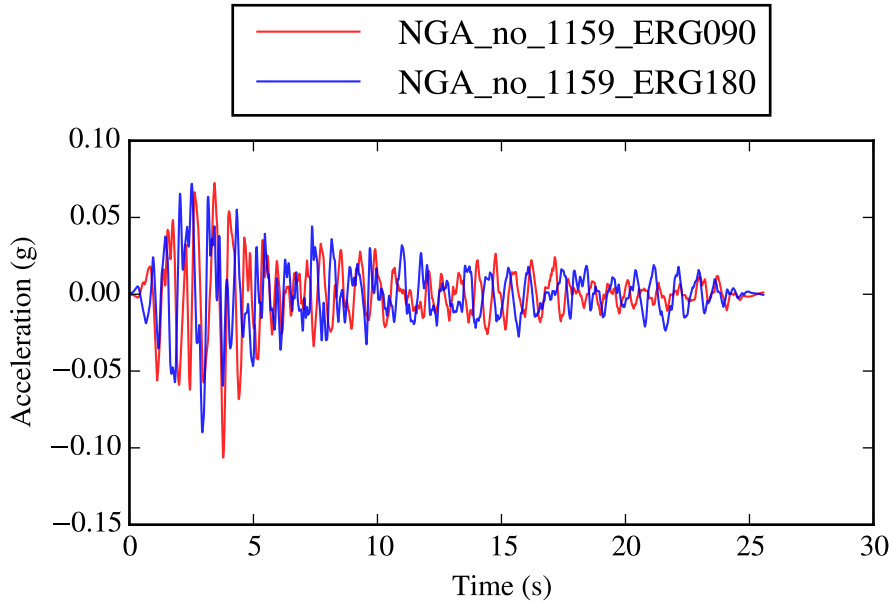


Figure F.18: WUS 18: *1159_ERG090* and *1159_ERG180*.

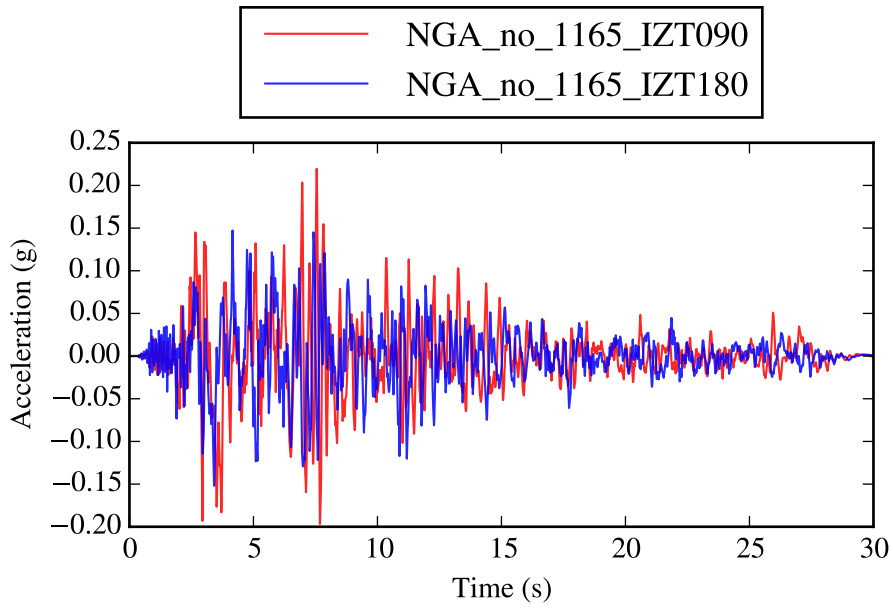


Figure F.19: WUS 19: *1165_IZT090* and *1165_IZT180*.

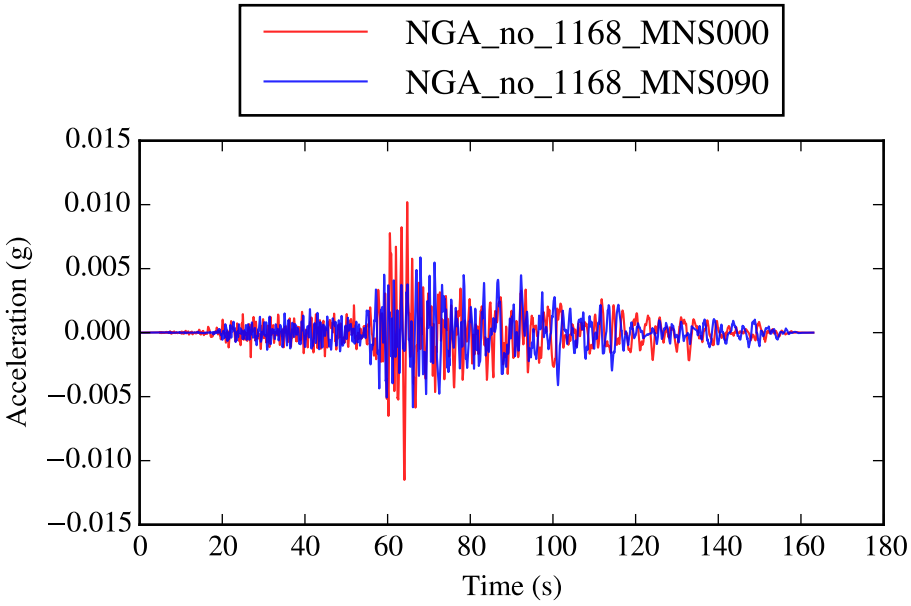


Figure F.20: WUS 20: *1168_MNS000* and *1168_MNS090*.

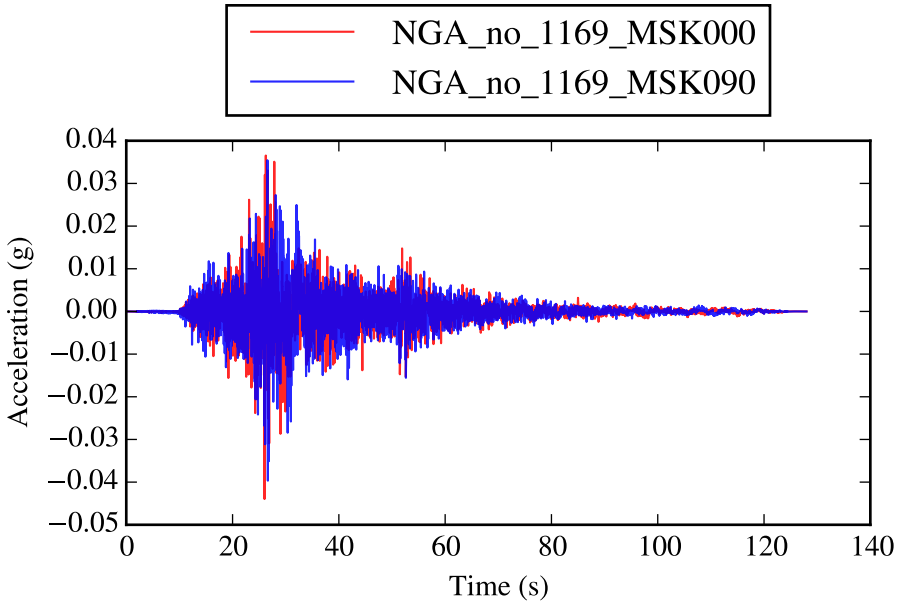


Figure F.21: WUS 21: *1169_MSK000* and *1169_MSK090*.

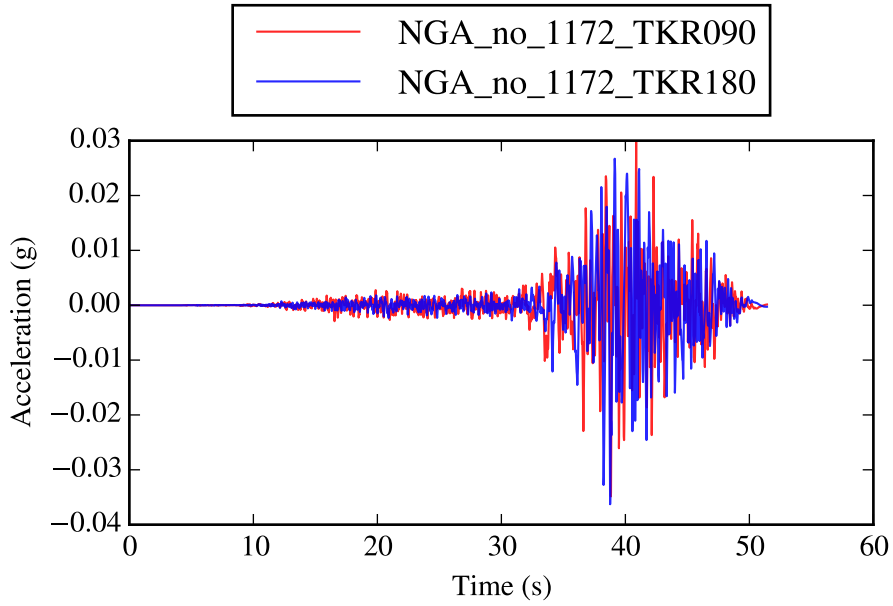


Figure F.22: WUS 22: *1172_TKR090* and *1172_TKR180*.

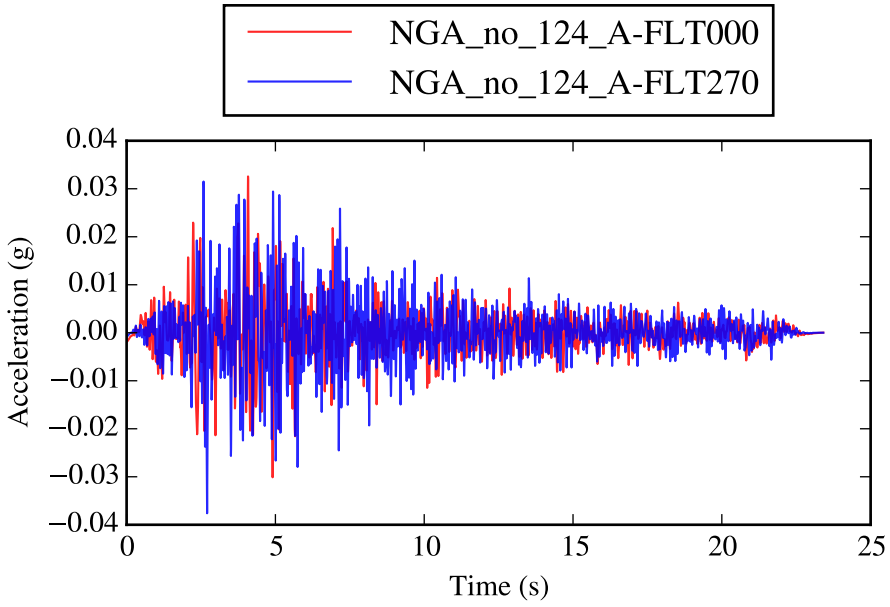


Figure F.23: WUS 23: *124_A-FLT000* and *124_A-FLT270*.

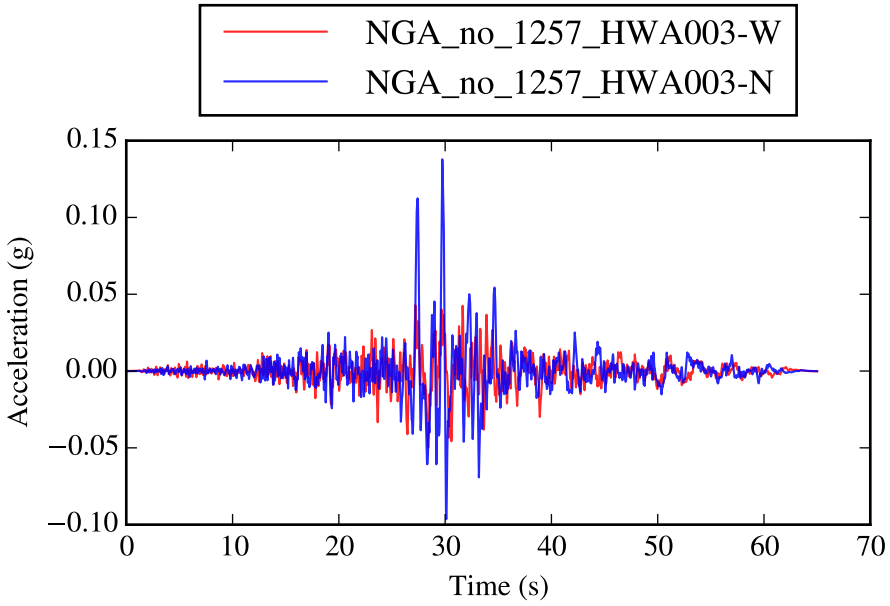


Figure F.24: WUS 24: *1257_HWA003-W* and *1257_HWA003-N*.

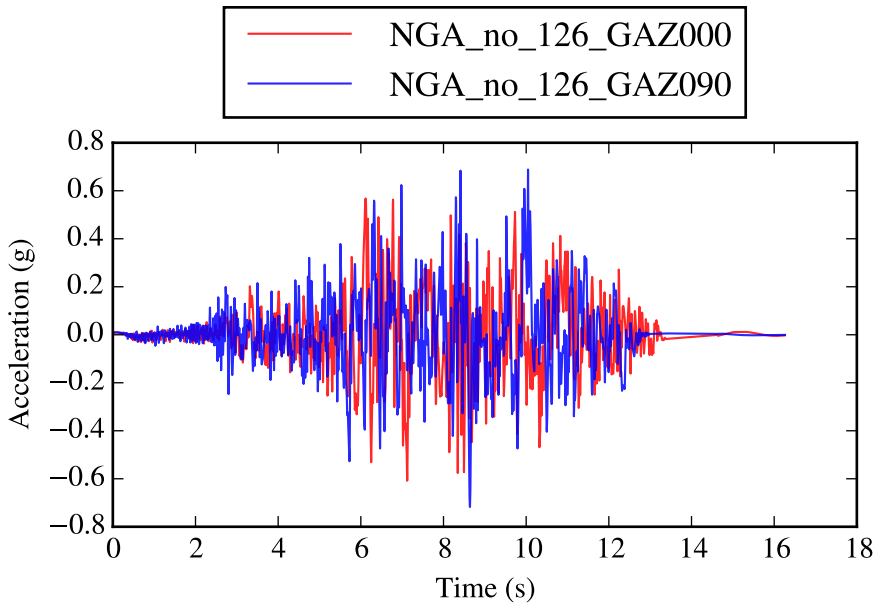


Figure F.25: WUS 25: *126_GAZ000* and *126_GAZ090*.

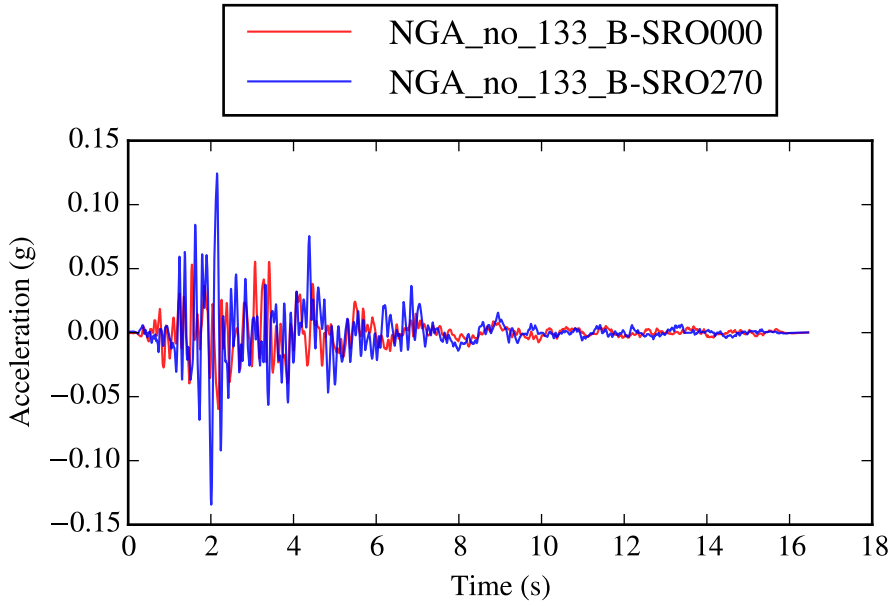


Figure F.26: WUS 26: *133_B-SRO000* and *133_B-SRO270*.

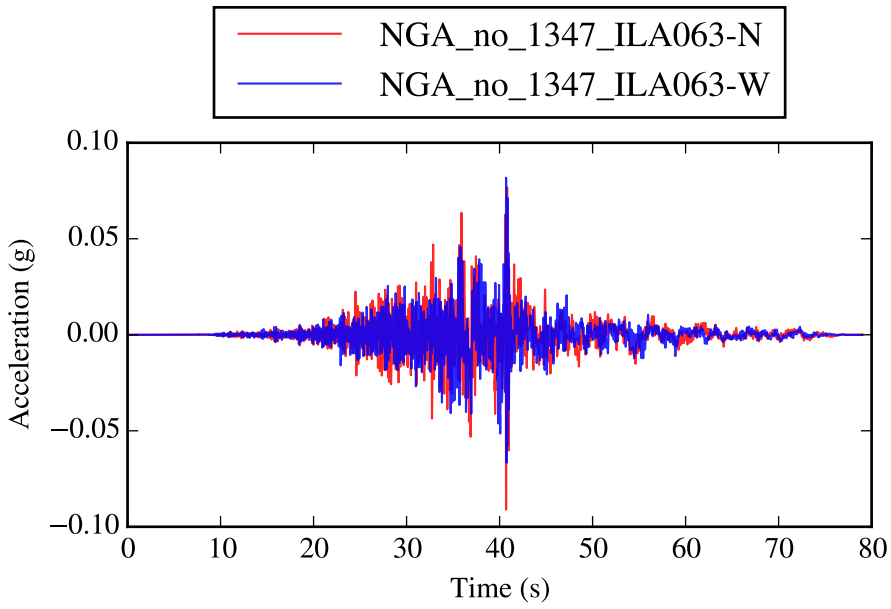


Figure F.27: WUS 27: *1347_ILA063-N* and *1347_ILA063-W*.

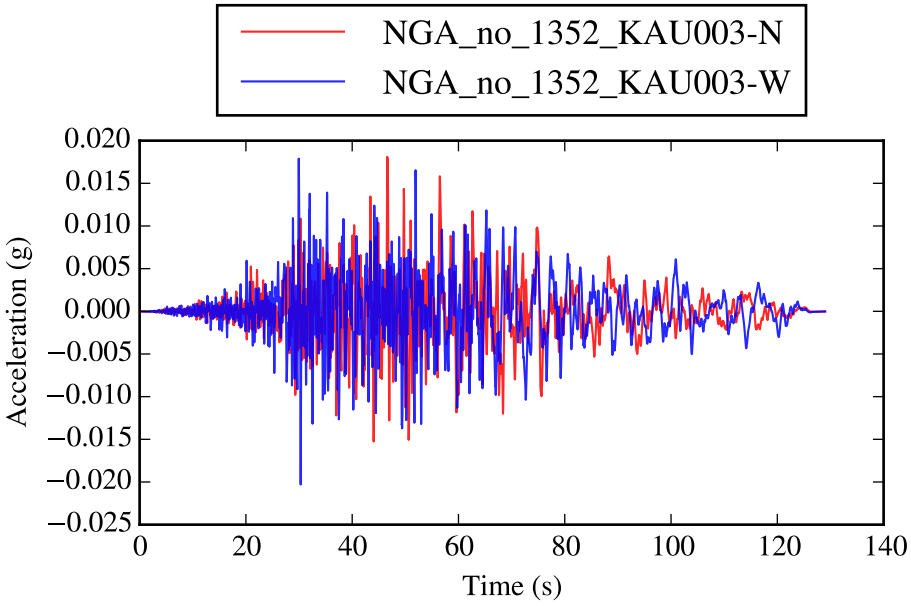


Figure F.28: WUS 28: *1352_KAU003-N* and *1352_KAU003-W*.

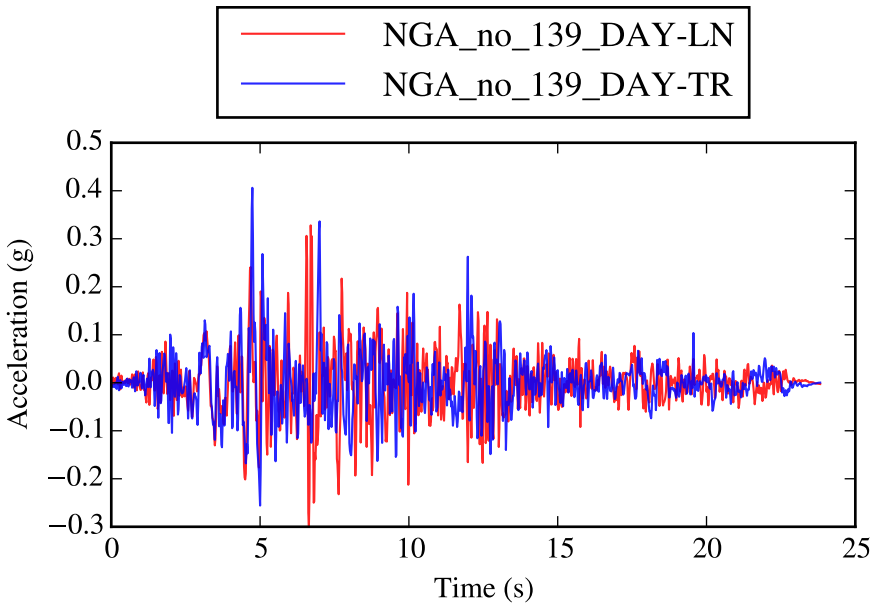


Figure F.29: WUS 29: *139_DAY-LN* and *139_DAY-TR*.

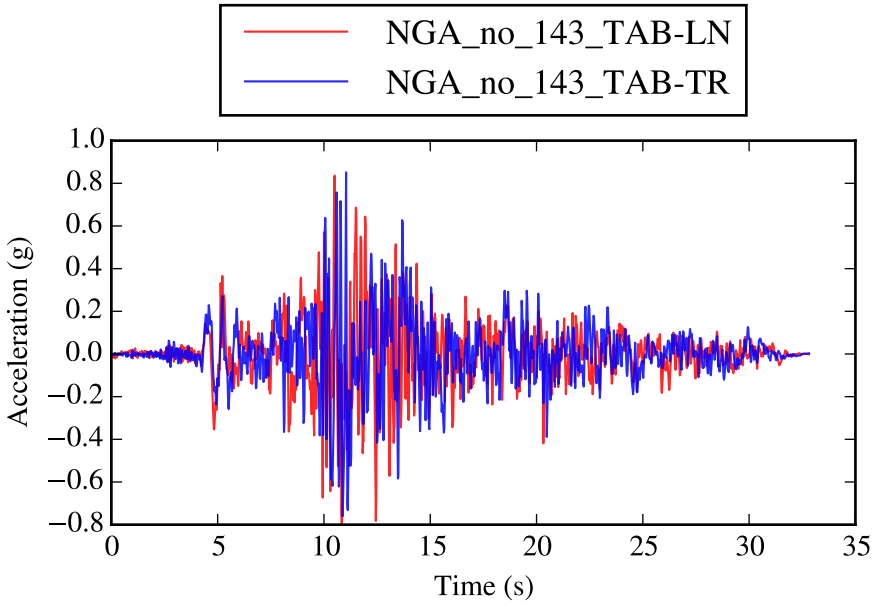


Figure F.30: WUS 30: *143_TAB-LN* and *143_TAB-TR*.

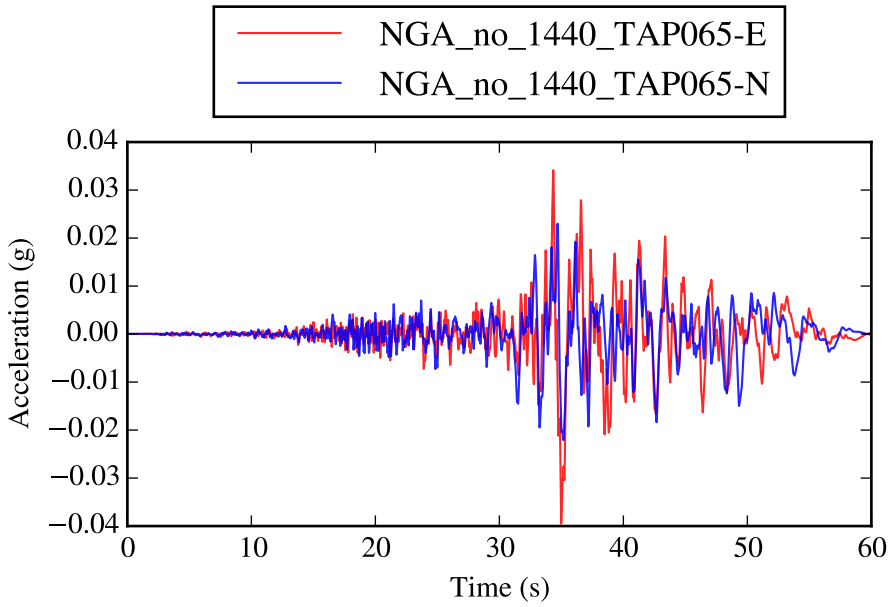


Figure F.31: WUS 31: *1440_TAP065-E* and *1440_TAP065-N*.

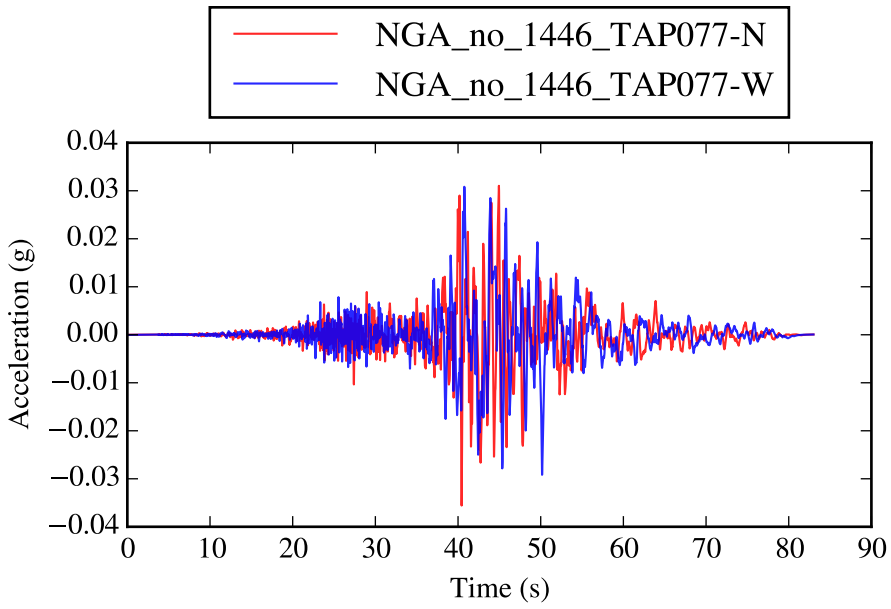


Figure F.32: WUS 32: *1446_TAP077-N* and *1446_TAP077-W*.

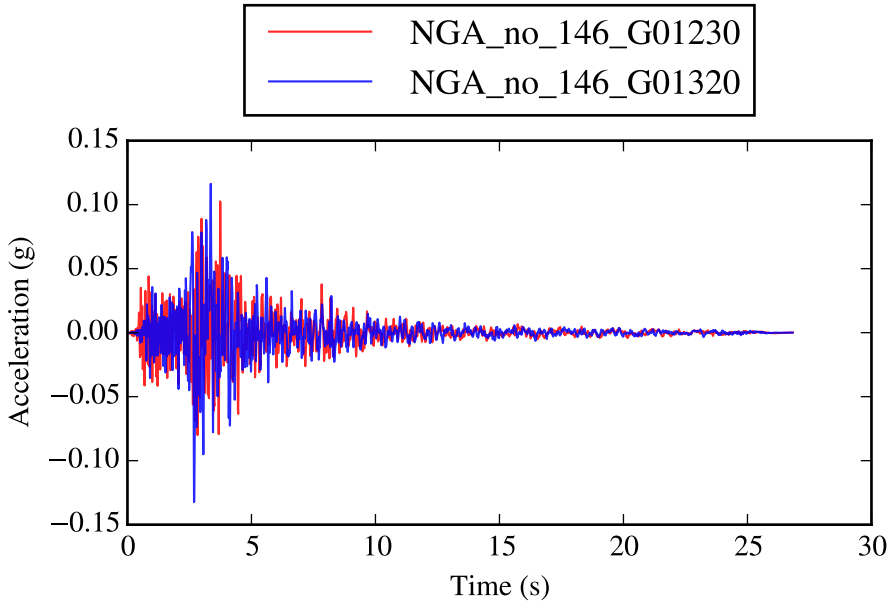


Figure F.33: WUS 33: *146_G01230* and *146_G01320*.

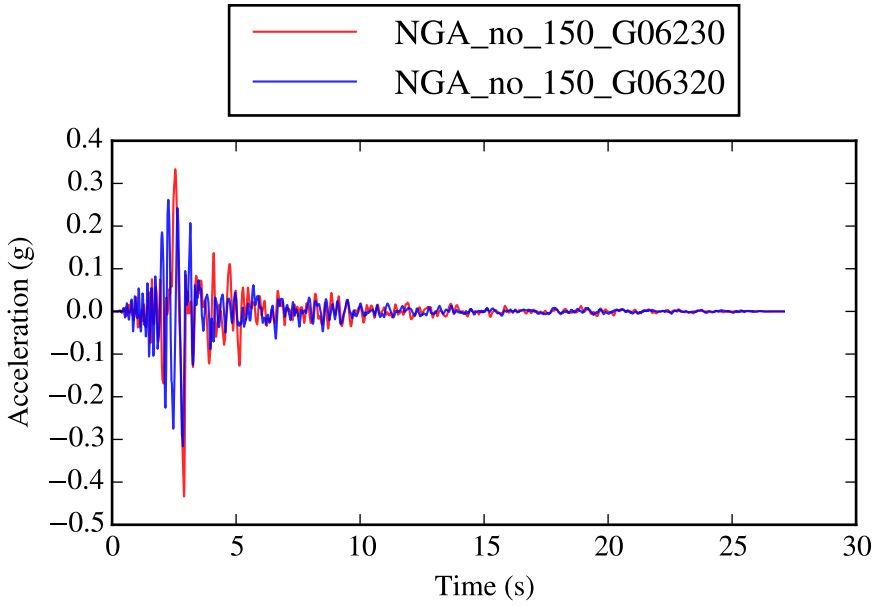


Figure F.34: WUS 34: *150_G06230* and *150_G06320*.

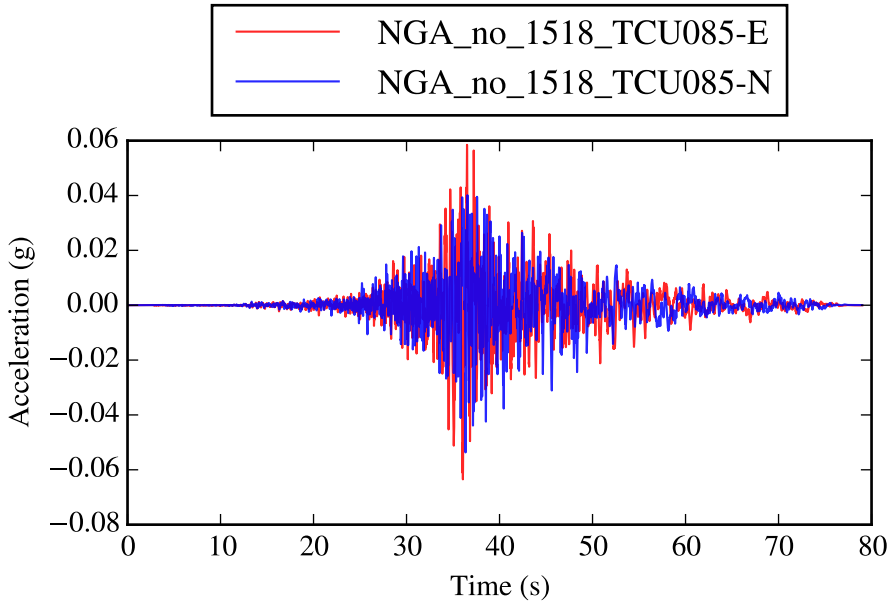


Figure F.35: WUS 35: *1518_TCU085-E* and *1518_TCU085-N*.

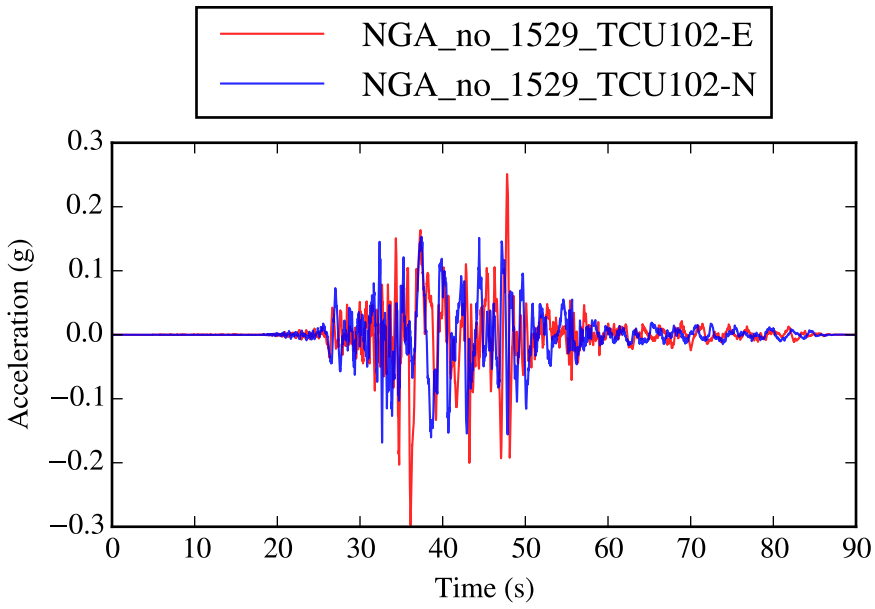


Figure F.36: WUS 36: *1529_TCU102-E* and *1529_TCU102-N*.

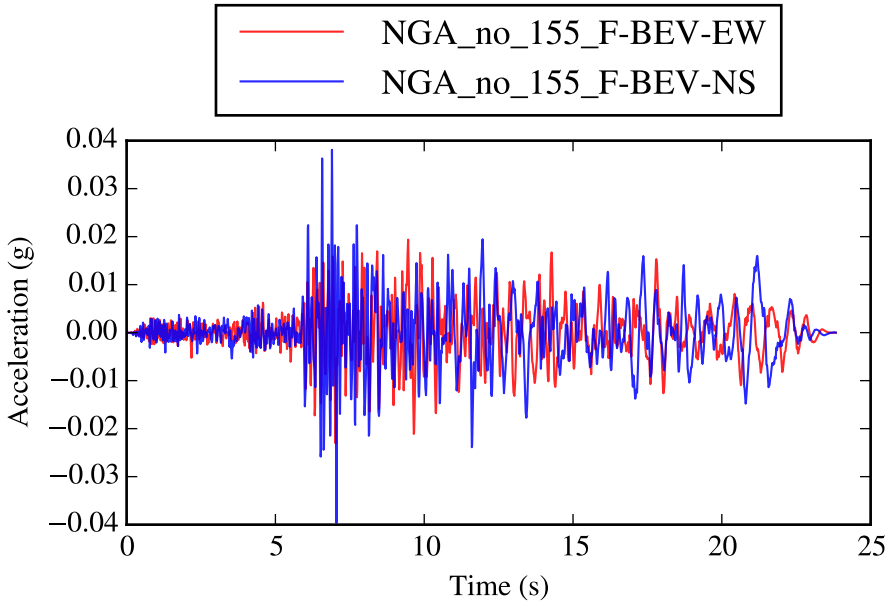


Figure F.37: WUS 37: *155_F-BEV-EW* and *155_F-BEV-NS*.

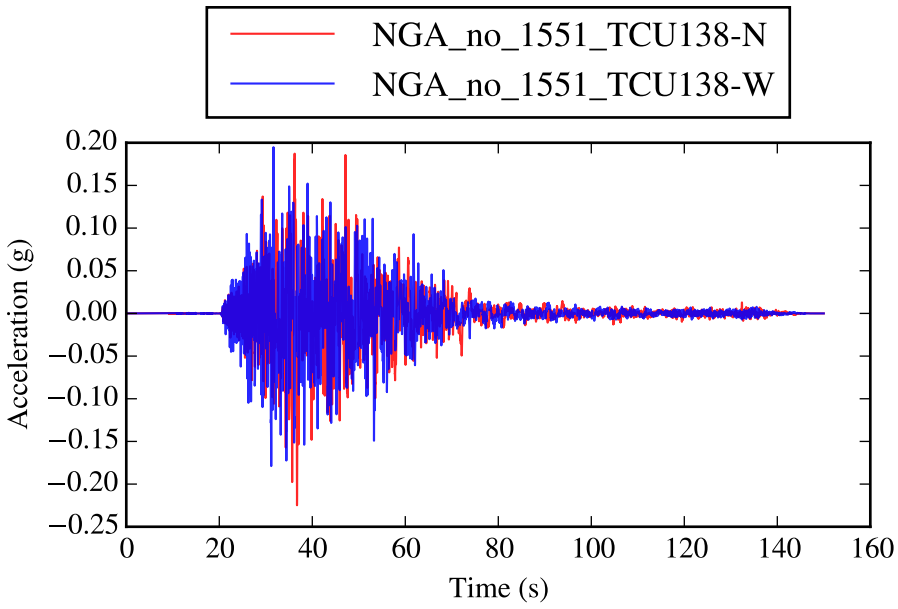


Figure F.38: WUS 38: *1551_TCU138-N* and *1551_TCU138-W*.

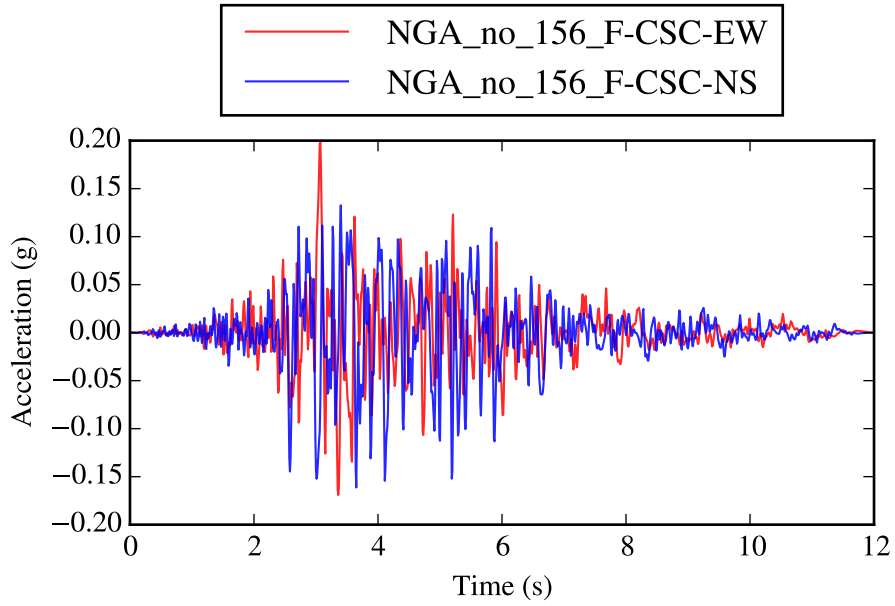


Figure F.39: WUS 39: *156_F-CSC-EW* and *156_F-CSC-NS*.

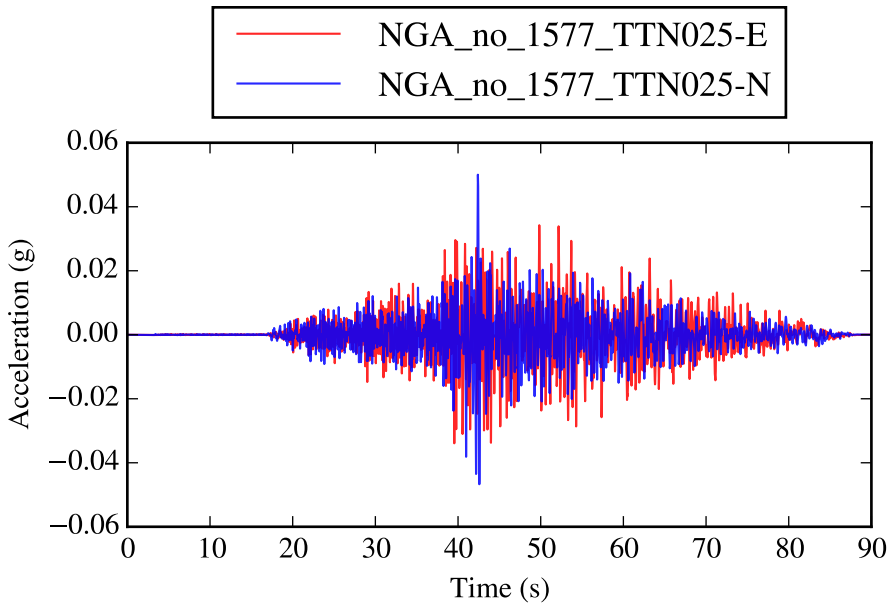


Figure F.40: WUS 40: *1577_TTN025-E* and *1577_TTN025-N*.

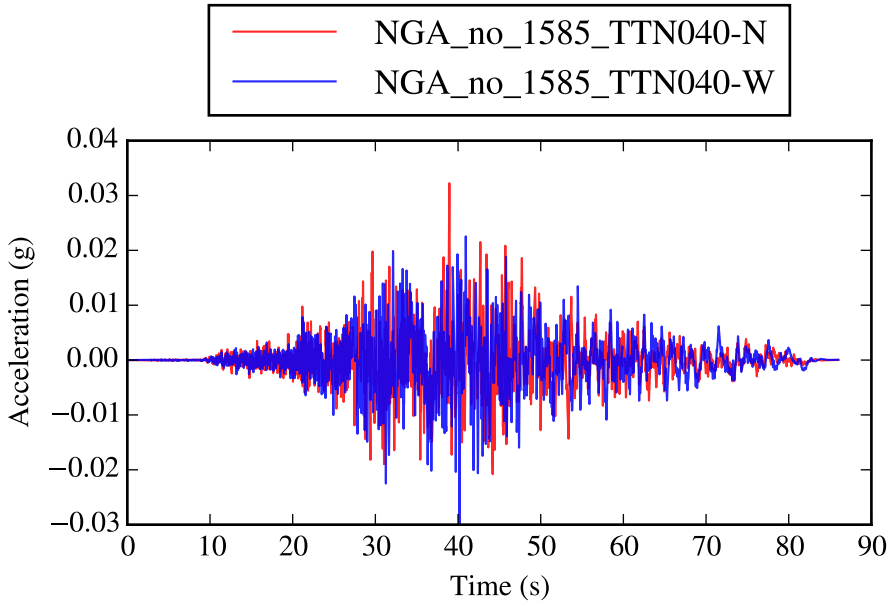


Figure F.41: WUS 41: *1585_TTN040-N* and *1585_TTN040-W*.

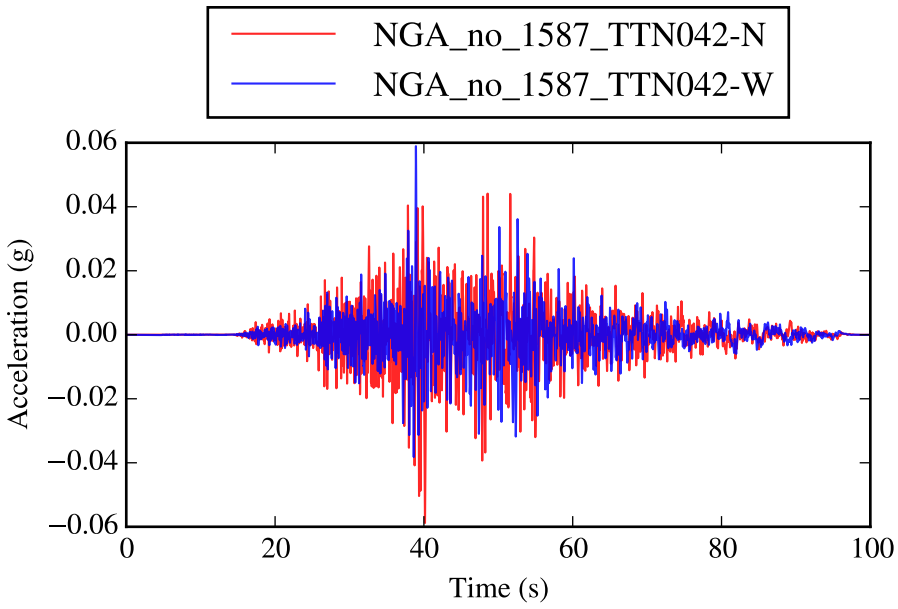


Figure F.42: WUS 42: *1587_TTN042-N* and *1587_TTN042-W*.

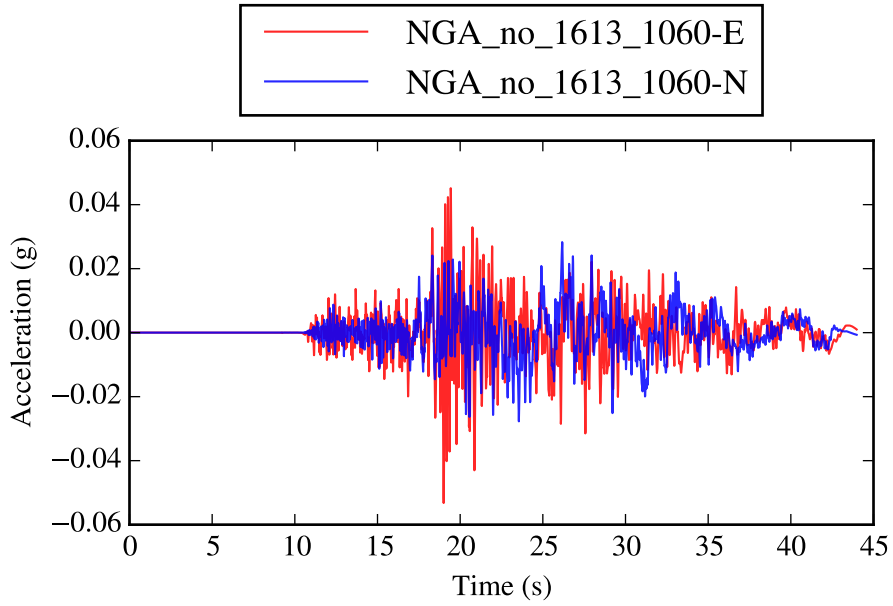


Figure F.43: WUS 43: *1613_1060-E* and *1613_1060-N*.

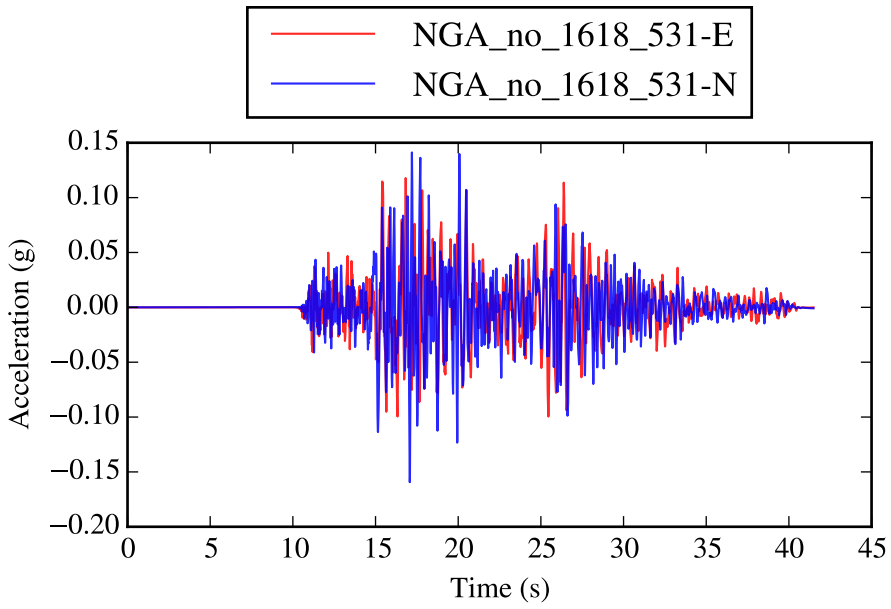


Figure F.44: WUS 44: *1618_531-E* and *1618_531-N*.

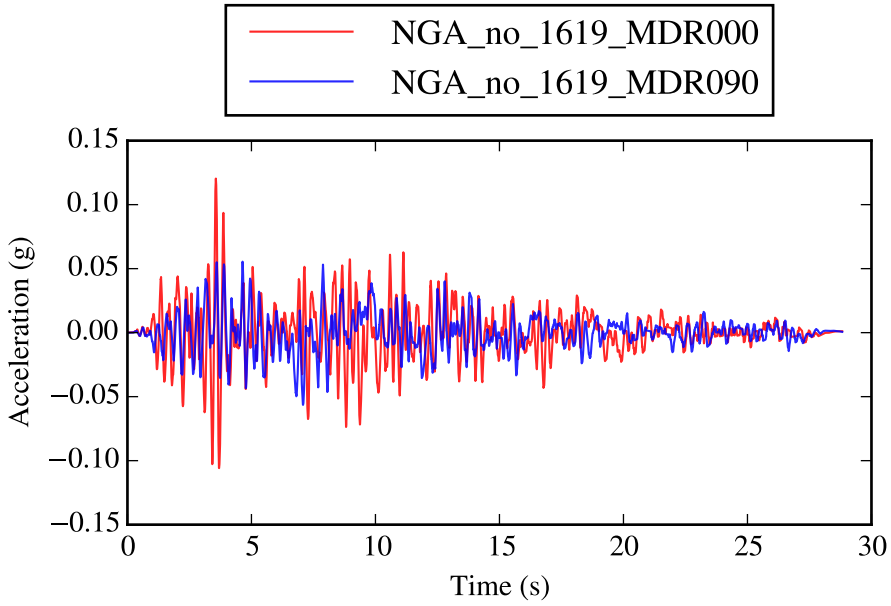


Figure F.45: WUS 45: *1619_MDR000* and *1619_MDR090*.

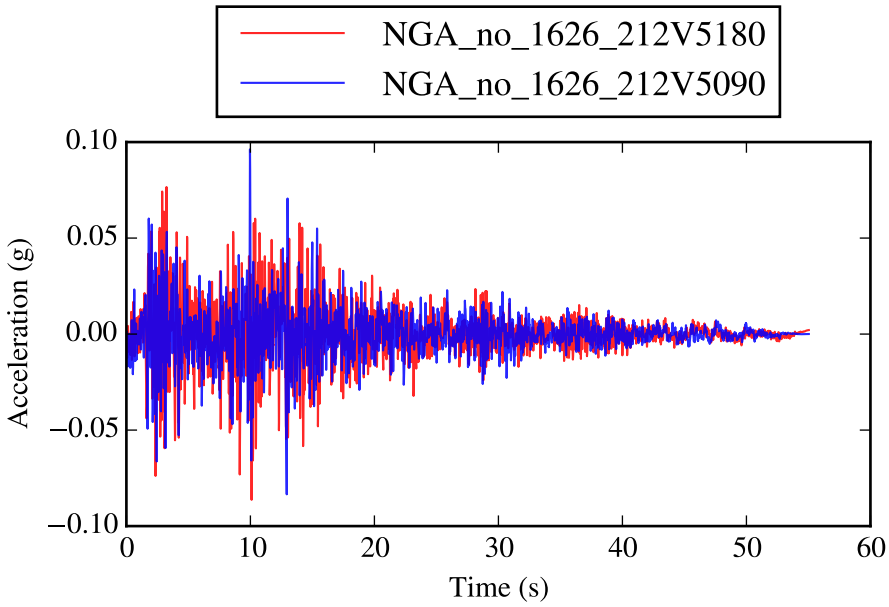


Figure F.46: WUS 46: *1626_212V5180* and *1626_212V5090*.

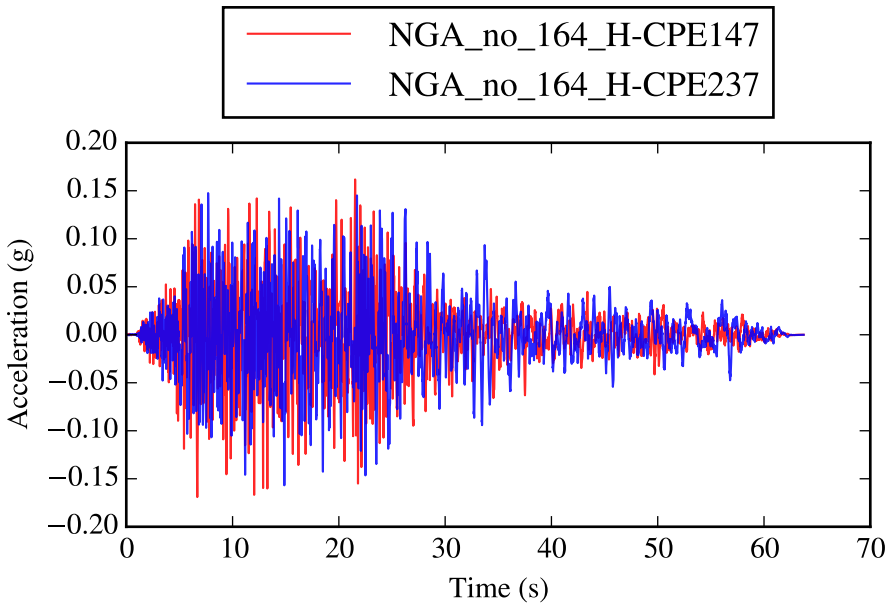


Figure F.47: WUS 47: *164_H-CPE147* and *164_H-CPE237*.

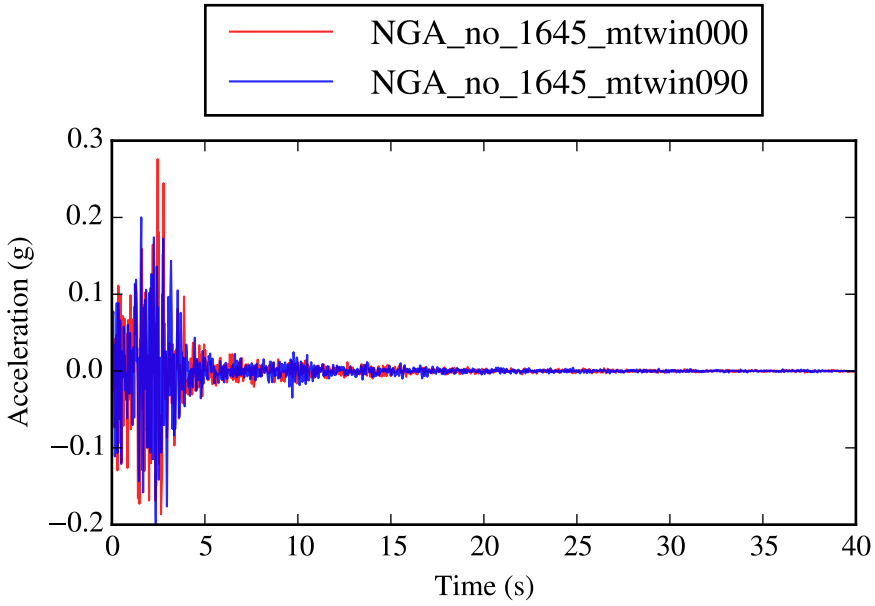


Figure F.48: WUS 48: *1645_mtwin000* and *1645_mtwin090*.

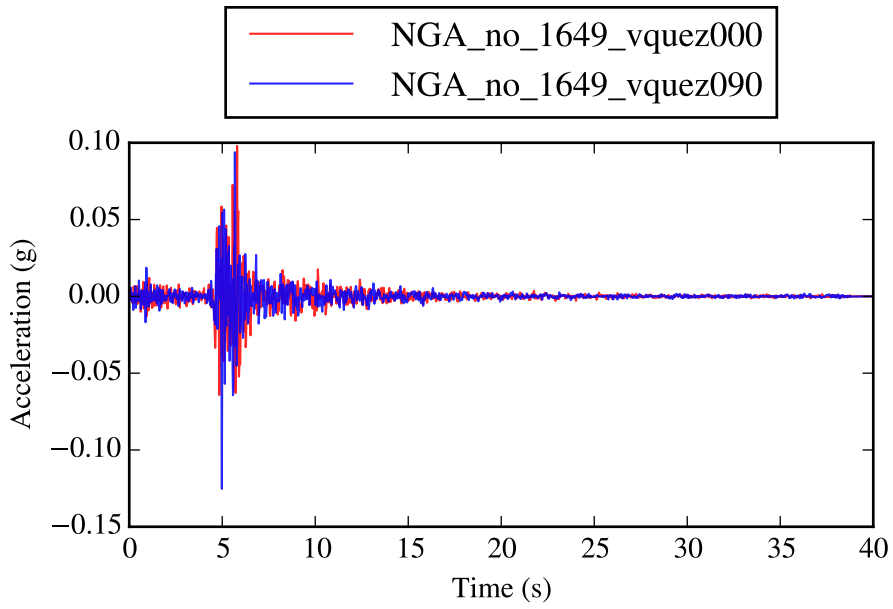


Figure F.49: WUS 49: *1649_vquez000* and *1649_vquez090*.

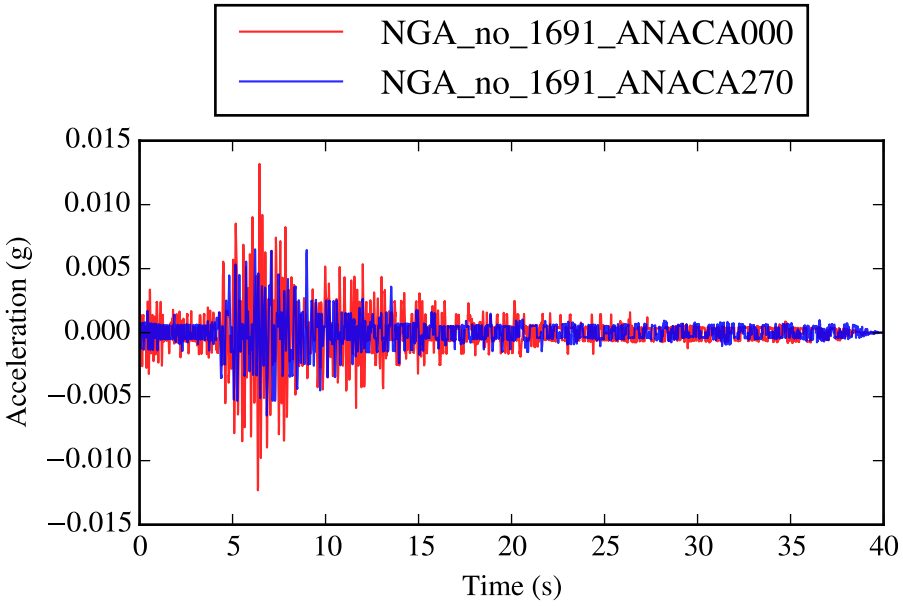


Figure F.50: WUS 50: *1691_ANACA000* and *1691_ANACA270*.

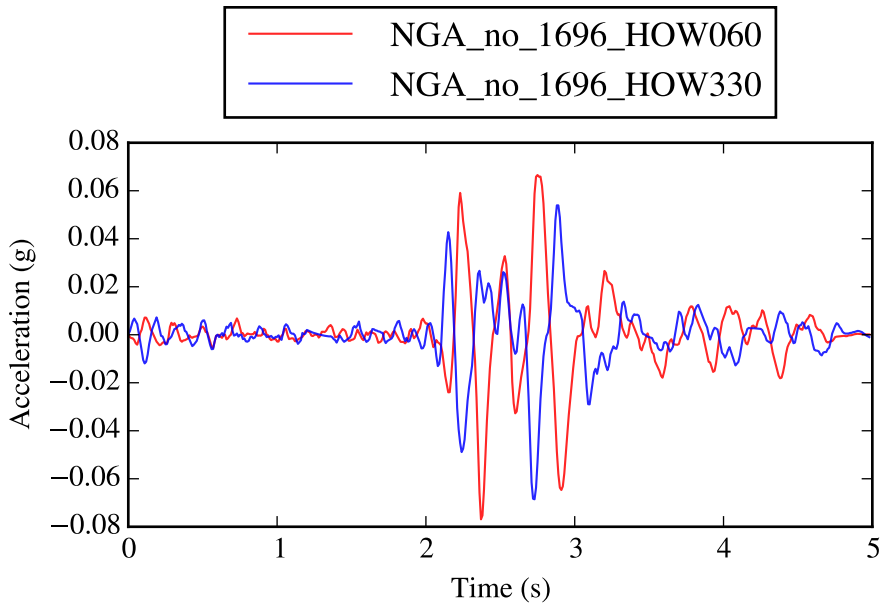


Figure F.51: WUS 51: *1696_HOW060* and *1696_HOW330*.

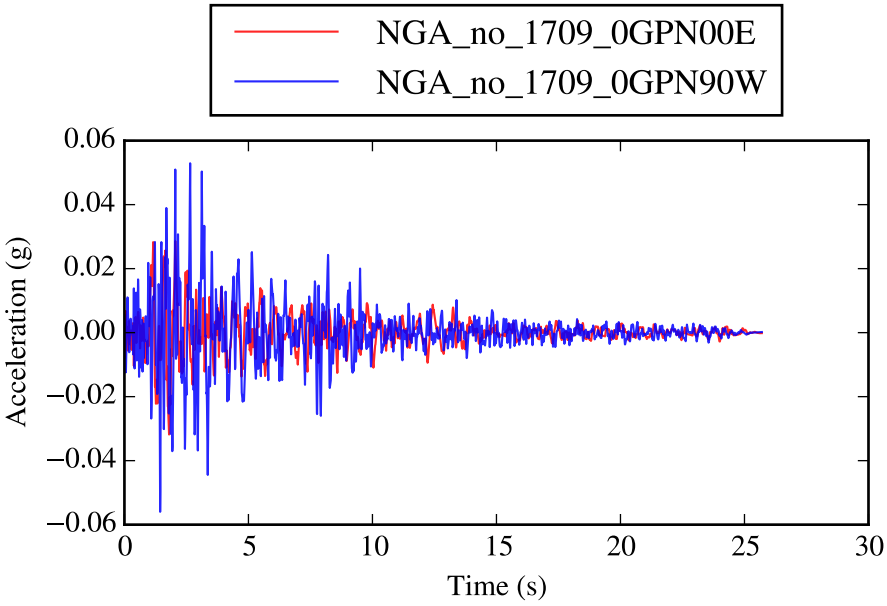


Figure F.52: WUS 52: *1709_0GPN00E* and *1709_0GPN90W*.

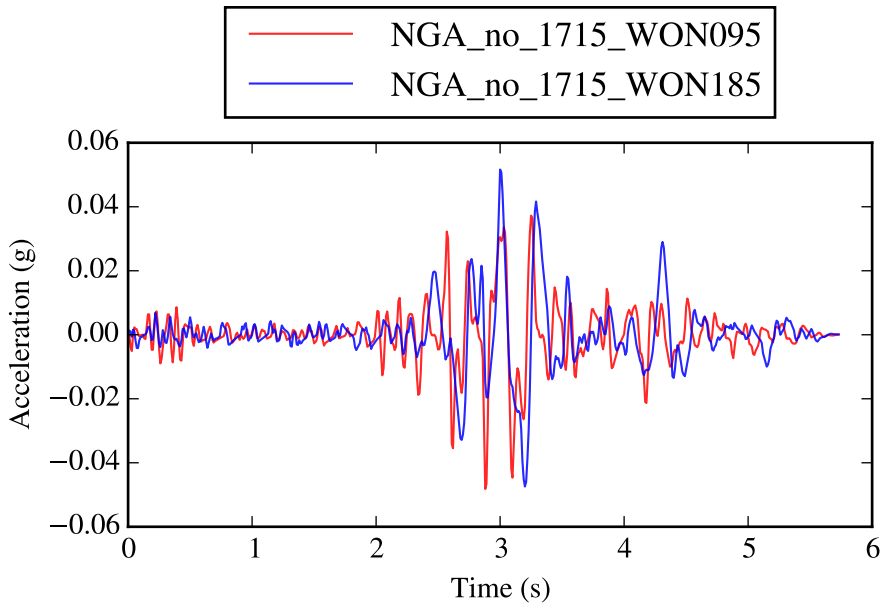


Figure F.53: WUS 53: *1715_WON095* and *1715_WON185*.

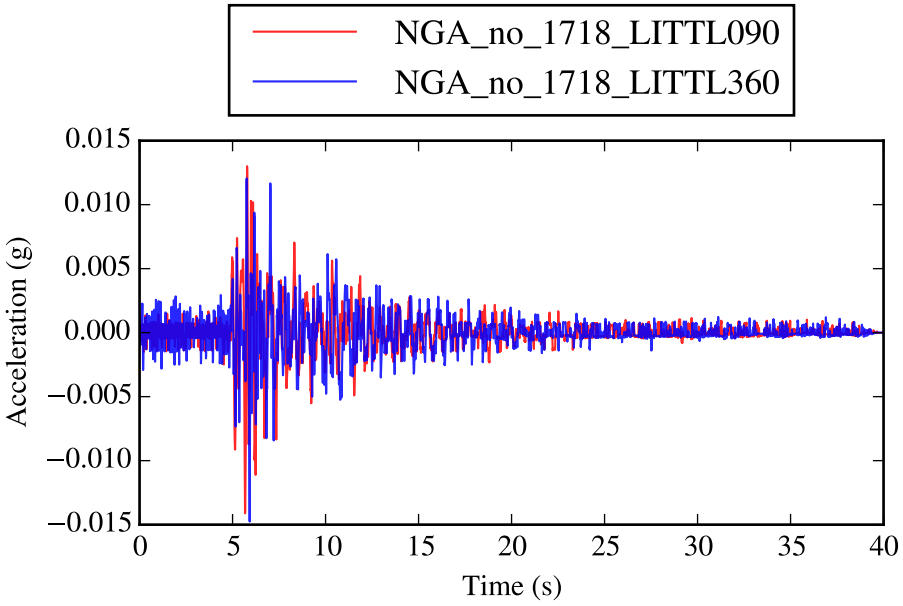


Figure F.54: WUS 54: *1718_LITTL090* and *1718_LITTL360*.

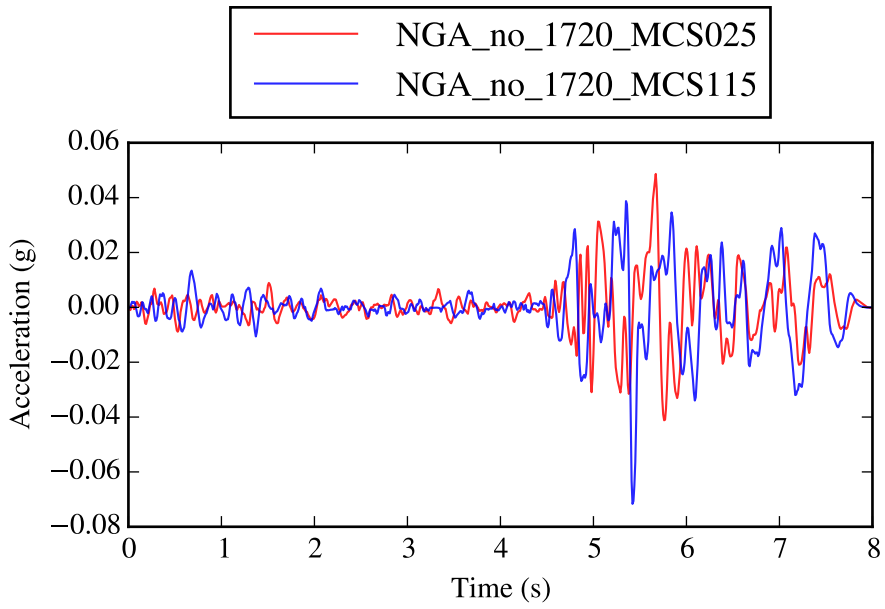


Figure F.55: WUS 55: *1720_MCS025* and *1720_MCS115*.

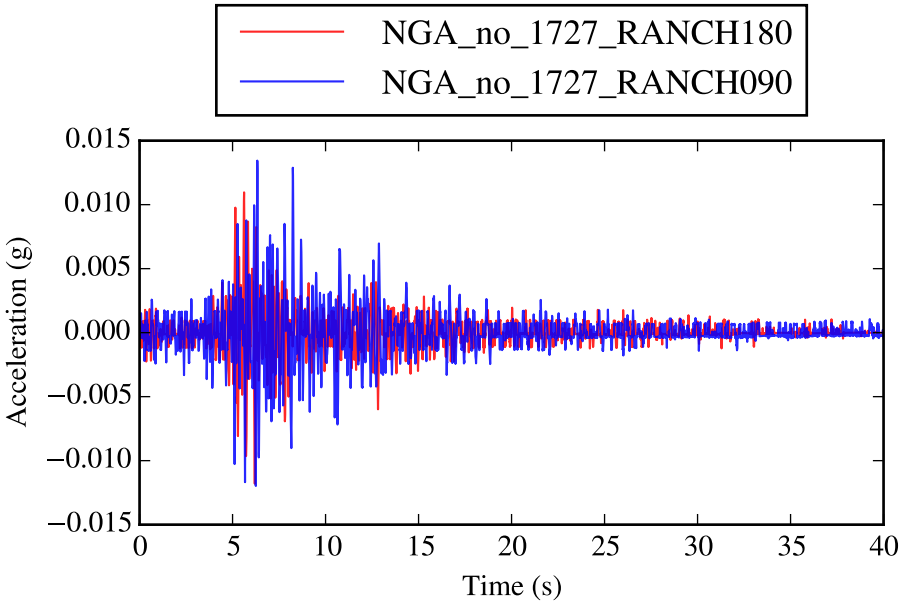


Figure F.56: WUS 56: *1727_RANCH180* and *1727_RANCH090*.

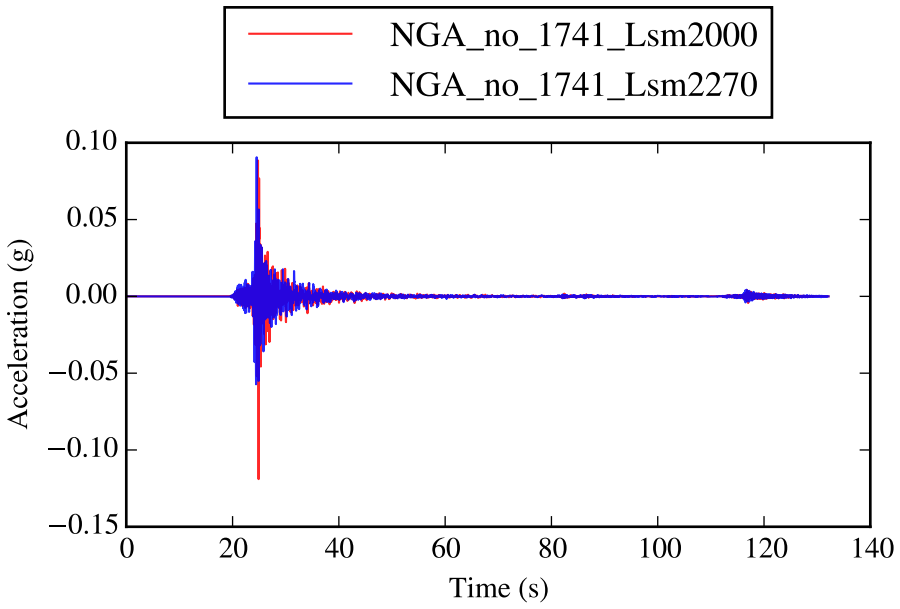


Figure F.57: WUS 57: *1741_Lsm2000* and *1741_Lsm2270*.

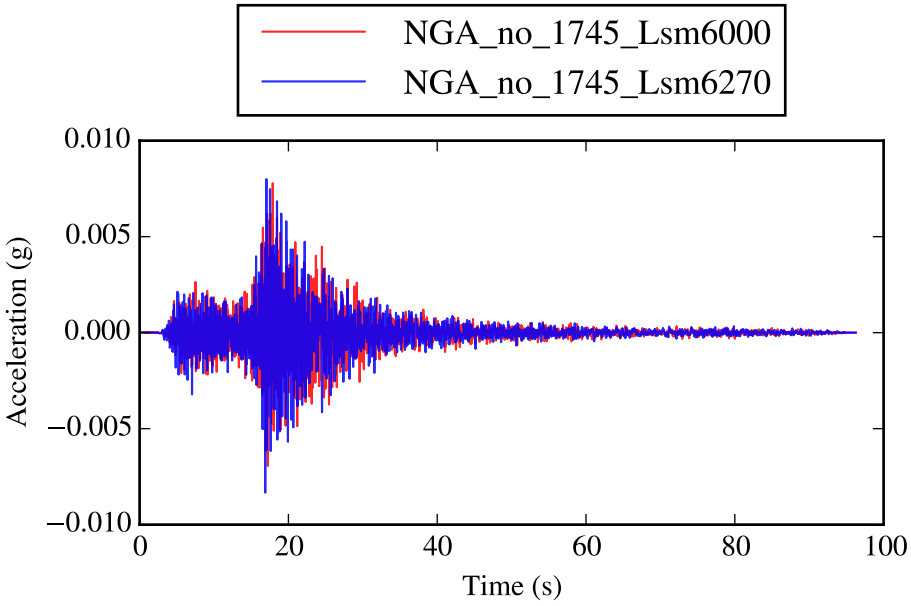


Figure F.58: WUS 58: *1745_Lsm6000* and *1745_Lsm6270*.

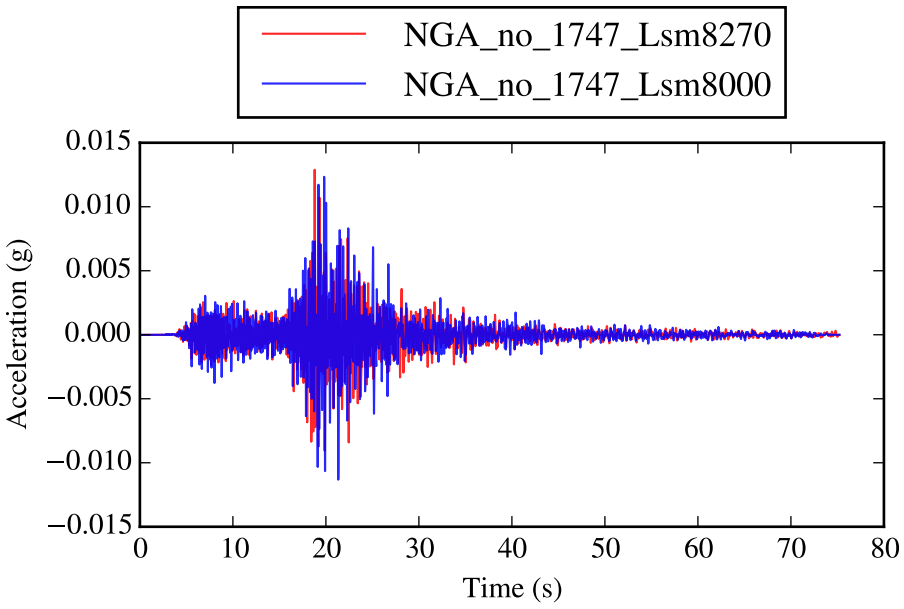


Figure F.59: WUS 59: *1747_Lsm8270* and *1747_Lsm8000*.

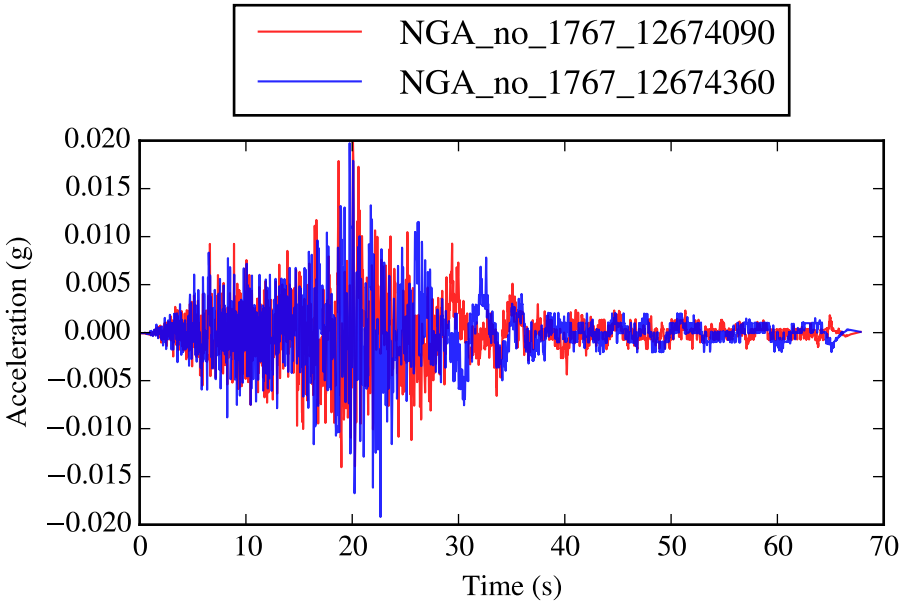


Figure F.60: WUS 60: *1767_12674090* and *1767_12674360*.

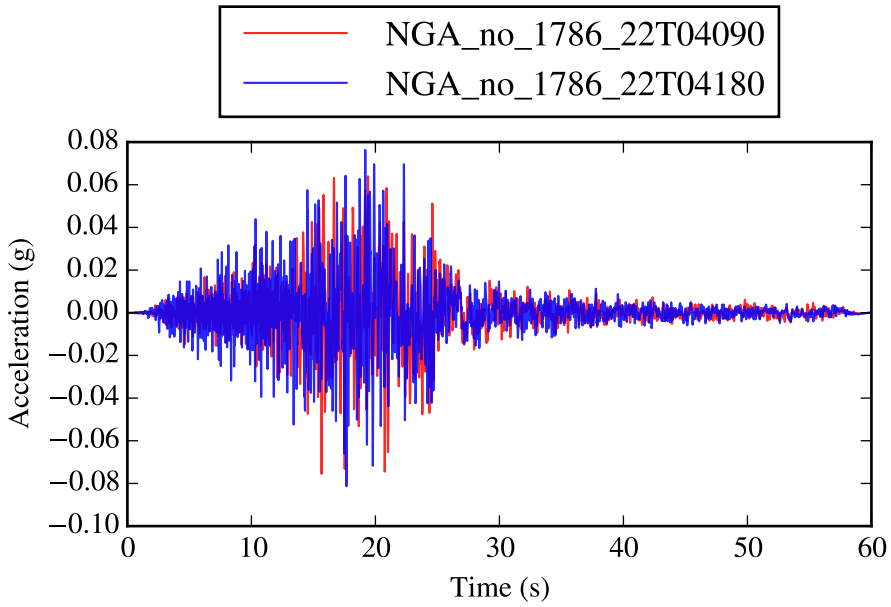


Figure F.61: WUS 61: *1786_22T04090* and *1786_22T04180*.

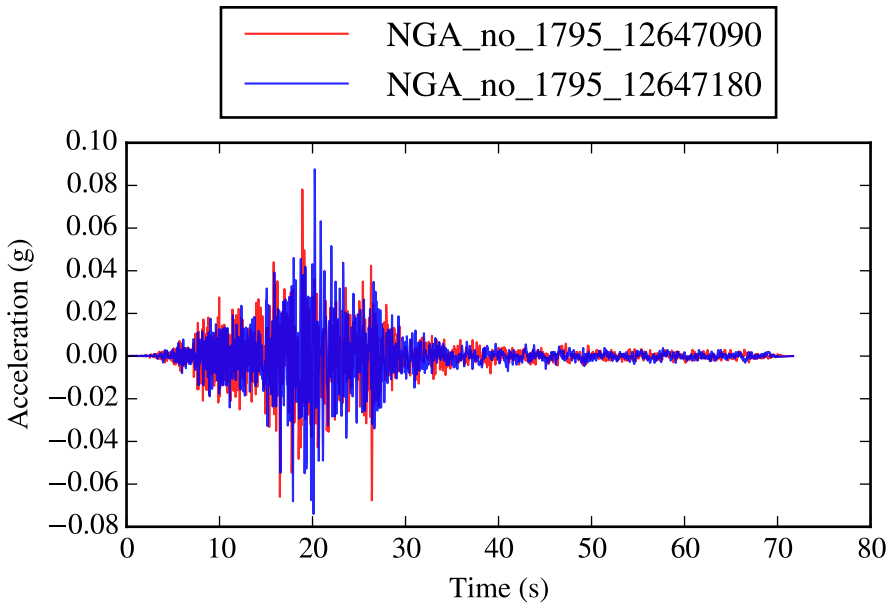


Figure F.62: WUS 62: *1795_12647090* and *1795_12647180*.

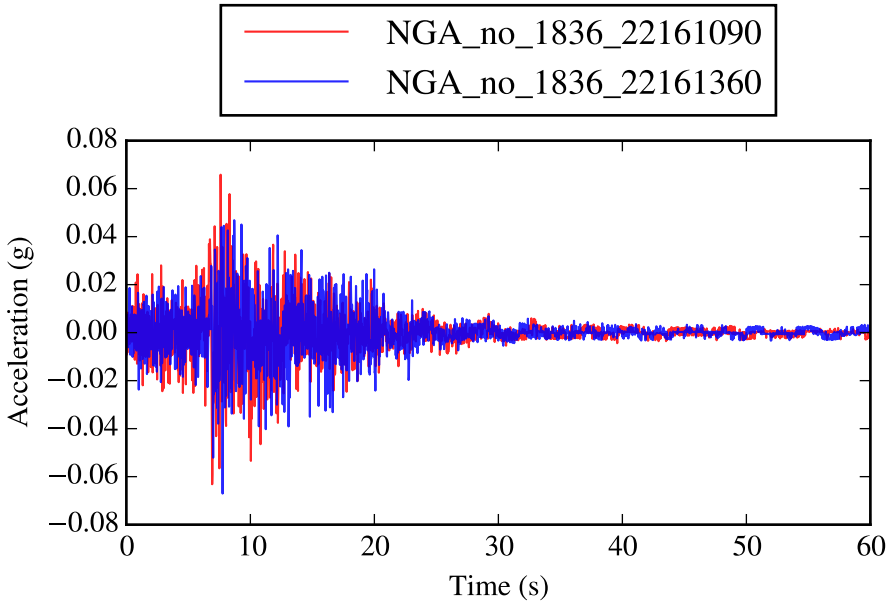


Figure F.63: WUS 63: *1836_22161090* and *1836_22161360*.

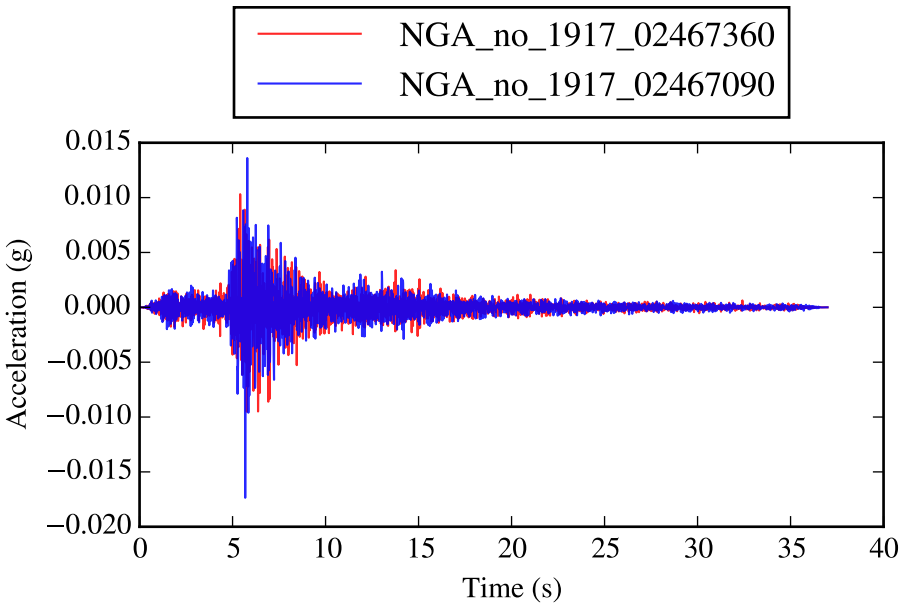


Figure F.64: WUS 64: *1917_02467360* and *1917_02467090*.

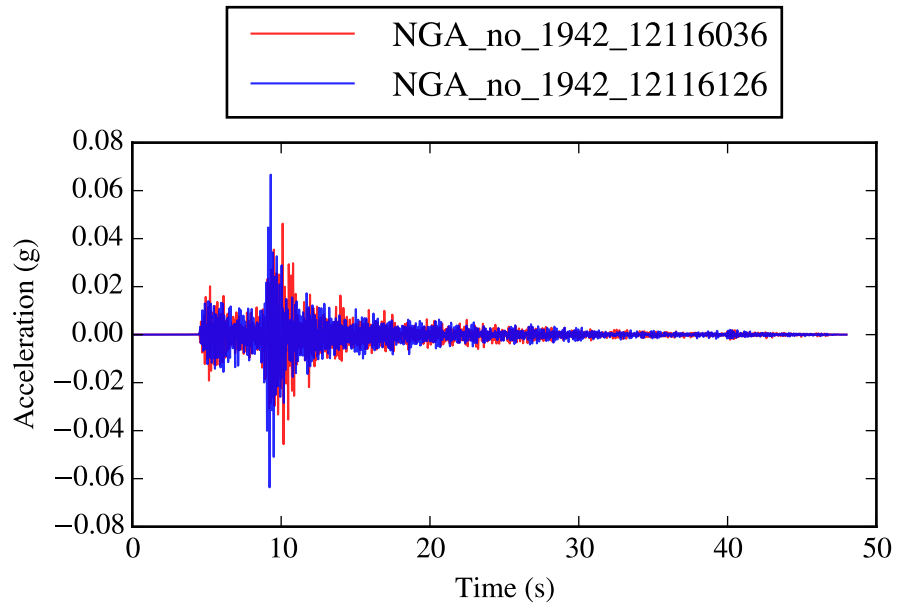


Figure F.65: WUS 65: *1942_12116036* and *1942_12116126*.

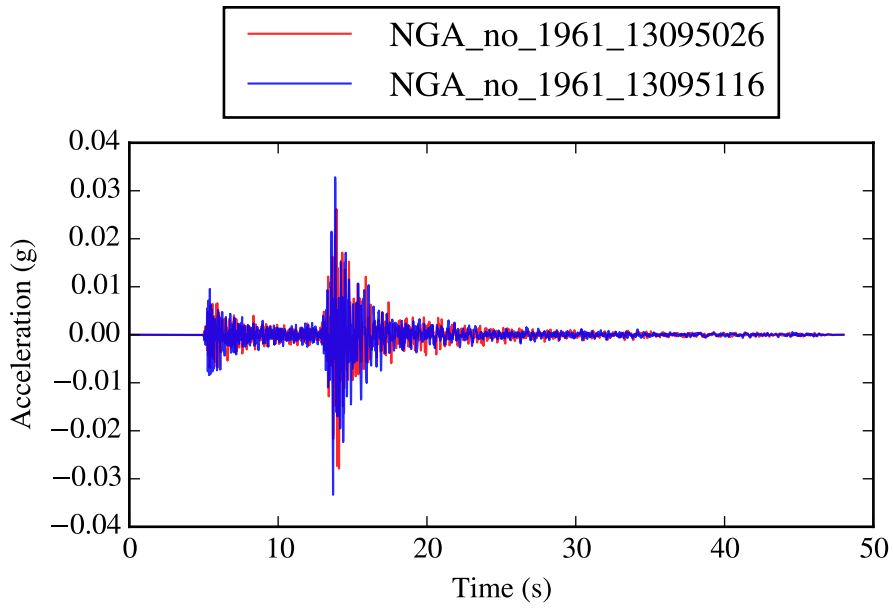


Figure F.66: WUS 66: *1961_13095026* and *1961_13095116*.

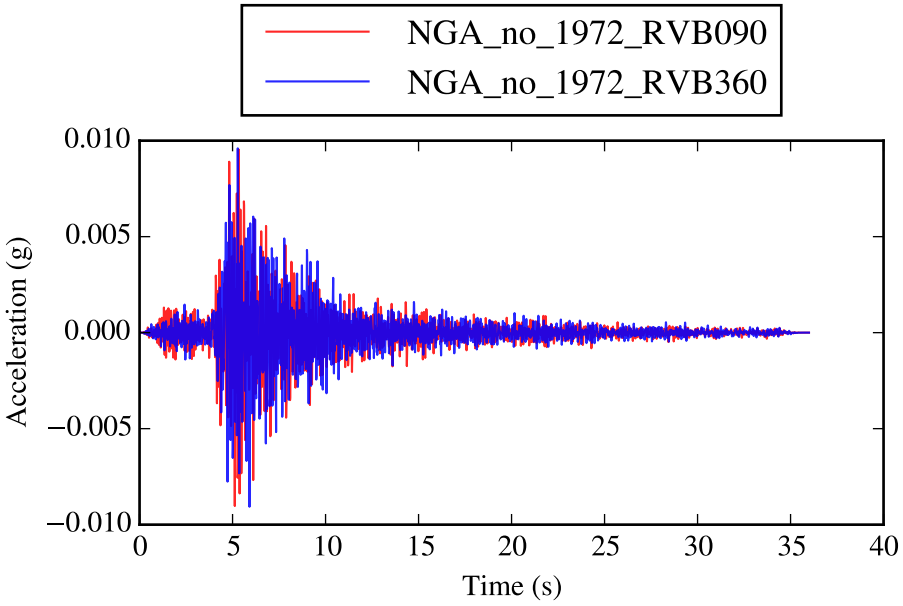


Figure F.67: WUS 67: *1972_RVB090* and *1972_RVB360*.

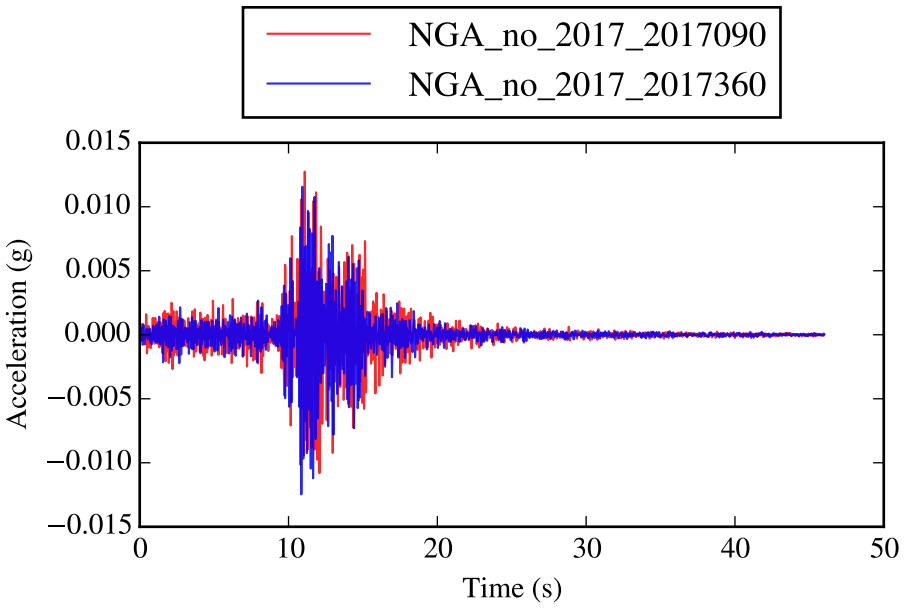


Figure F.68: WUS 68: *2017_2017090* and *2017_2017360*.

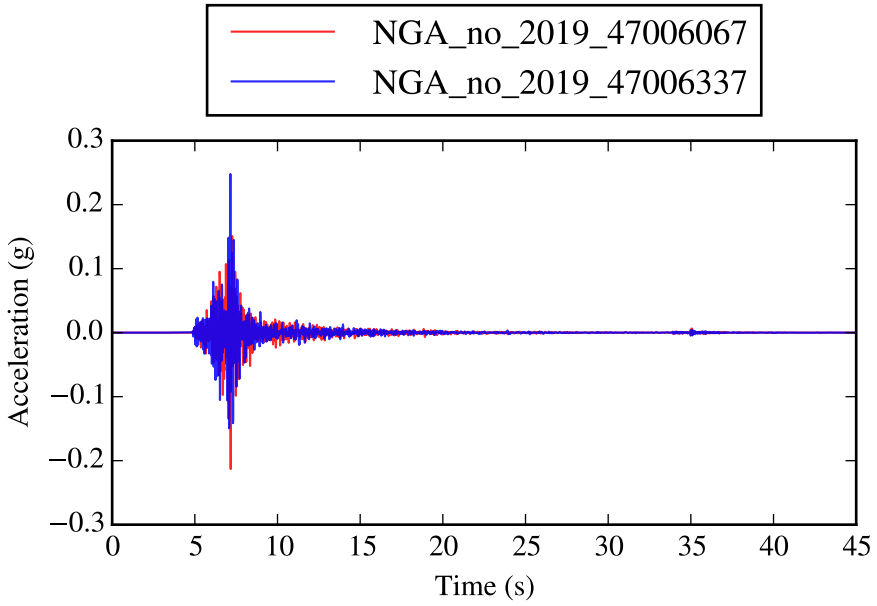


Figure F.69: WUS 69: *2019_47006067* and *2019_47006337*.

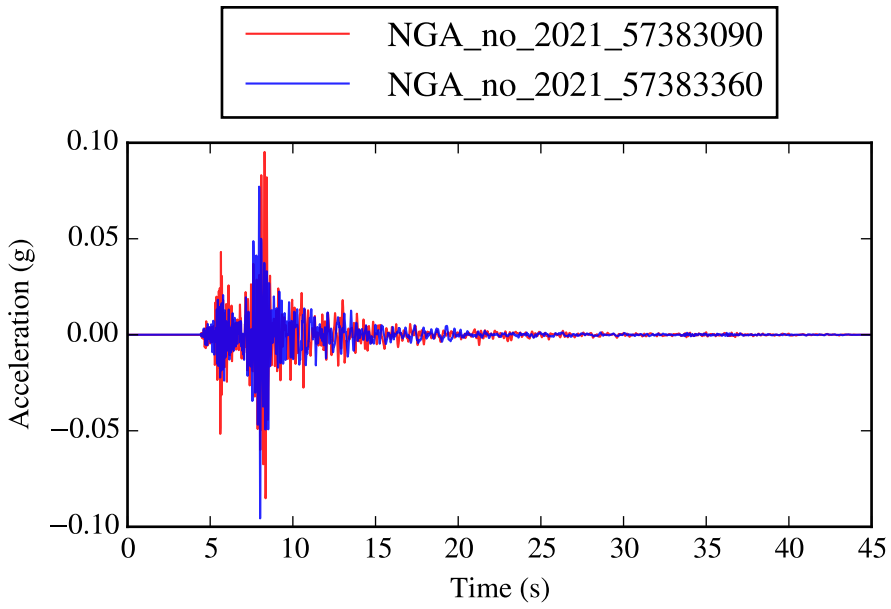


Figure F.70: WUS 70: *2021_57383090* and *2021_57383360*.

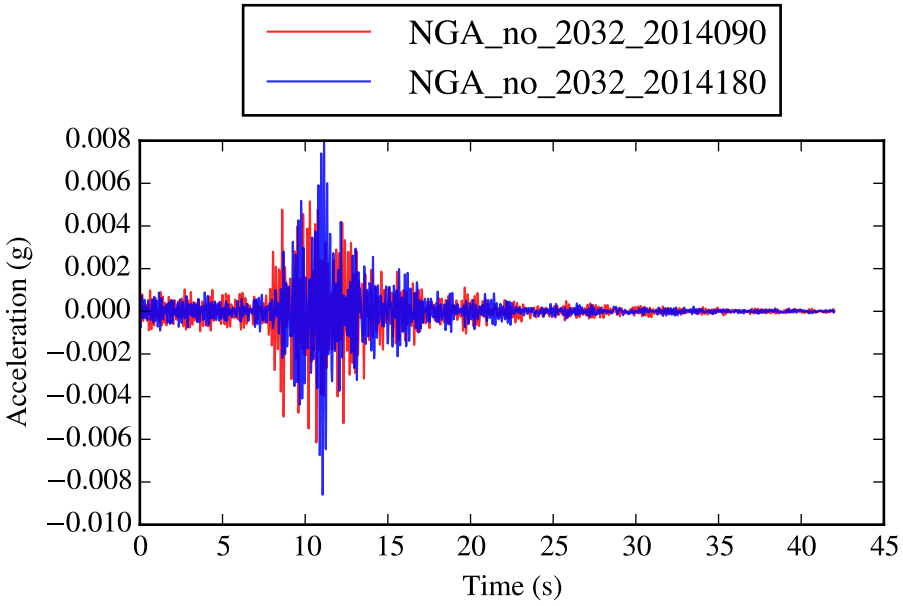


Figure F.71: WUS 71: *2032_2014090* and *2032_2014180*.

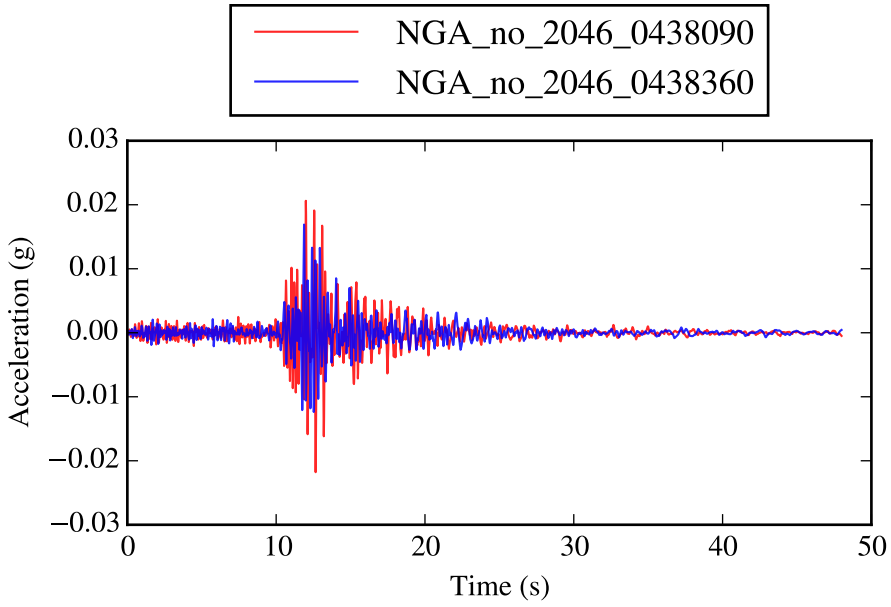


Figure F.72: WUS 72: *2046_0438090* and *2046_0438360*.

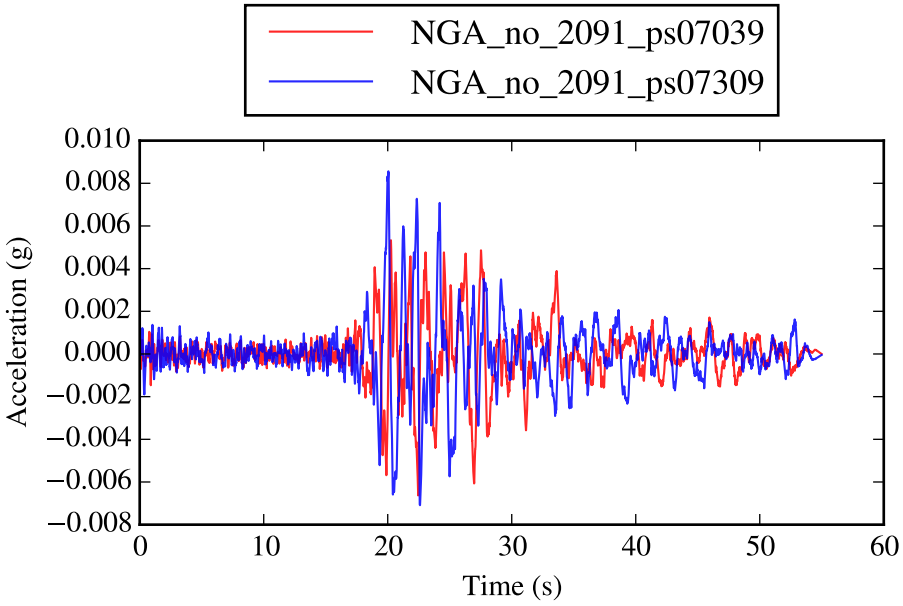


Figure F.73: WUS 73: *2091_ps07039* and *2091_ps07309*.

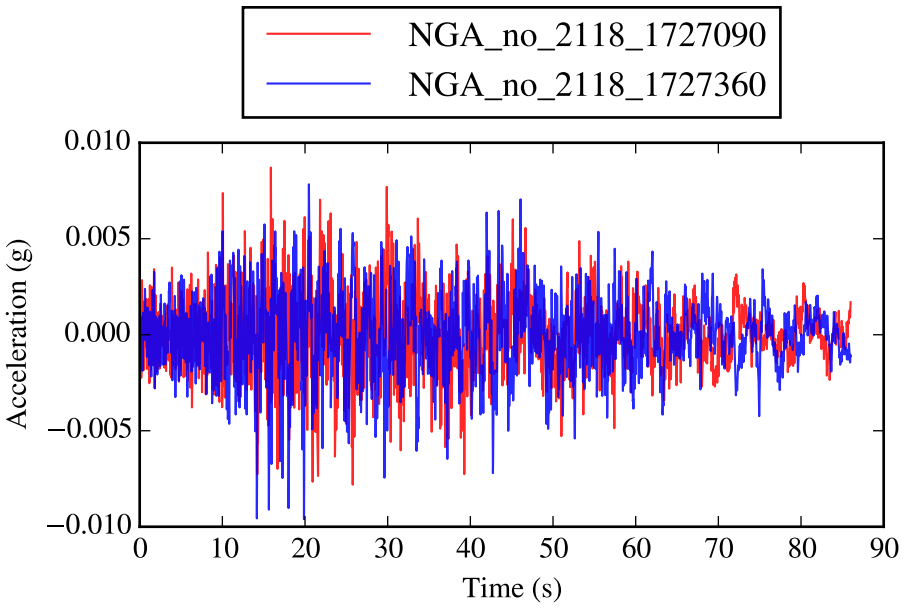


Figure F.74: WUS 74: *2118_1727090* and *2118_1727360*.

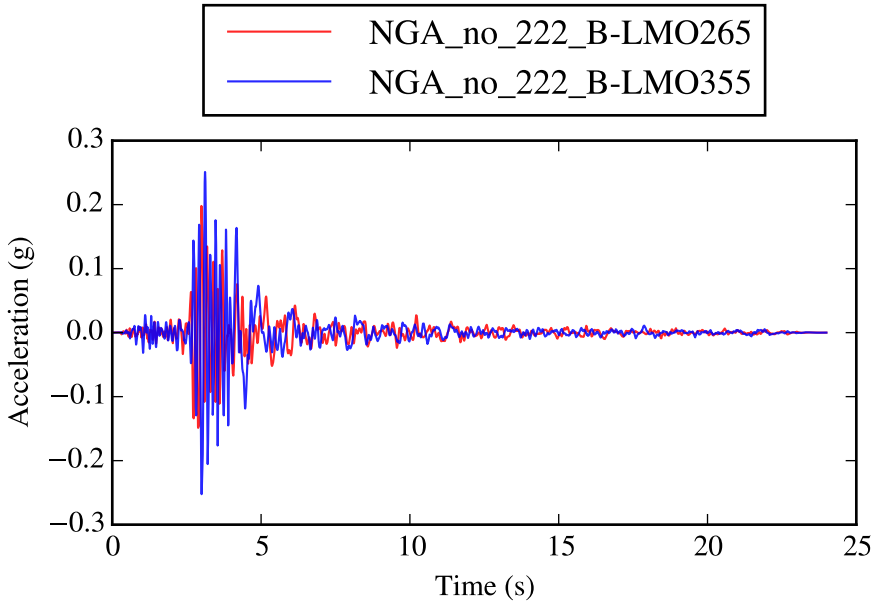


Figure F.75: WUS 75: *222_B-LMO265* and *222_B-LMO355*.

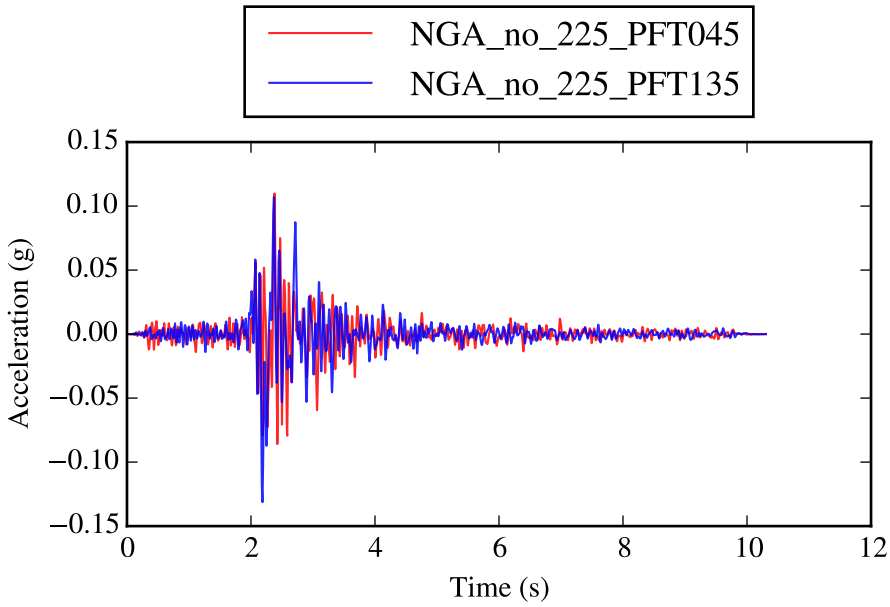


Figure F.76: WUS 76: *225_PFT045* and *225_PFT135*.

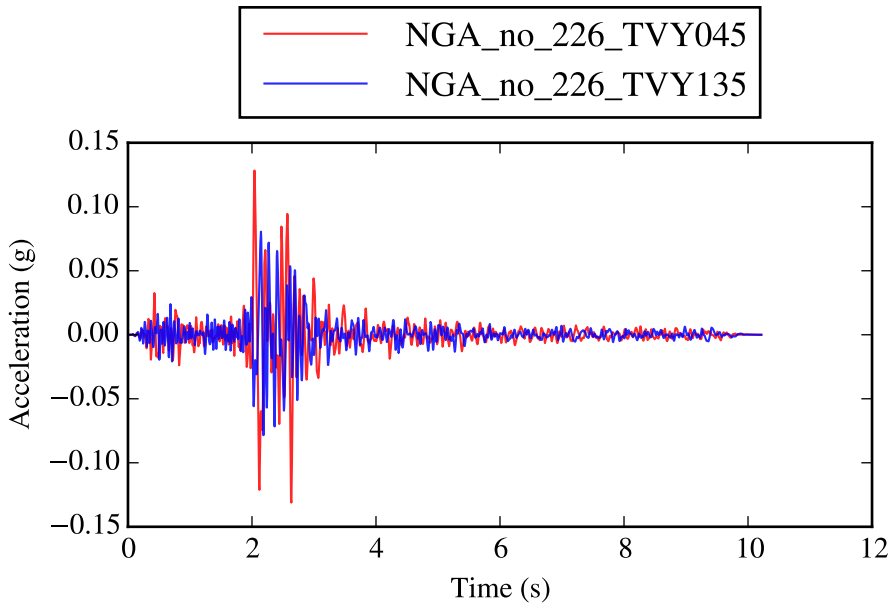


Figure F.77: WUS 77: *226_TVY045* and *226_TVY135*.

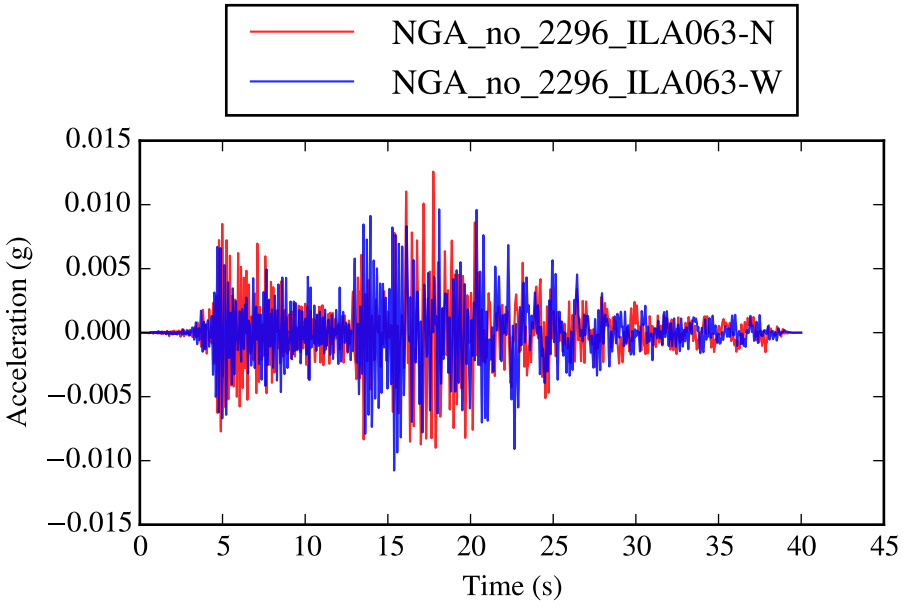


Figure F.78: WUS 78: *2296_ILA063-N* and *2296_ILA063-W*.

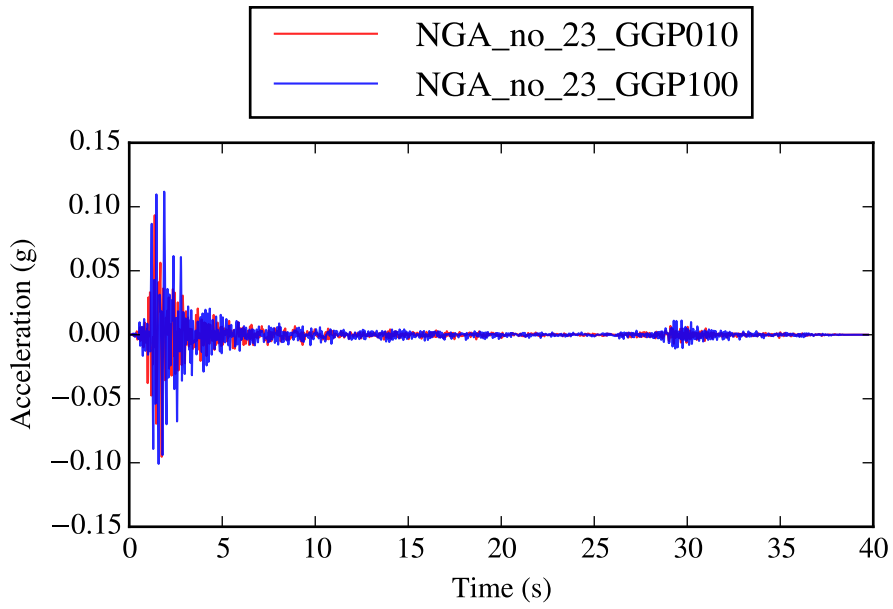


Figure F.79: WUS 79: *23_GGP010* and *23_GGP100*.

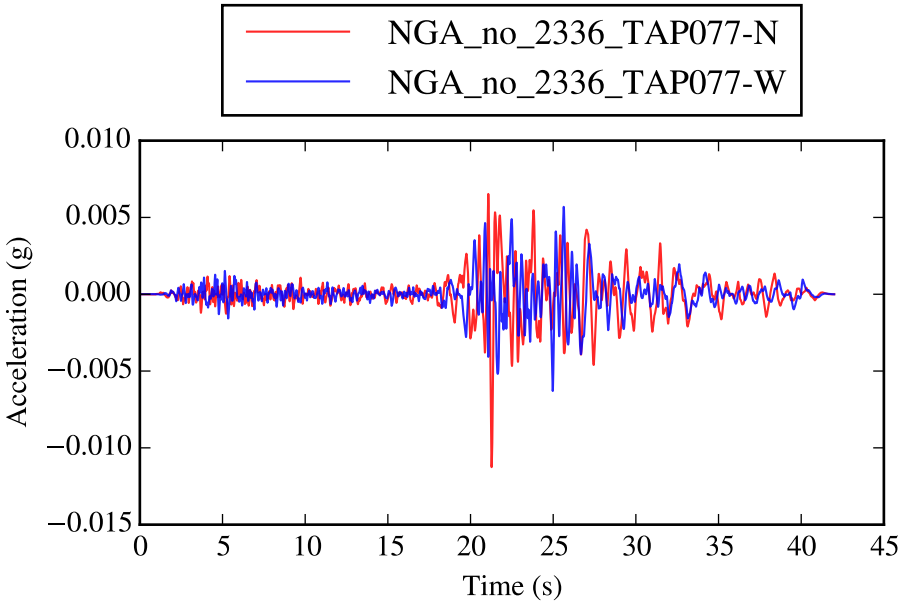


Figure F.80: WUS 80: *2336_TAP077-N* and *2336_TAP077-W*.

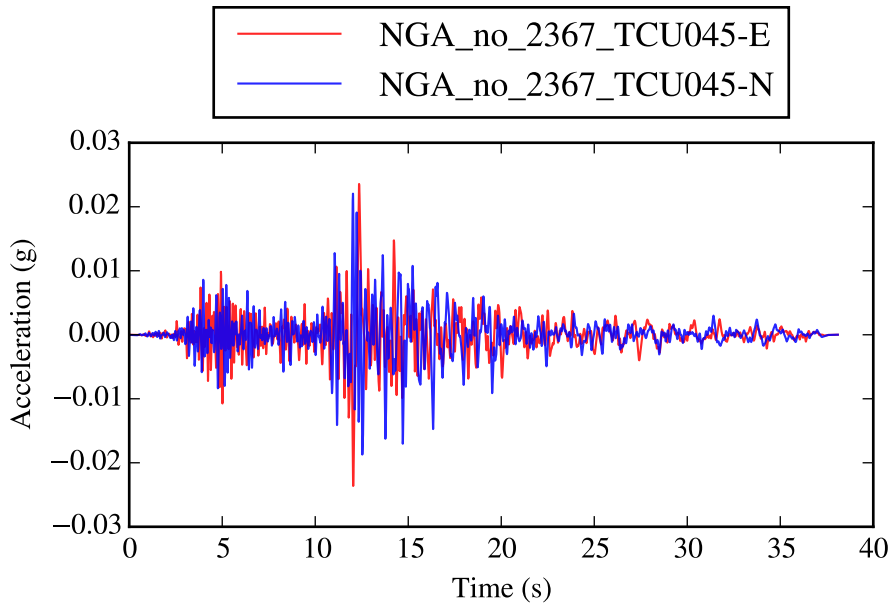


Figure F.81: WUS 81: *2367_TCU045-E* and *2367_TCU045-N*.

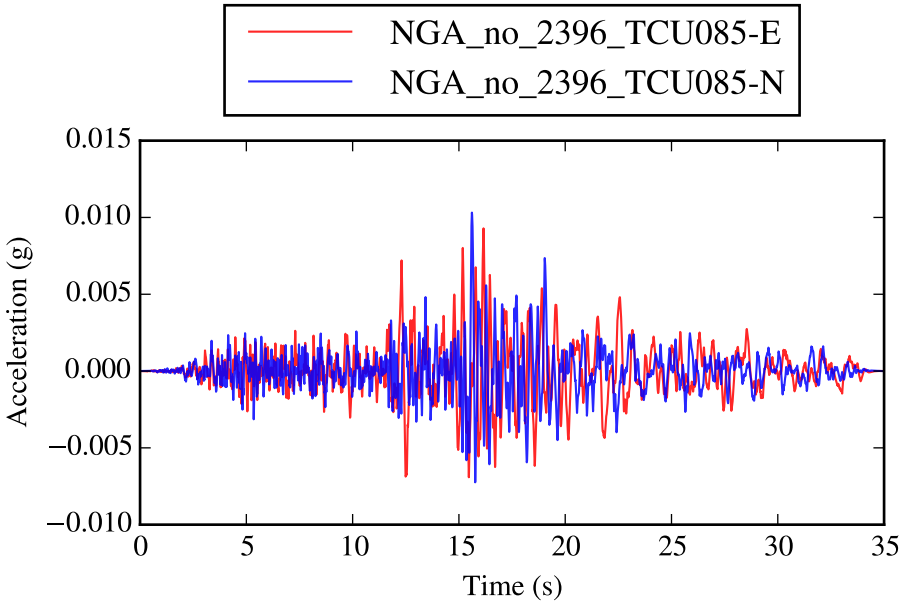


Figure F.82: WUS 82: *2396_TCU085-E* and *2396_TCU085-N*.

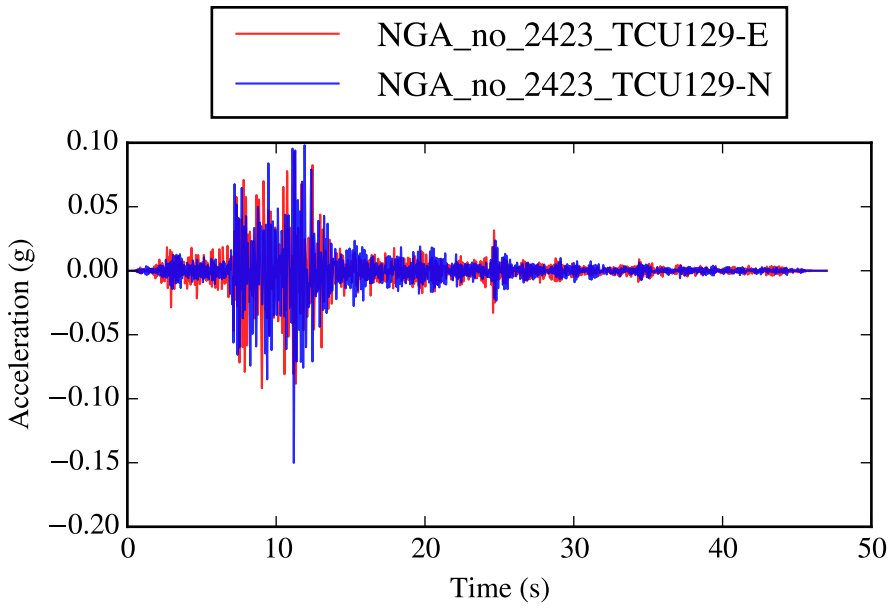


Figure F.83: WUS 83: *2423_TCU129-E* and *2423_TCU129-N*.

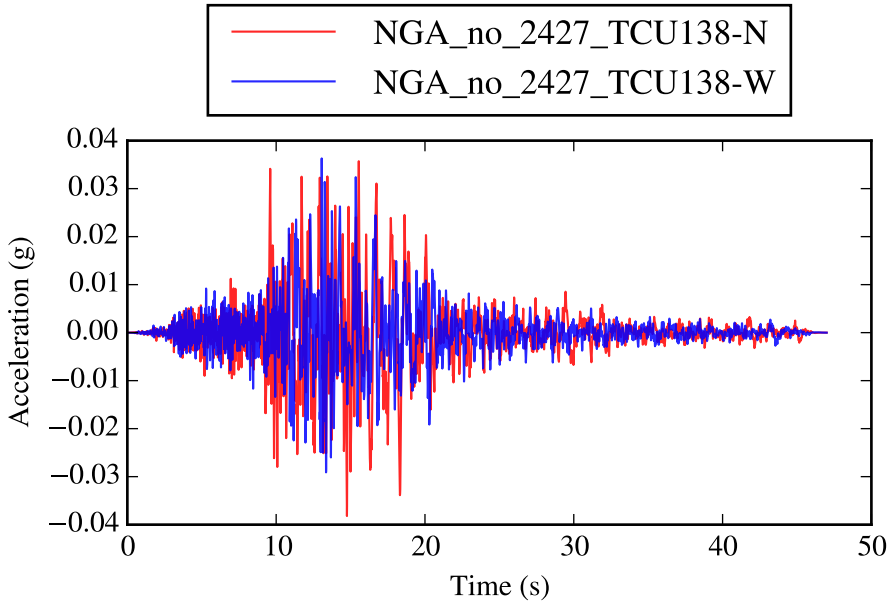


Figure F.84: WUS 84: *2427_TCU138-N* and *2427_TCU138-W*.

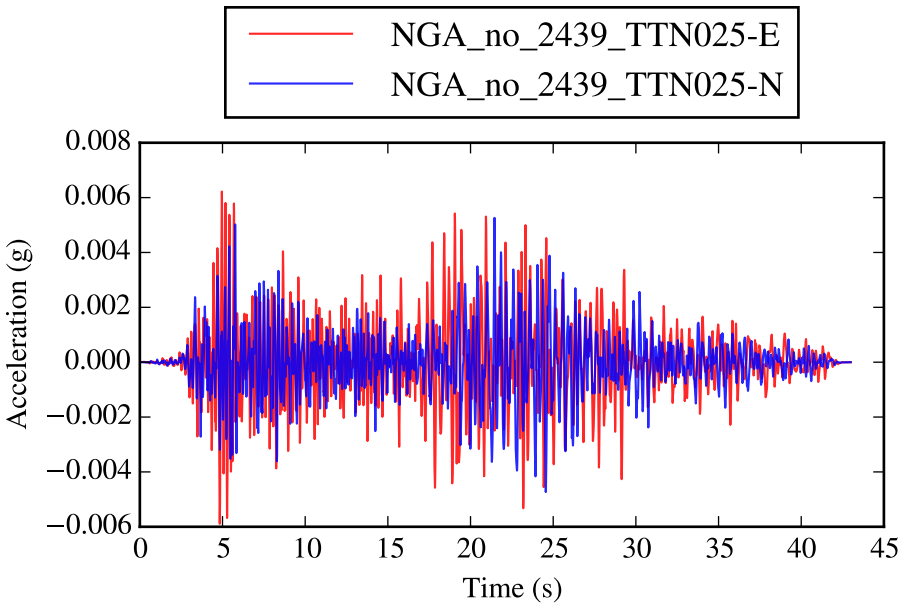


Figure F.85: WUS 85: *2439_TTN025-E* and *2439_TTN025-N*.

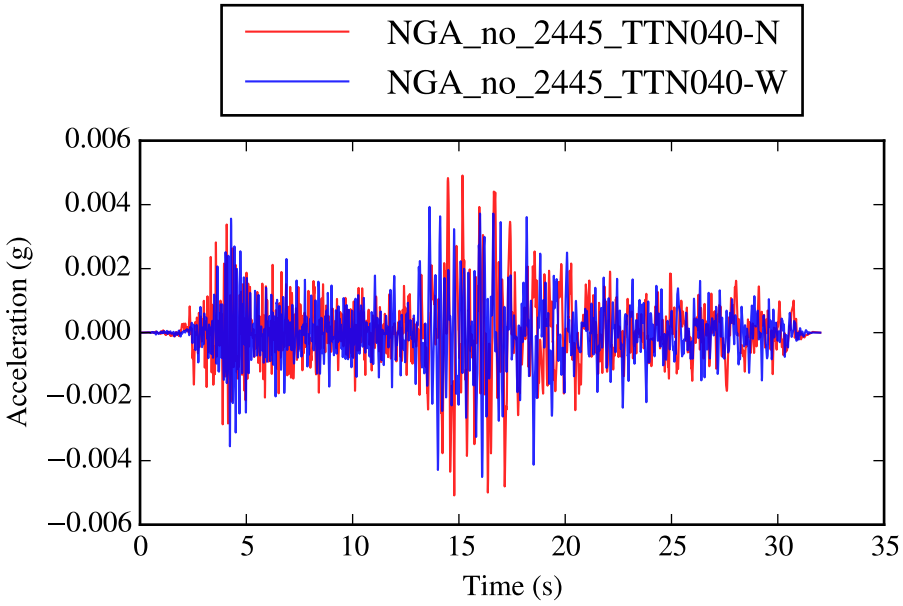


Figure F.86: WUS 86: *2445_TTN040-N* and *2445_TTN040-W*.

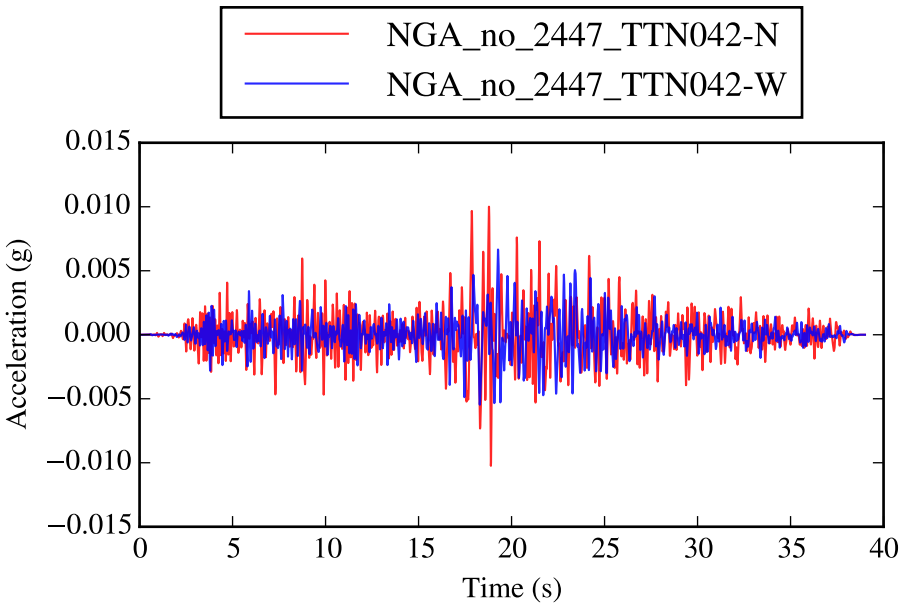


Figure F.87: WUS 87: *2447_TTN042-N* and *2447_TTN042-W*.

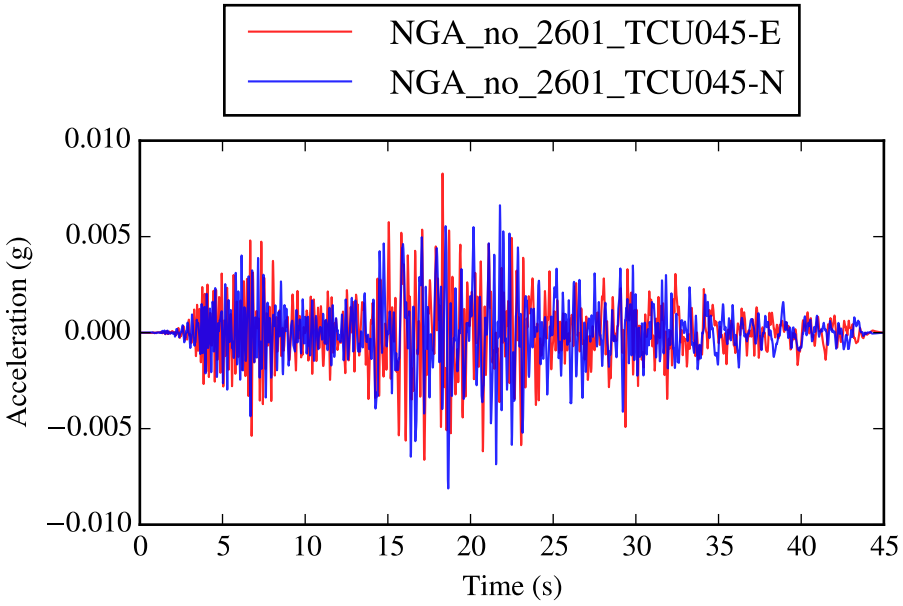


Figure F.88: WUS 88: *2601_TCU045-E* and *2601_TCU045-N*.

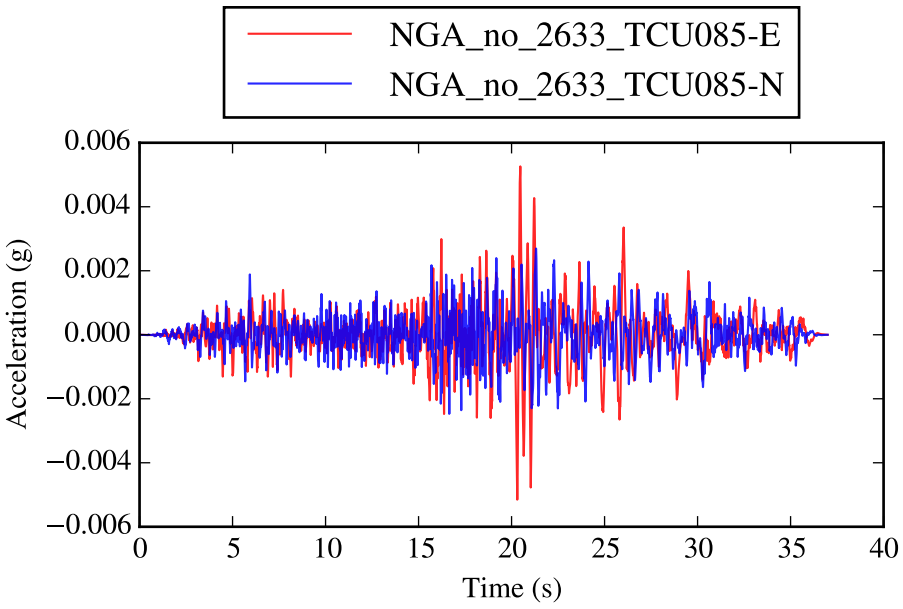


Figure F.89: WUS 89: *2633_TCU085-E* and *2633_TCU085-N*.

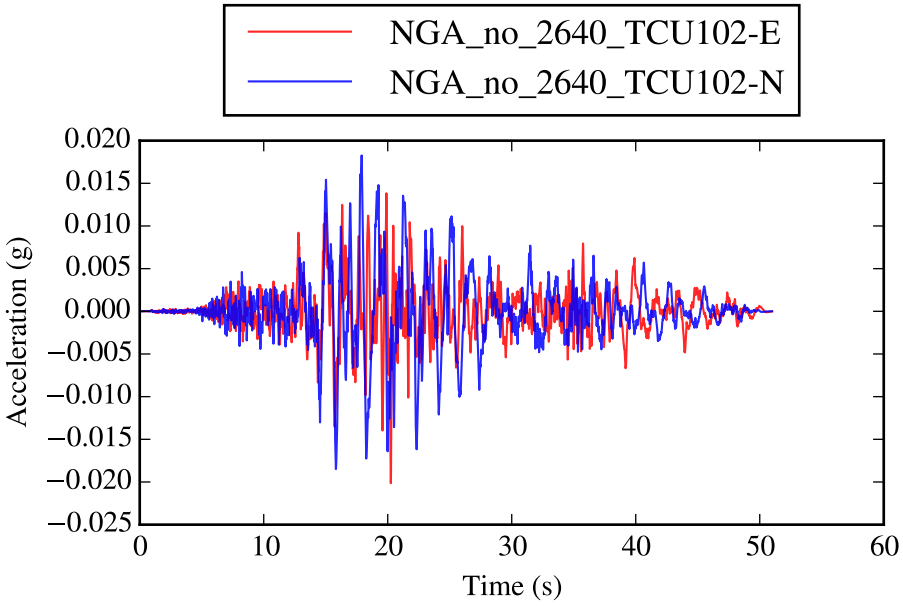


Figure F.90: WUS 90: *2640_TCU102-E* and *2640_TCU102-N*.

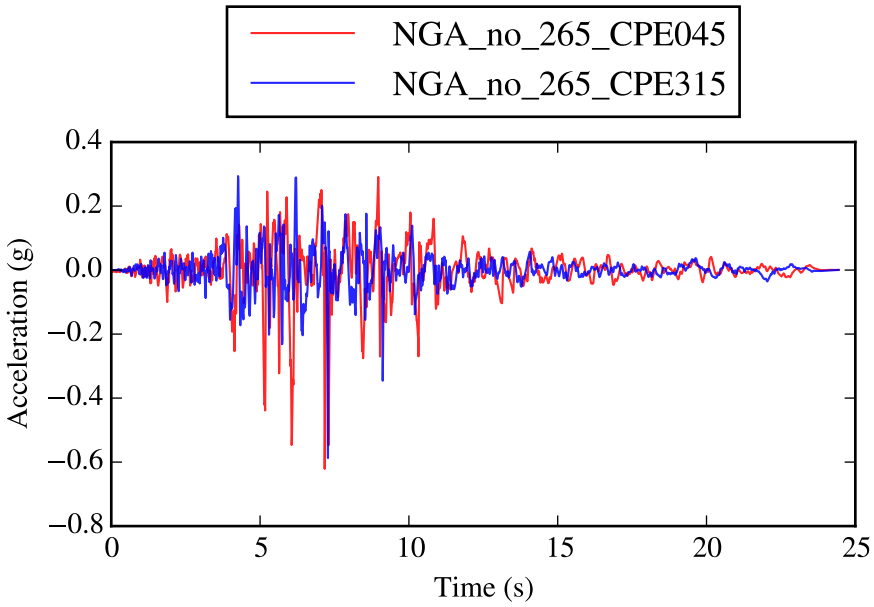


Figure F.91: WUS 91: *265_CPE045* and *265_CPE315*.

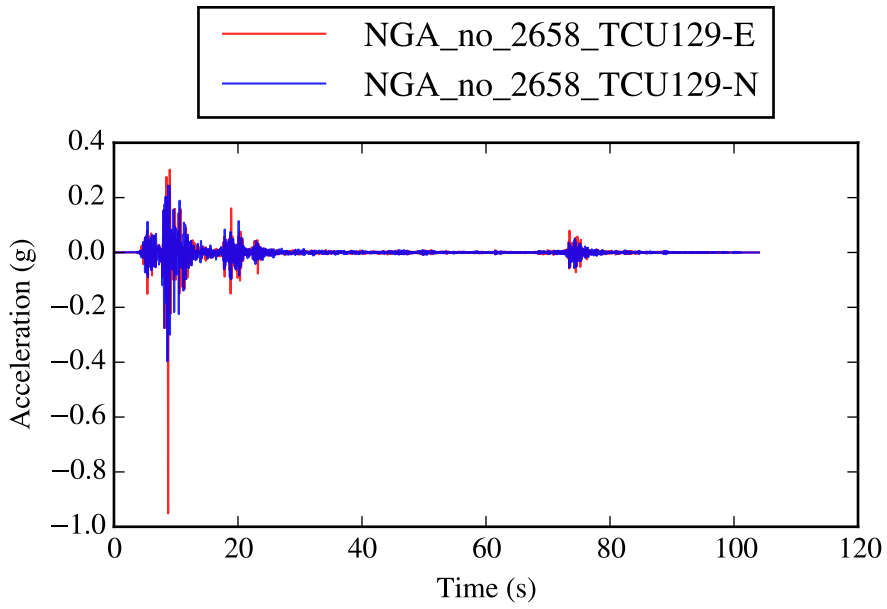


Figure F.92: WUS 92: *2658_TCU129-E* and *2658_TCU129-N*.

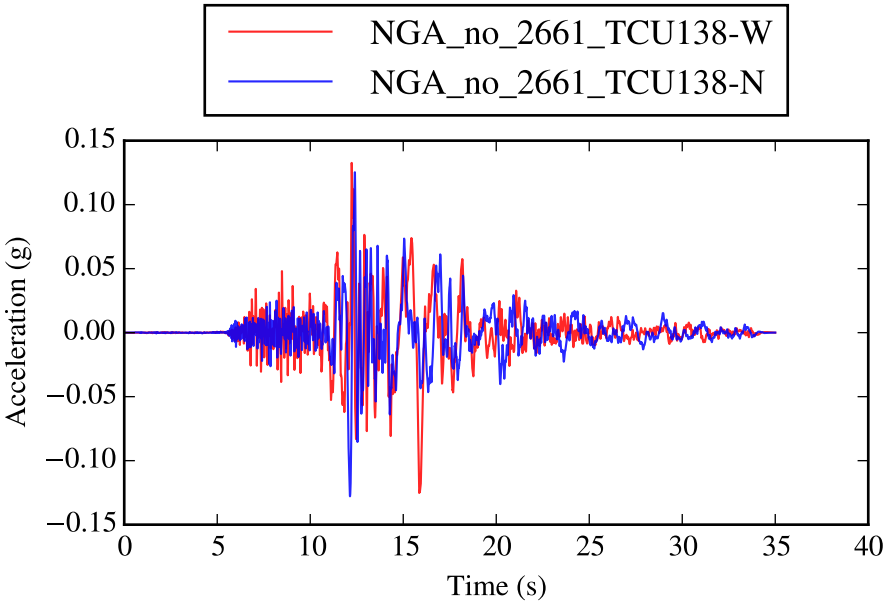


Figure F.93: WUS 93: *2661_TCU138-W* and *2661_TCU138-N*.

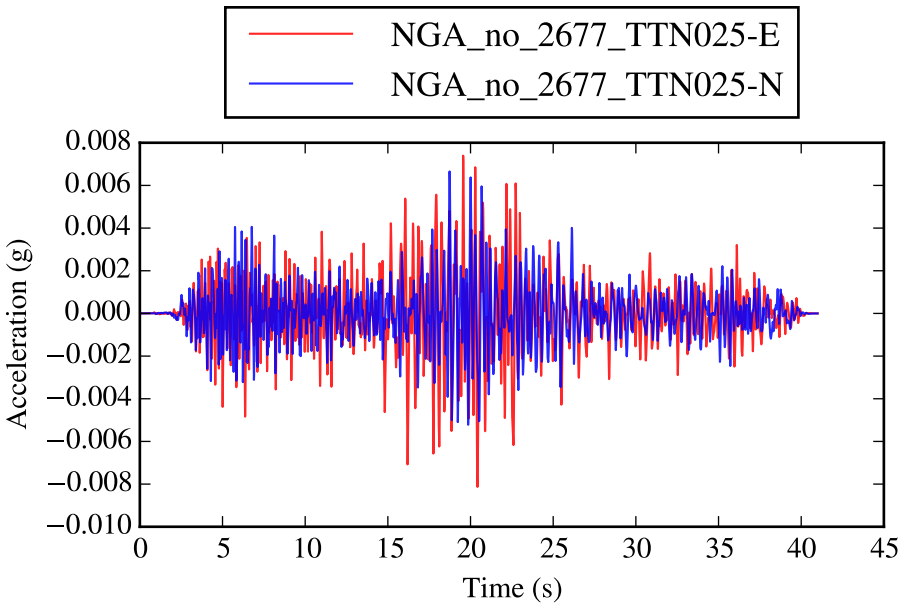


Figure F.94: WUS 94: *2677_TTN025-E* and *2677_TTN025-N*.

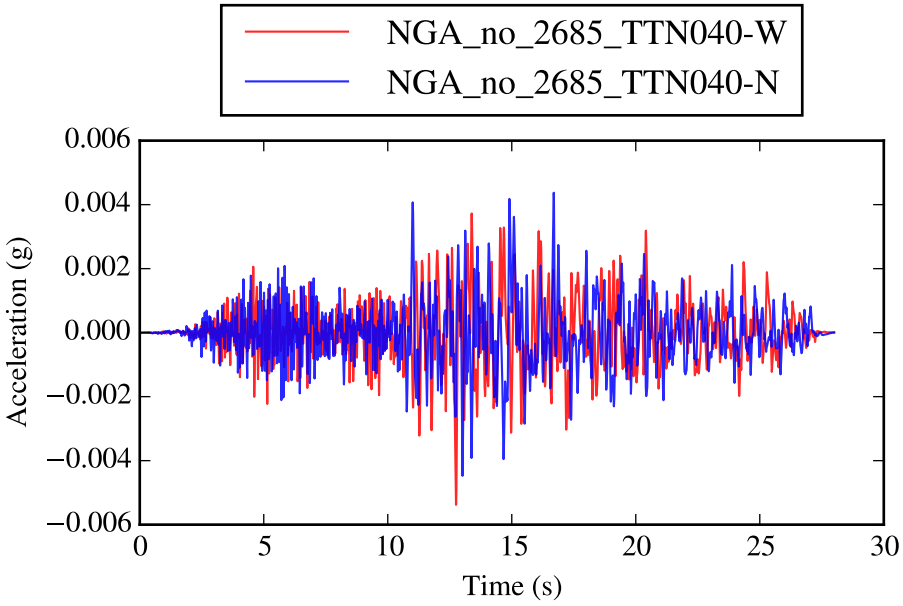


Figure F.95: WUS 95: *2685_TTN040-W* and *2685_TTN040-N*.

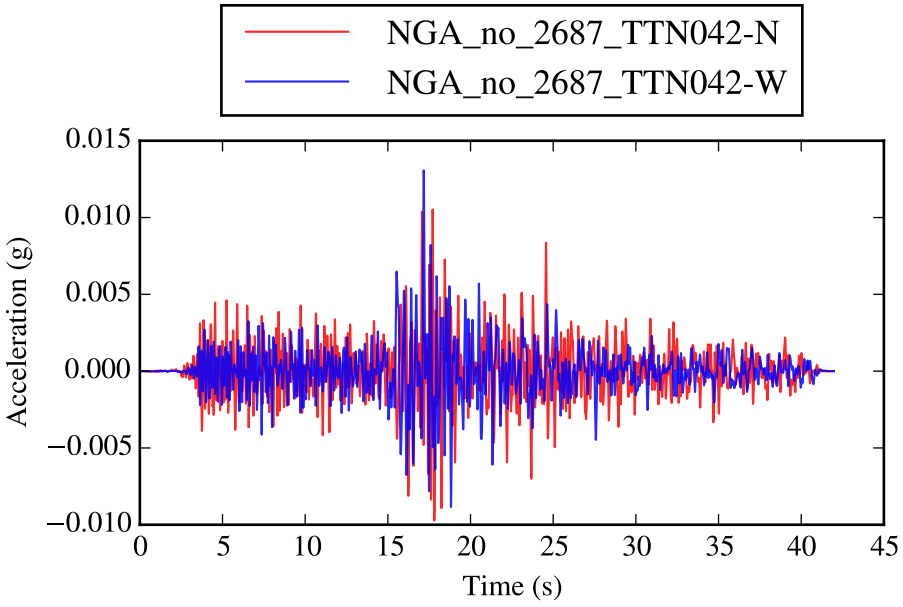


Figure F.96: WUS 96: *2687_TTN042-N* and *2687_TTN042-W*.

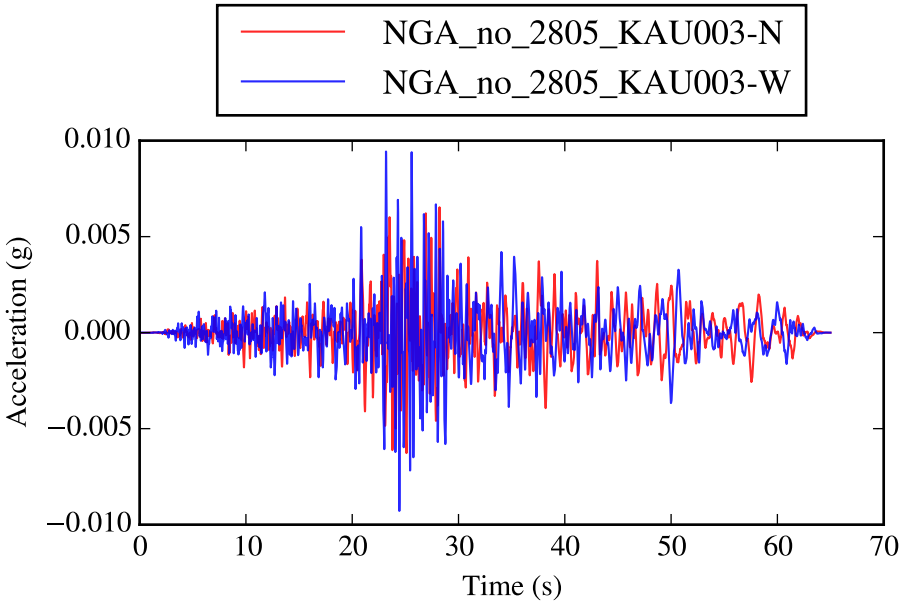


Figure F.97: WUS 97: *2805_KAU003-N* and *2805_KAU003-W*.

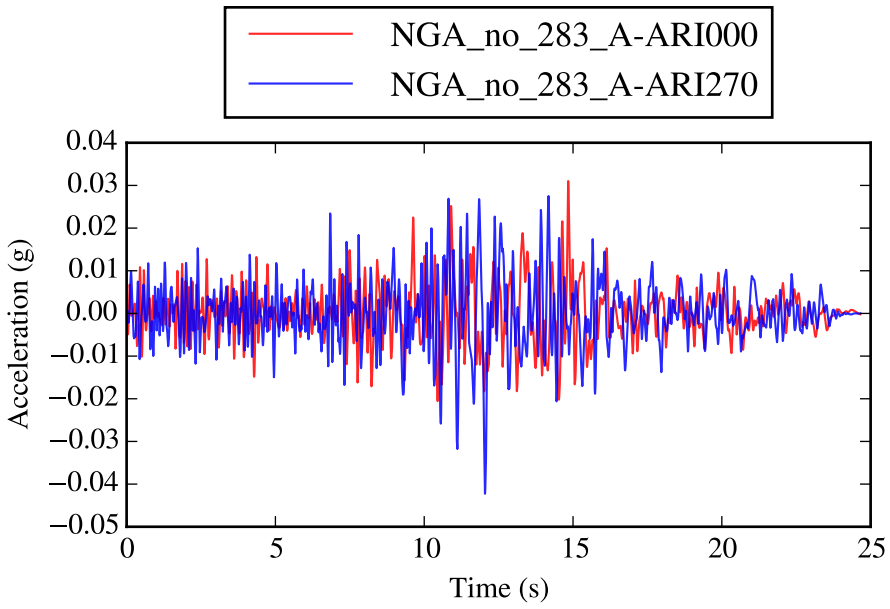


Figure F.98: WUS 98: *283_A-ARI000* and *283_A-ARI270*.

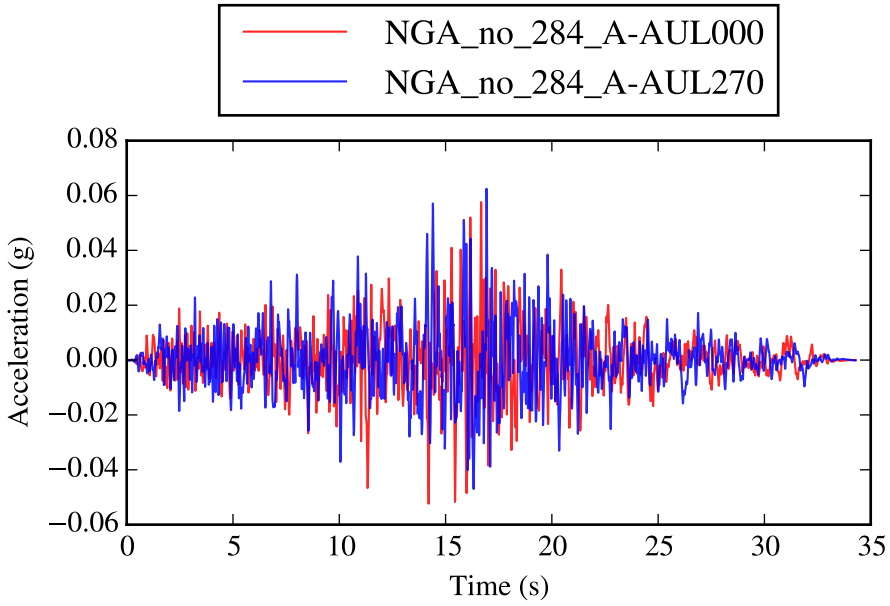


Figure F.99: WUS 99: *284_A-AUL000* and *284_A-AUL270*.

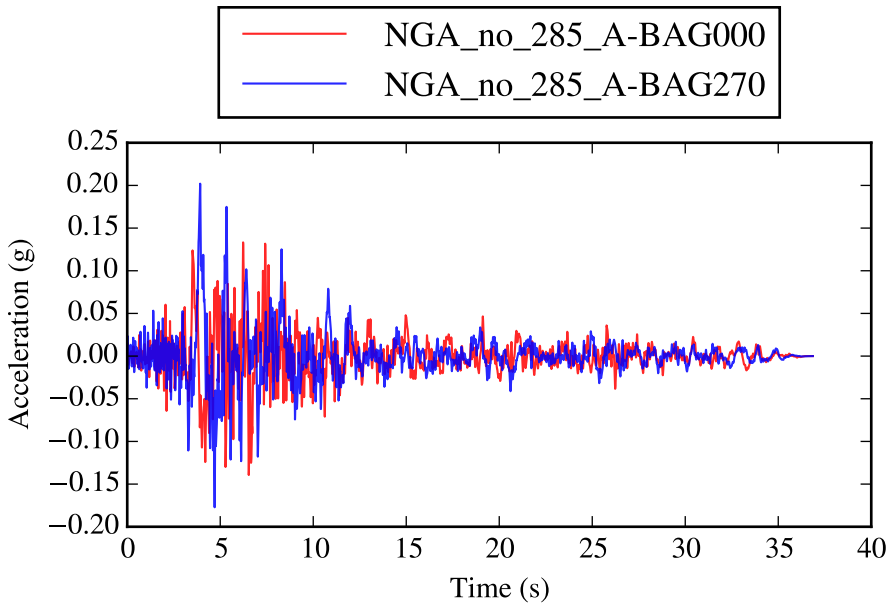


Figure F.100: WUS 100: *285_A-BAG000* and *285_A-BAG270*.

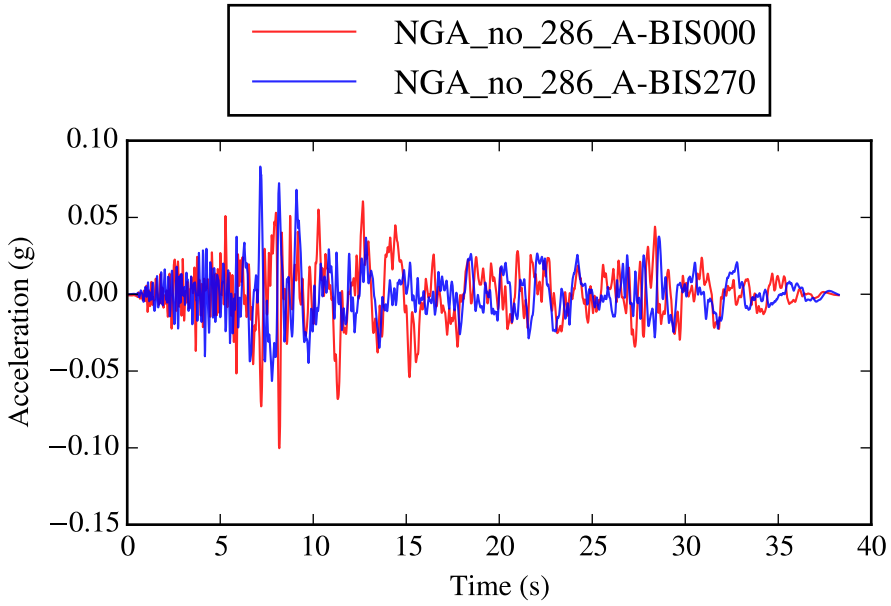


Figure F.101: WUS 101: *286_A-BIS000* and *286_A-BIS270*.

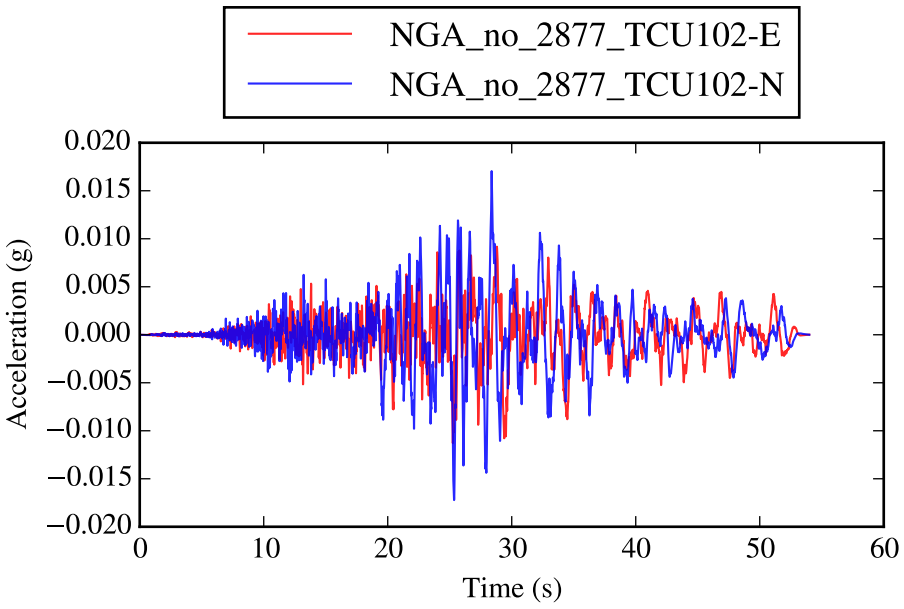


Figure F.102: WUS 102: *2877_TCU102-E* and *2877_TCU102-N*.

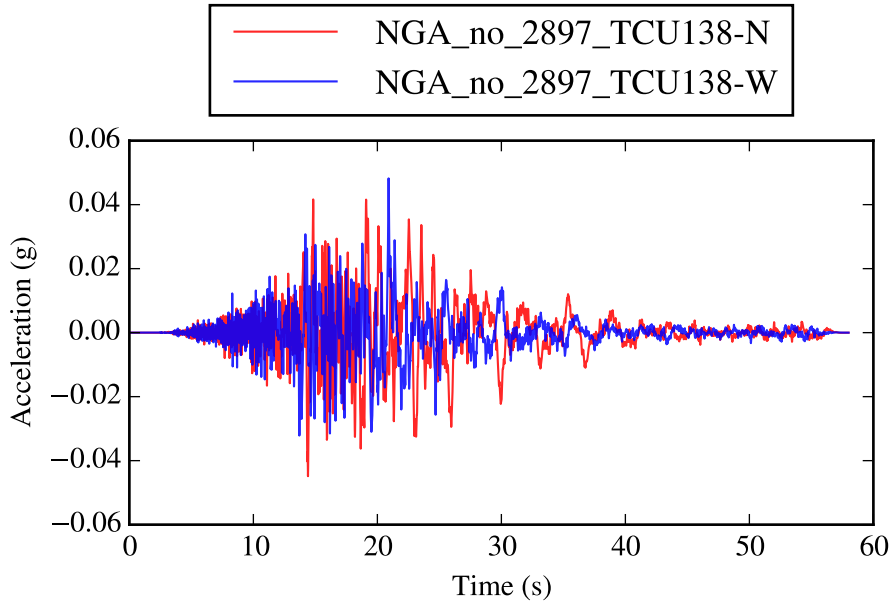


Figure F.103: WUS 103: *2897_TCU138-N* and *2897_TCU138-W*.

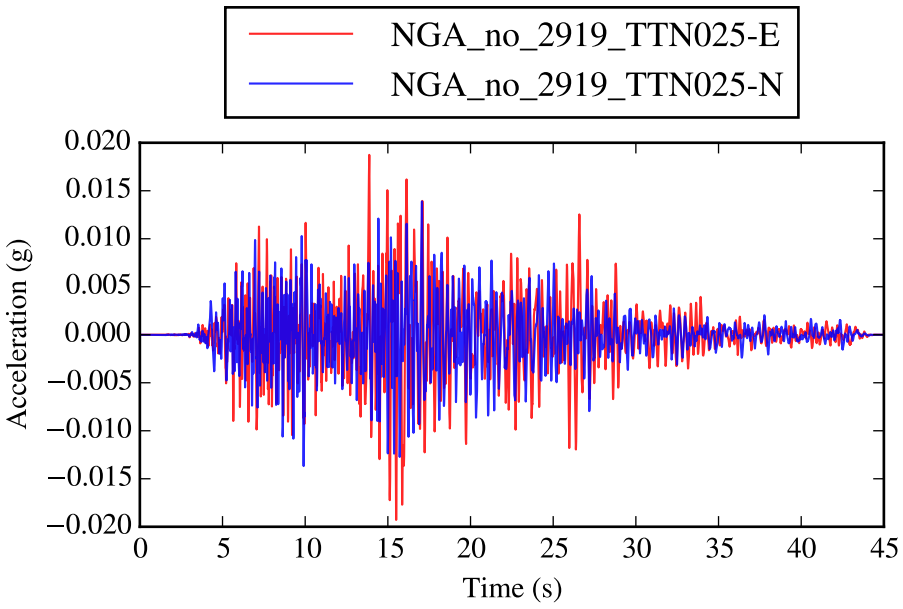


Figure F.104: WUS 104: *2919_TTN025-E* and *2919_TTN025-N*.

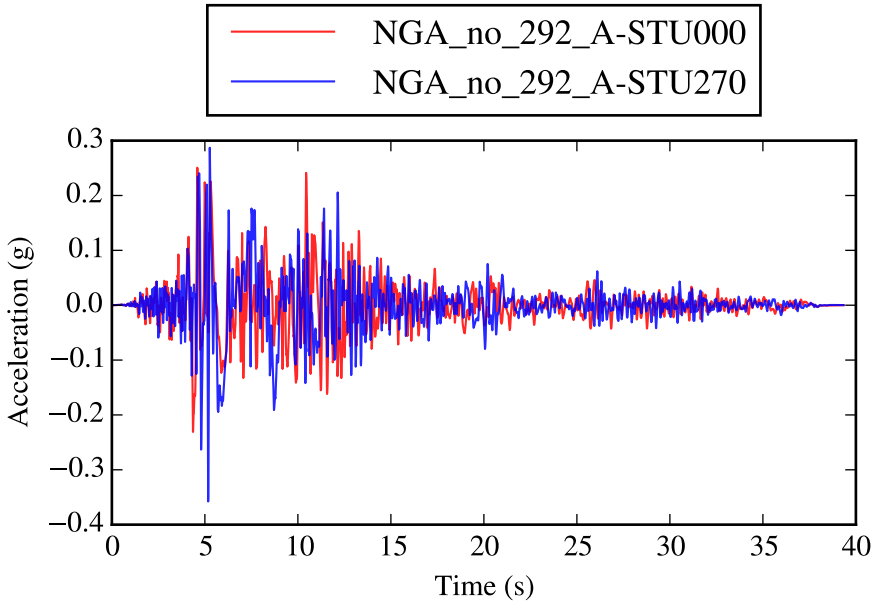


Figure F.105: WUS 105: *292_A-STU000* and *292_A-STU270*.

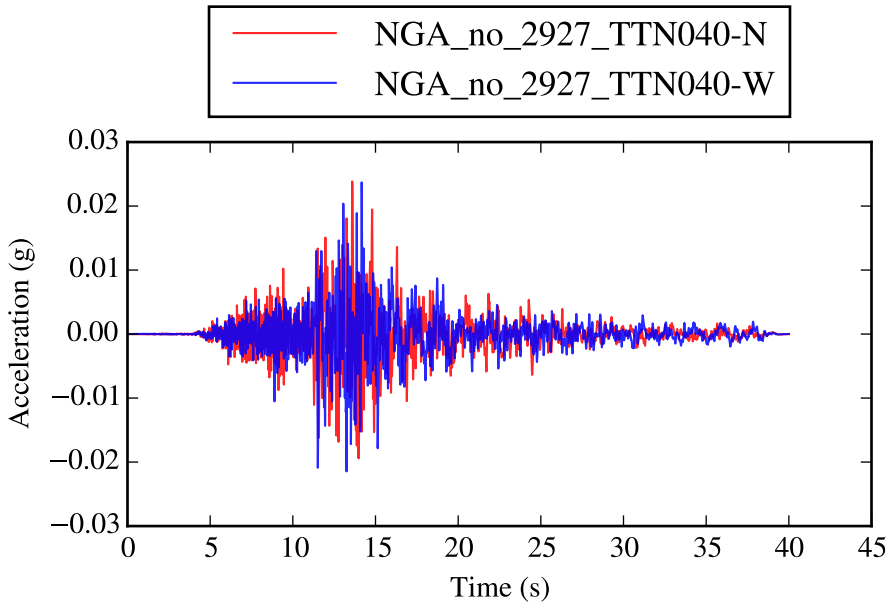


Figure F.106: WUS 106: *2927_TTN040-N* and *2927_TTN040-W*.

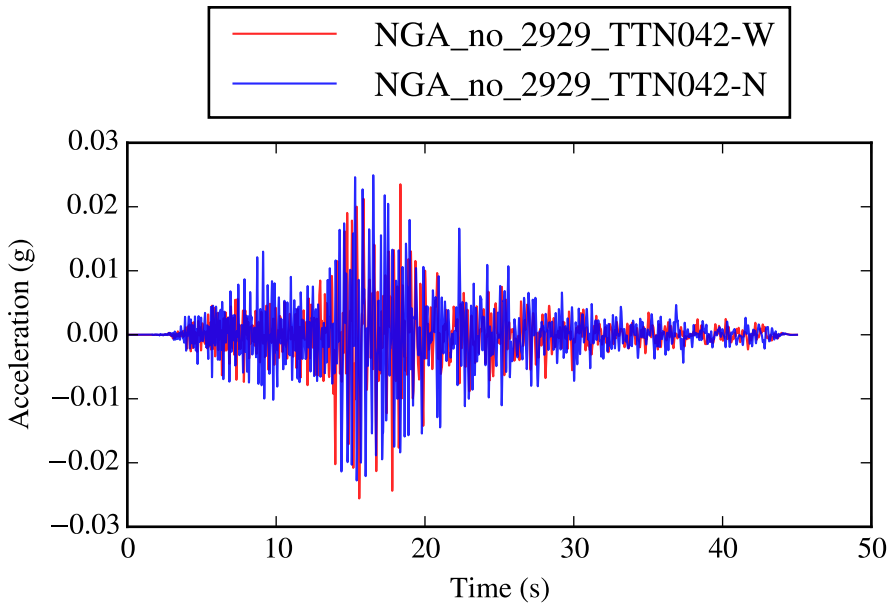


Figure F.107: WUS 107: *2929_TTN042-W* and *2929_TTN042-N*.

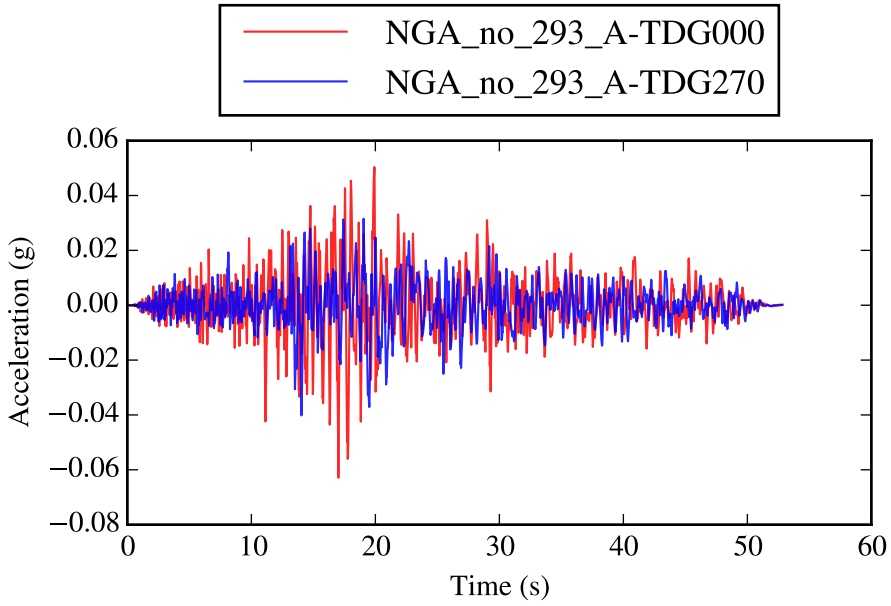


Figure F.108: WUS 108: *293_A-TDG000* and *293_A-TDG270*.

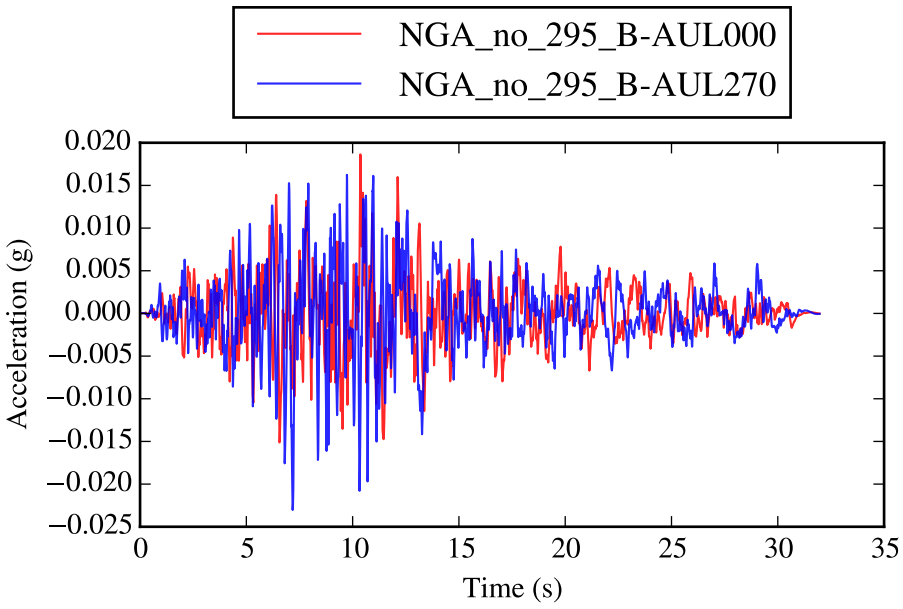


Figure F.109: WUS 109: *295_B-AUL000* and *295_B-AUL270*.

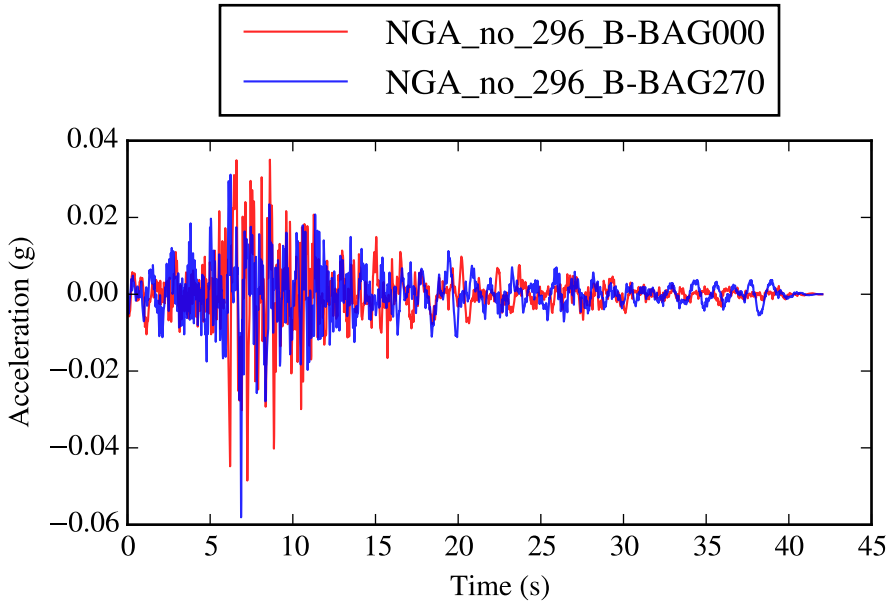


Figure F.110: WUS 110: *296_B-BAG000* and *296_B-BAG270*.

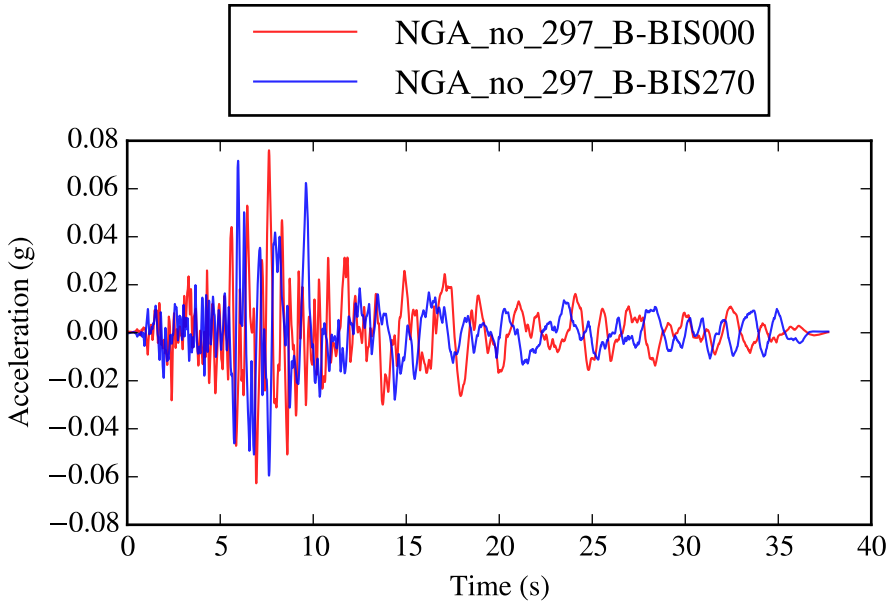


Figure F.111: WUS 111: *297_B-BIS000* and *297_B-BIS270*.

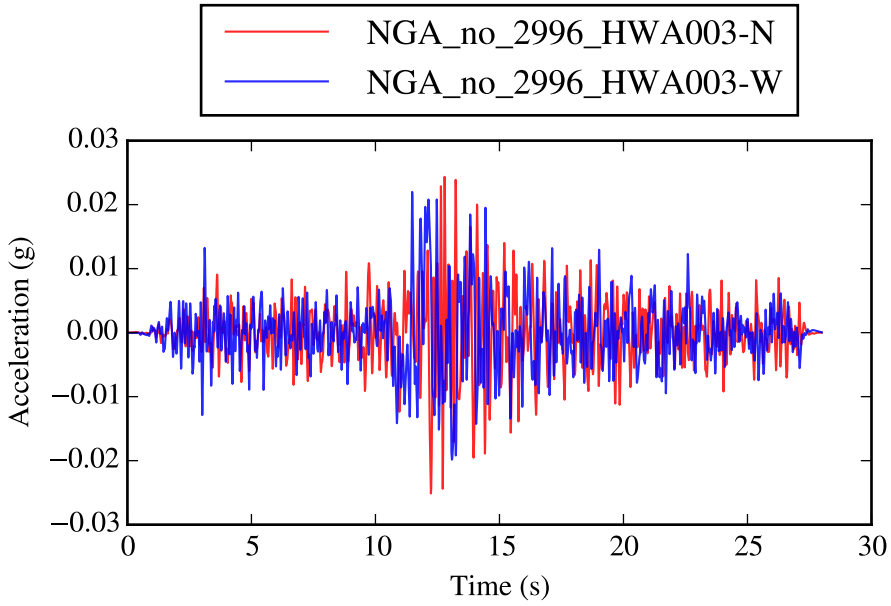


Figure F.112: WUS 112: *2996_HWA003-N* and *2996_HWA003-W*.

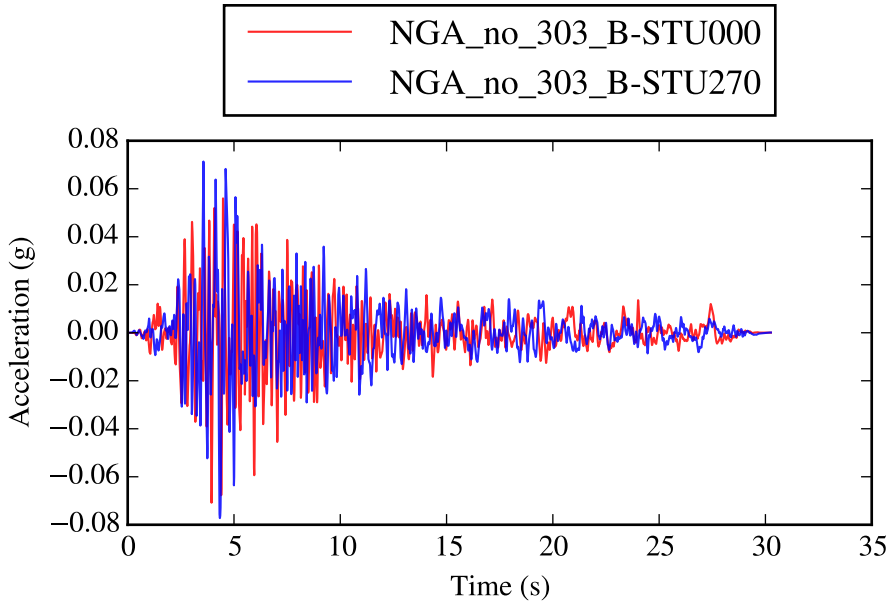


Figure F.113: WUS 113: *303_B-STU000* and *303_B-STU270*.

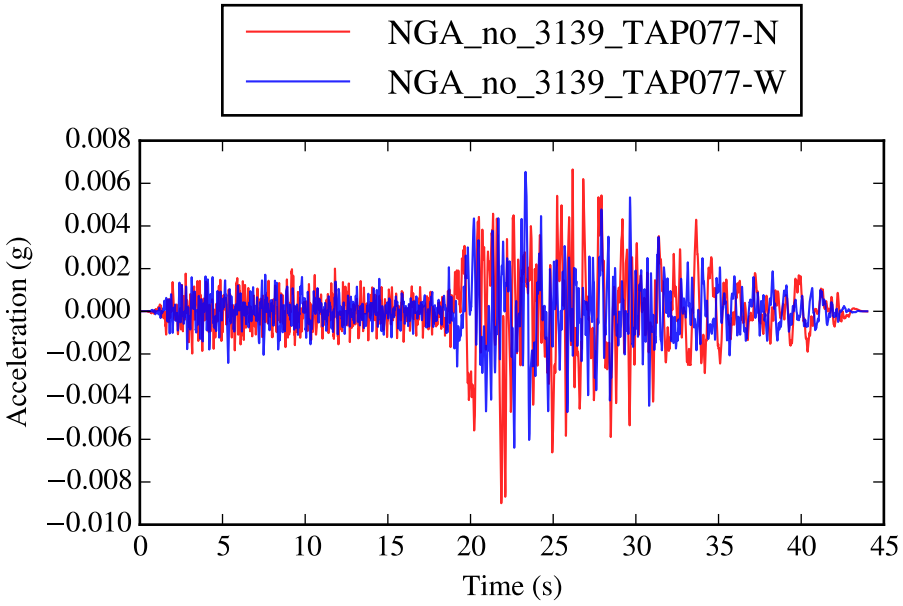


Figure F.114: WUS 114: *3139_TAP077-N* and *3139_TAP077-W*.

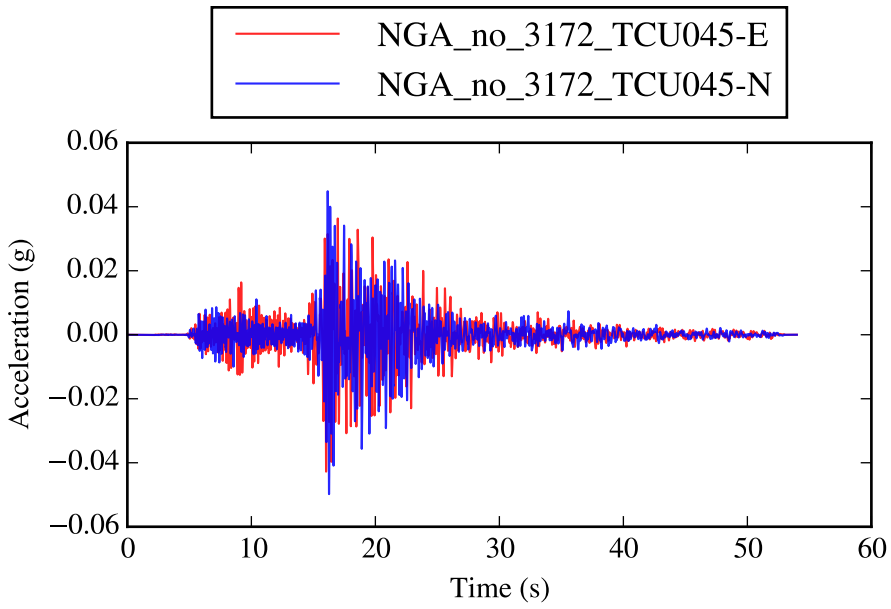


Figure F.115: WUS 115: *3172_TCU045-E* and *3172_TCU045-N*.

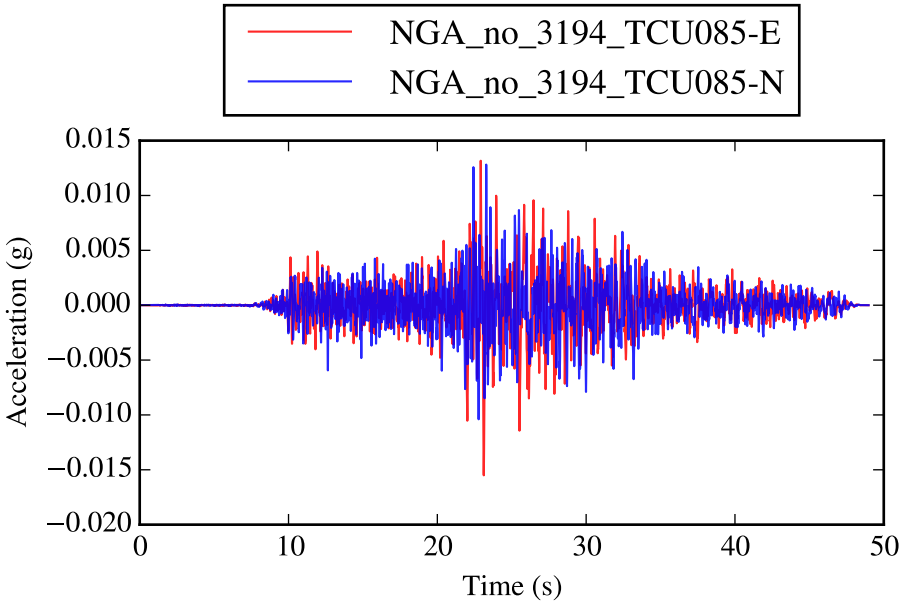


Figure F.116: WUS 116: *3194_TCU085-E* and *3194_TCU085-N*.

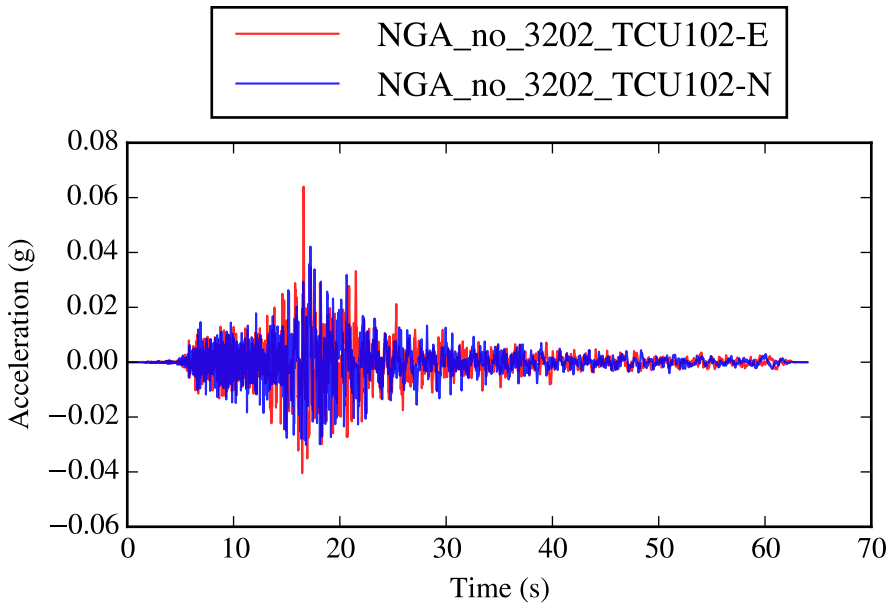


Figure F.117: WUS 117: *3202_TCU102-E* and *3202_TCU102-N*.

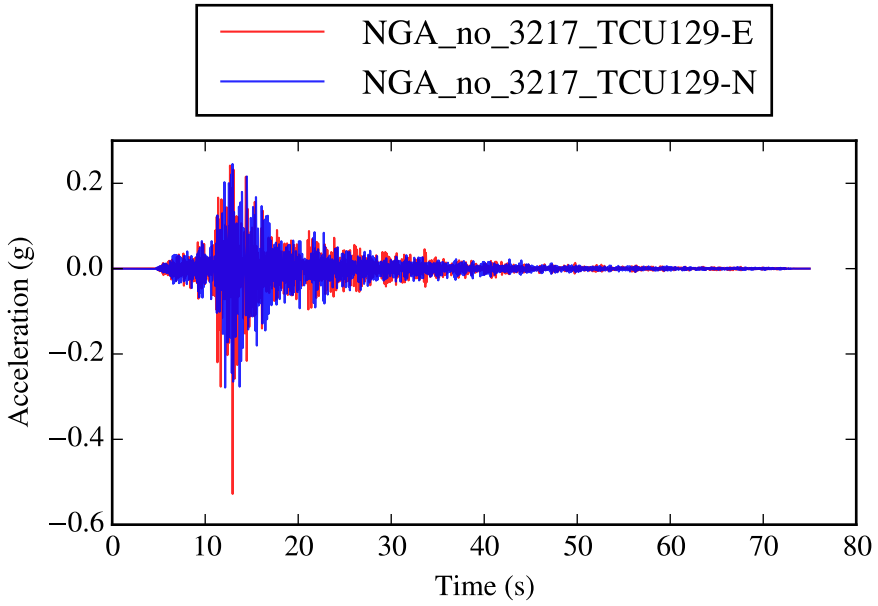


Figure F.118: WUS 118: *3217_TCU129-E* and *3217_TCU129-N*.

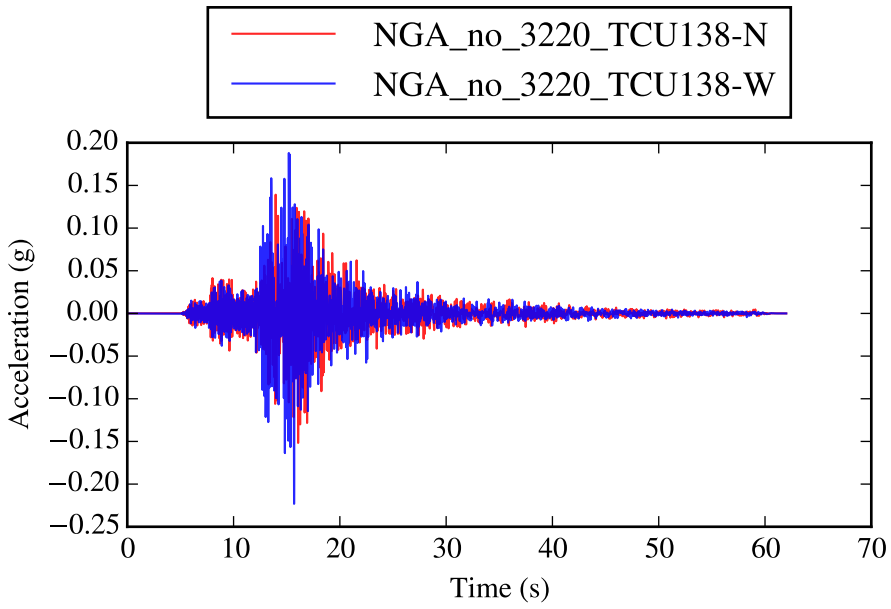


Figure F.119: WUS 119: *3220_TCU138-N* and *3220_TCU138-W*.

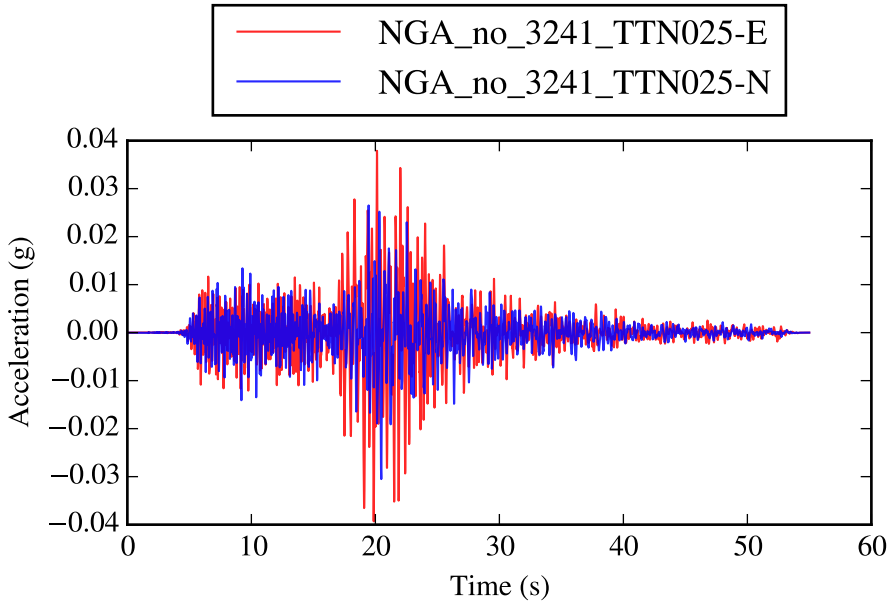


Figure F.120: WUS 120: *3241_TTN025-E* and *3241_TTN025-N*.

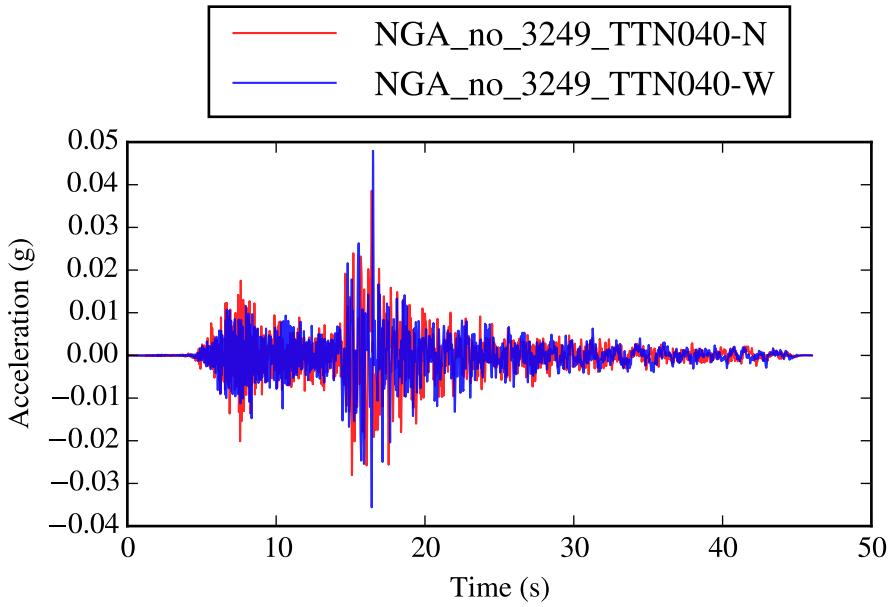


Figure F.121: WUS 121: *3249_TTN040-N* and *3249_TTN040-W*.

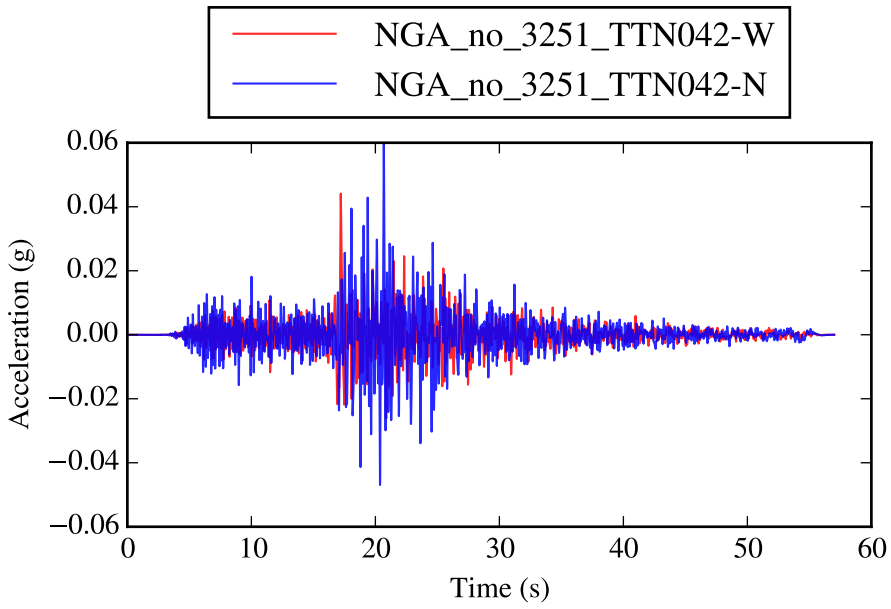


Figure F.122: WUS 122: *3251_TTN042-W* and *3251_TTN042-N*.

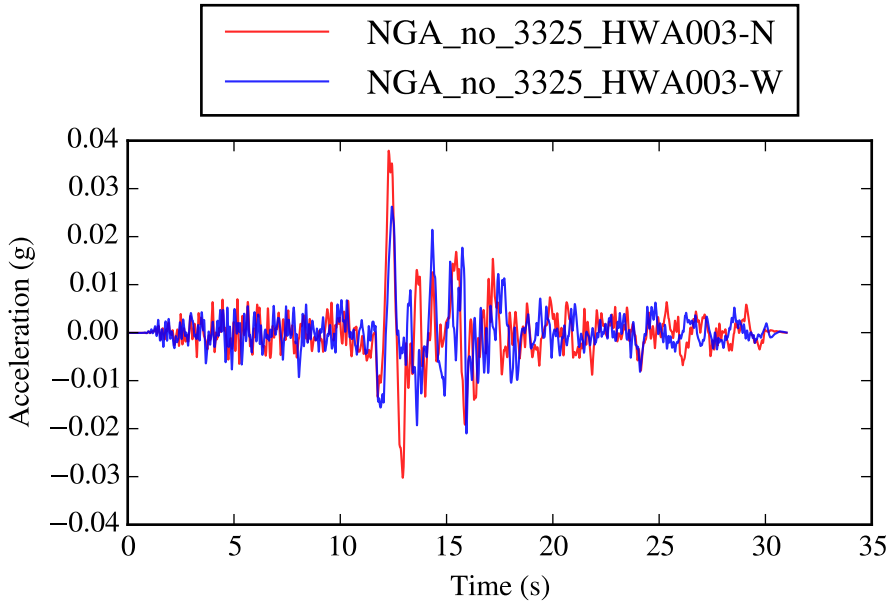


Figure F.123: WUS 123: *3325_HWA003-N* and *3325_HWA003-W*.

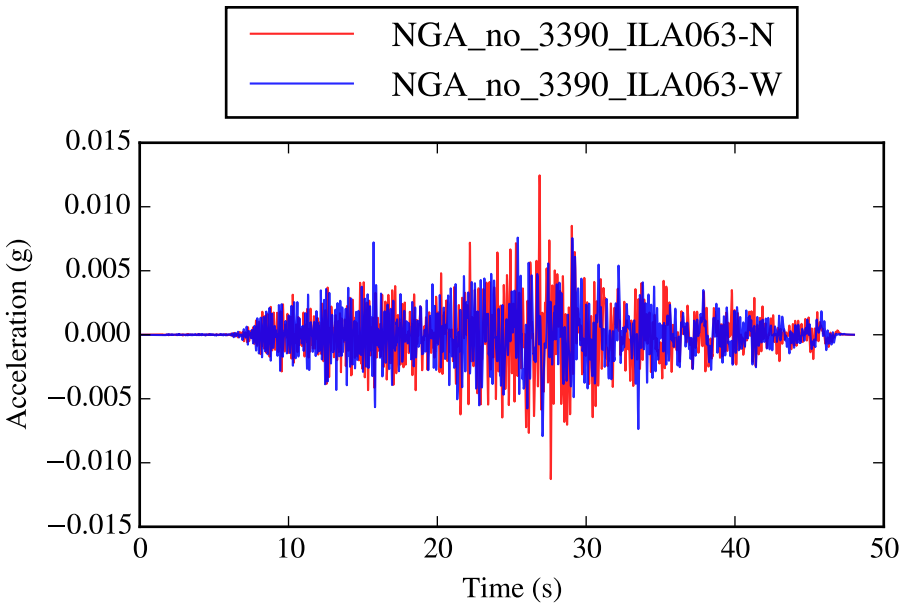


Figure F.124: WUS 124: *3390_ILA063-N* and *3390_ILA063-W*.

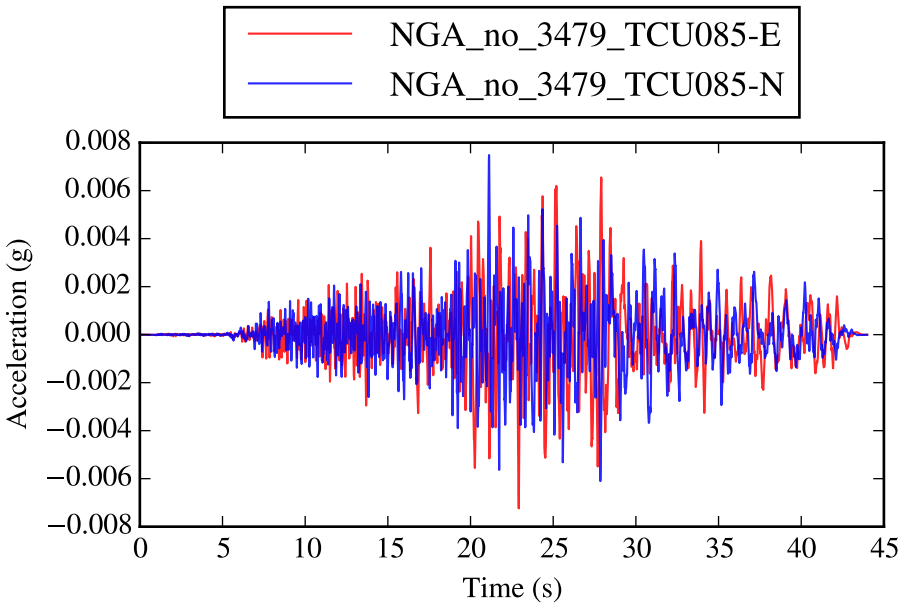


Figure F.125: WUS 125: *3479_TCU085-E* and *3479_TCU085-N*.

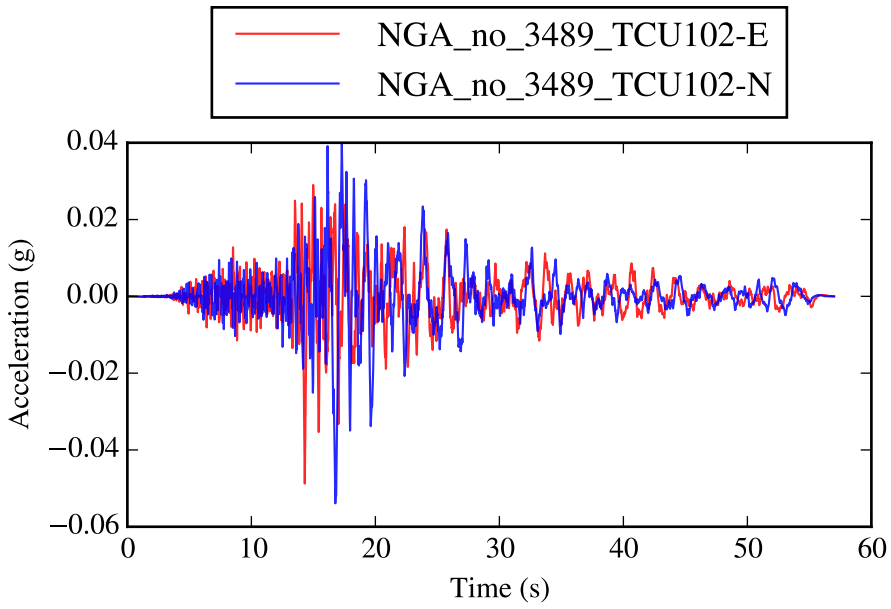


Figure F.126: WUS 126: *3489_TCU102-E* and *3489_TCU102-N*.

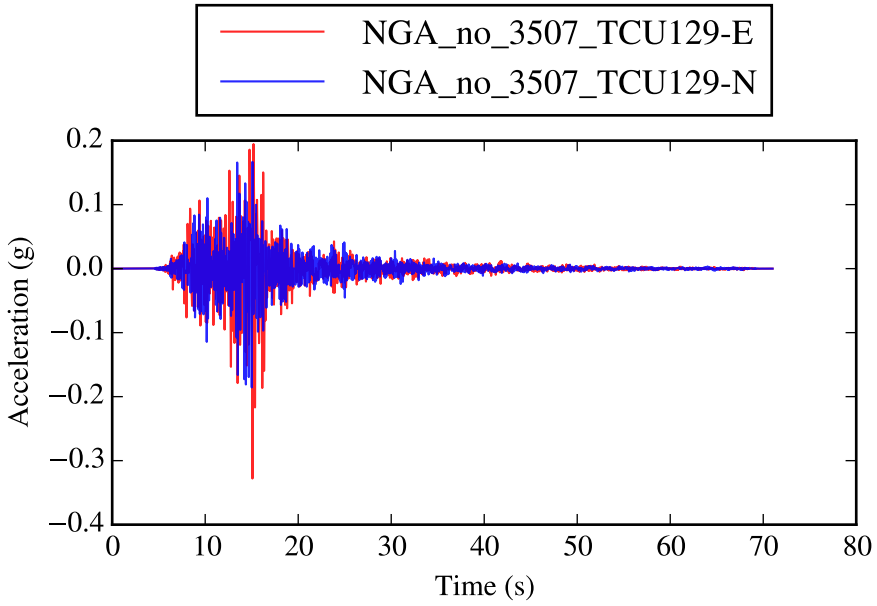


Figure F.127: WUS 127: *3507_TCU129-E* and *3507_TCU129-N*.

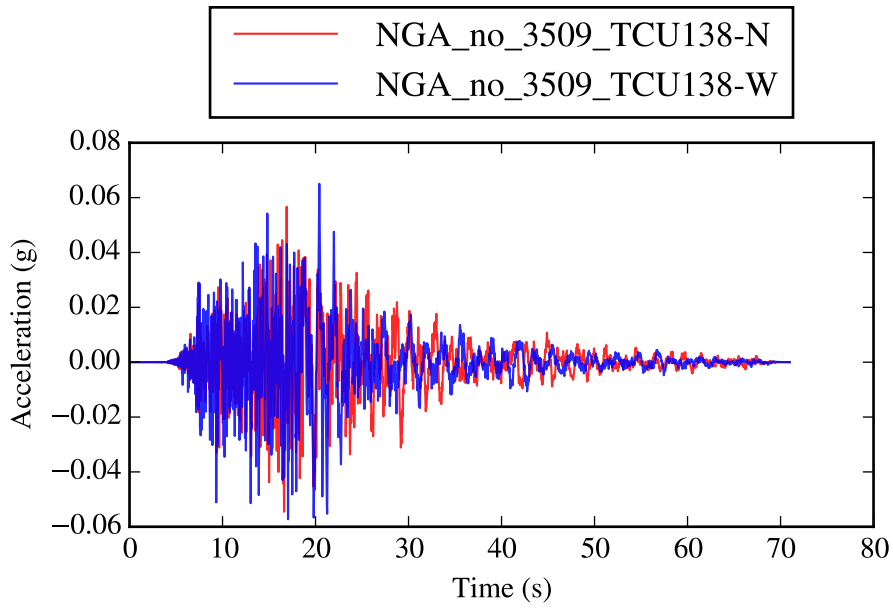


Figure F.128: WUS 128: *3509_TCU138-N* and *3509_TCU138-W*.

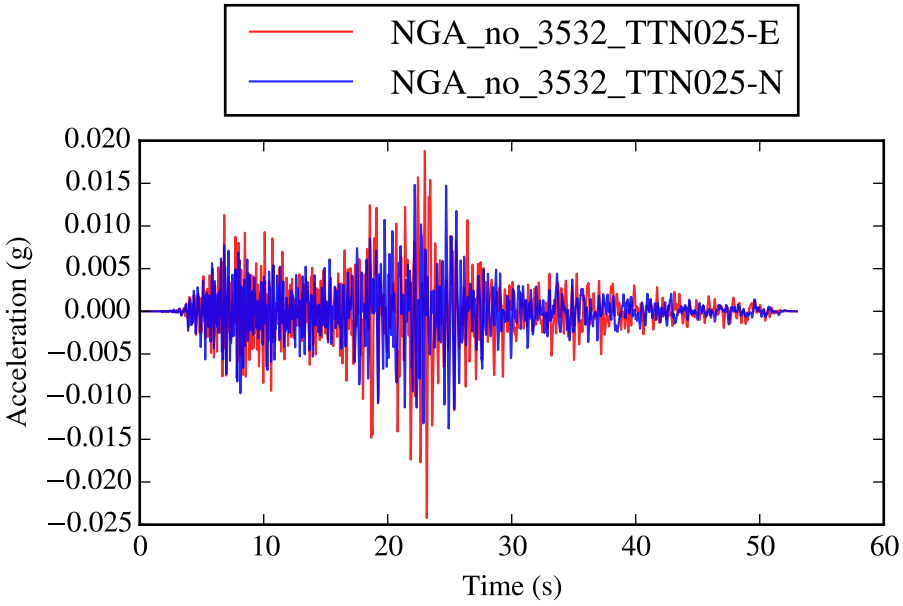


Figure F.129: WUS 129: *3532_TTN025-E* and *3532_TTN025-N*.

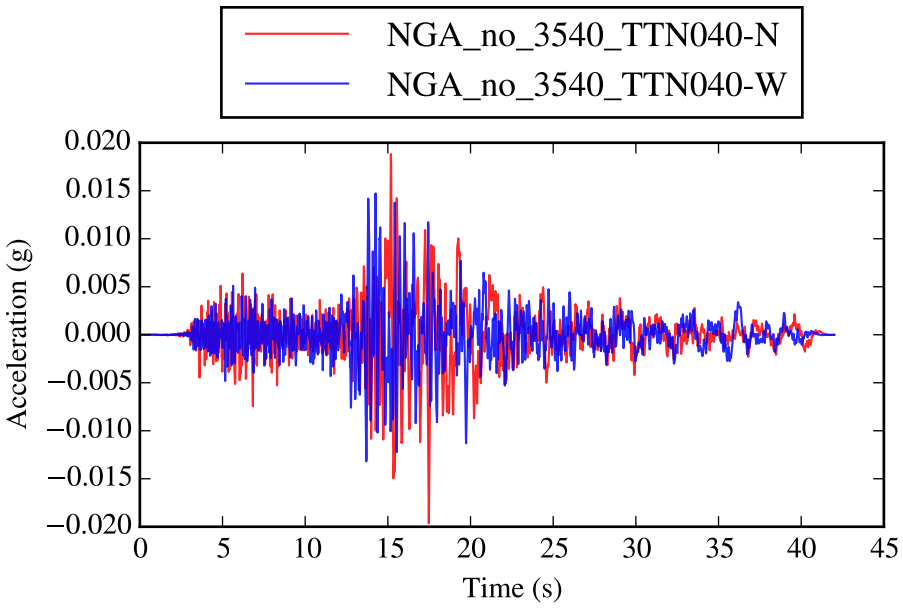


Figure F.130: WUS 130: *3540_TTN040-N* and *3540_TTN040-W*.

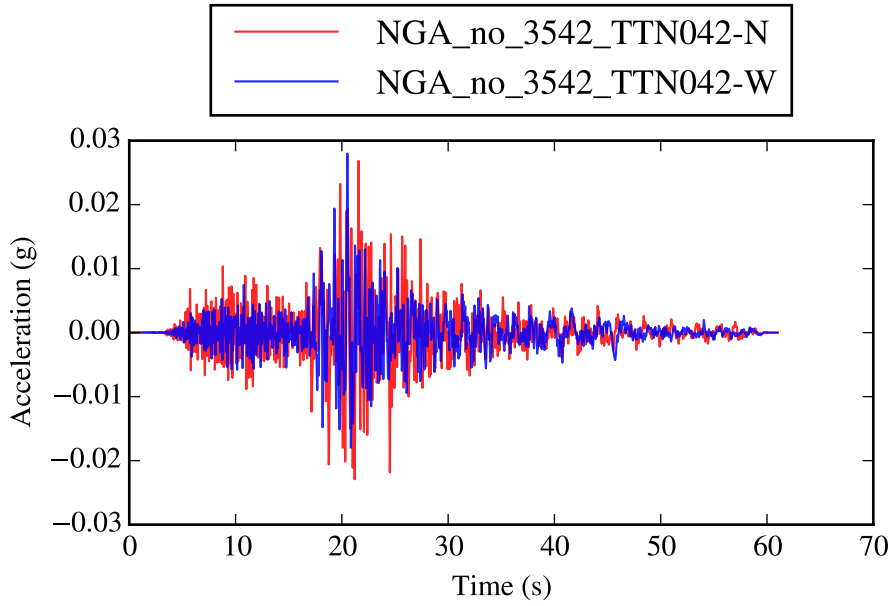


Figure F.131: WUS 131: *3542_TTN042-N* and *3542_TTN042-W*.

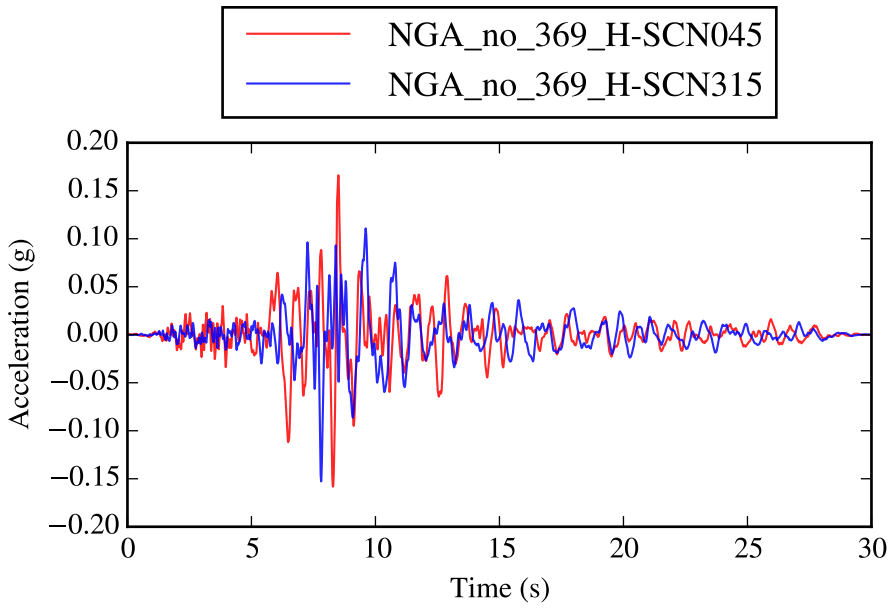


Figure F.132: WUS 132: *369_H-SCN045* and *369_H-SCN315*.

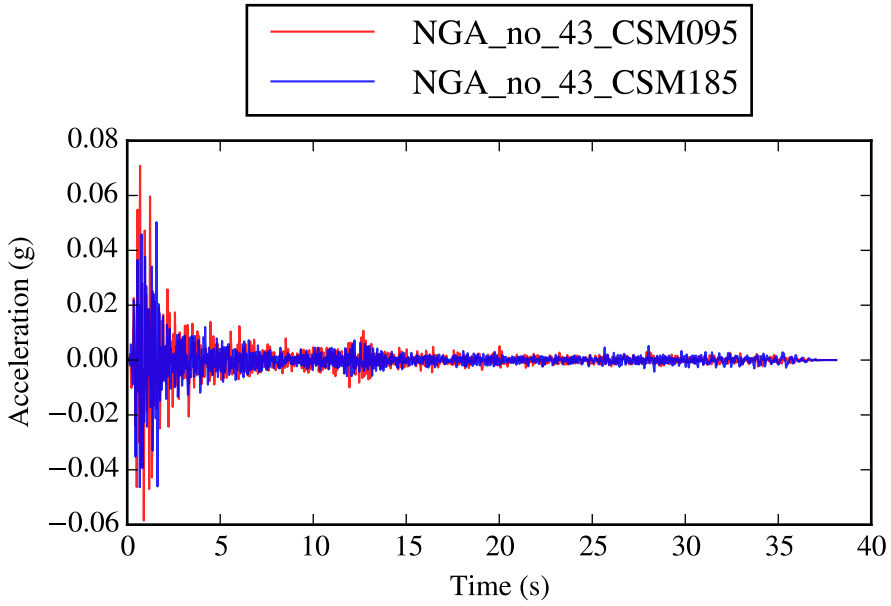


Figure F.133: WUS 133: *43_CSM095* and *43_CSM185*.

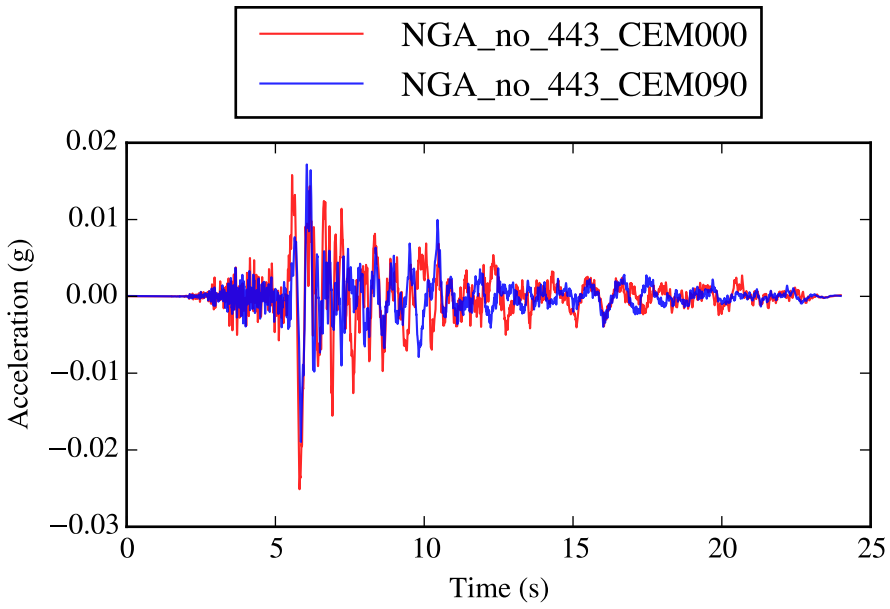


Figure F.134: WUS 134: *443_CEM000* and *443_CEM090*.

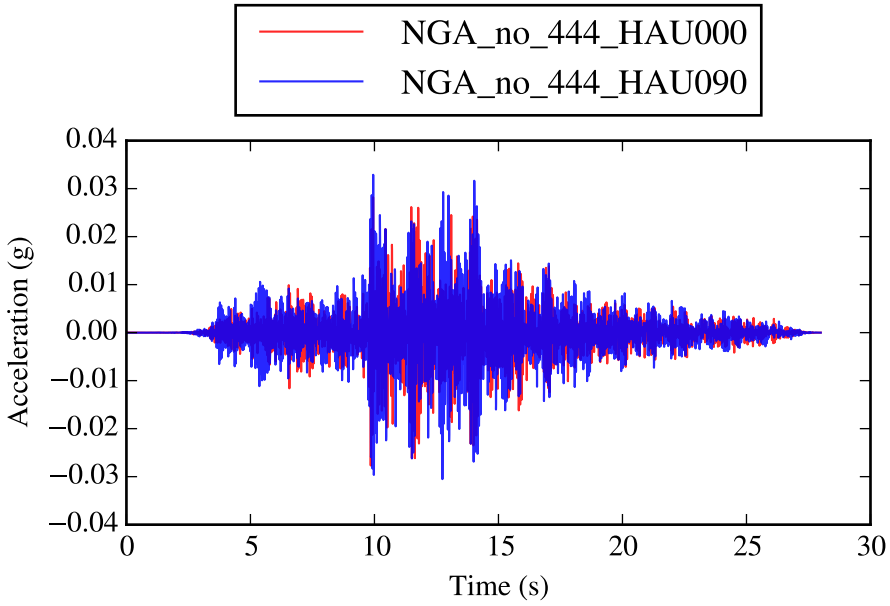


Figure F.135: WUS 135: *444_HAU000* and *444_HAU090*.

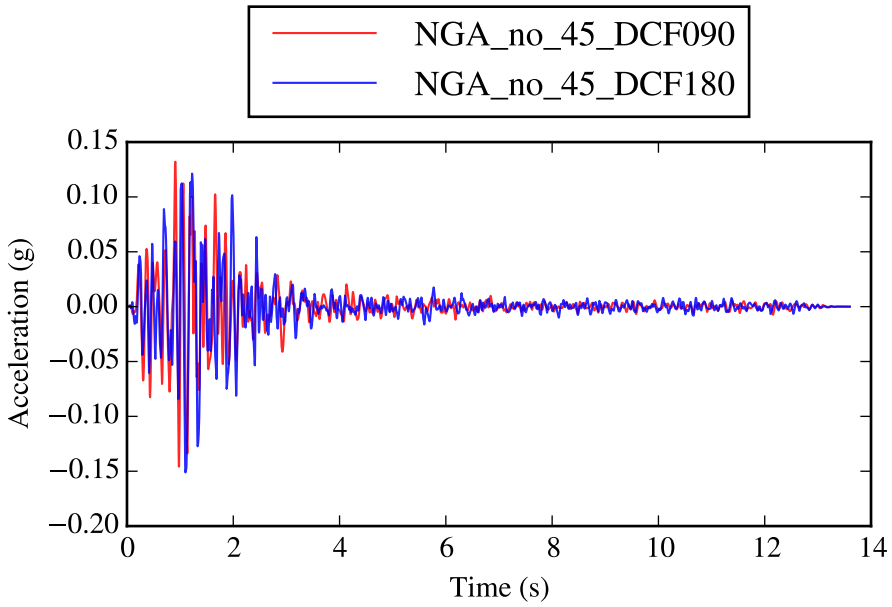


Figure F.136: WUS 136: *45_DCF090* and *45_DCF180*.

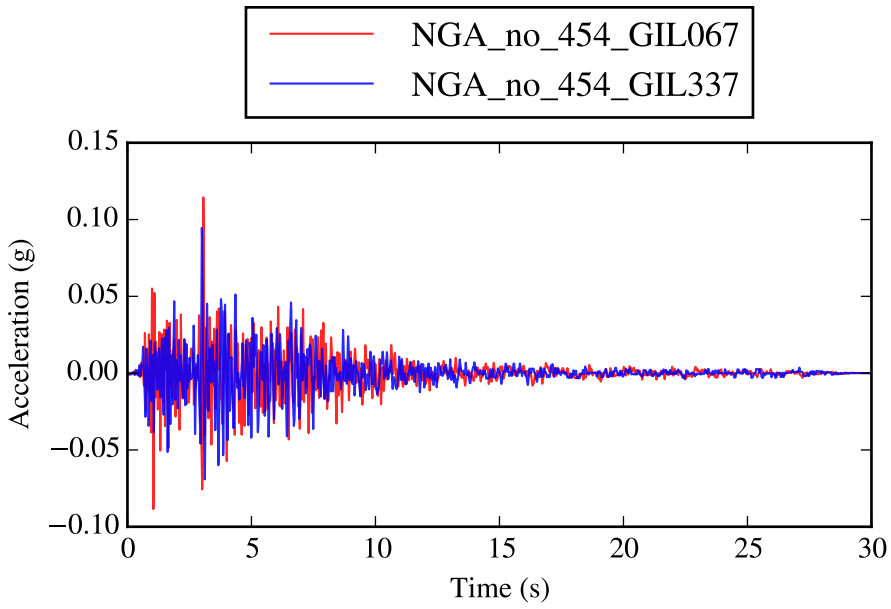


Figure F.137: WUS 137: *454_GIL067* and *454_GIL337*.

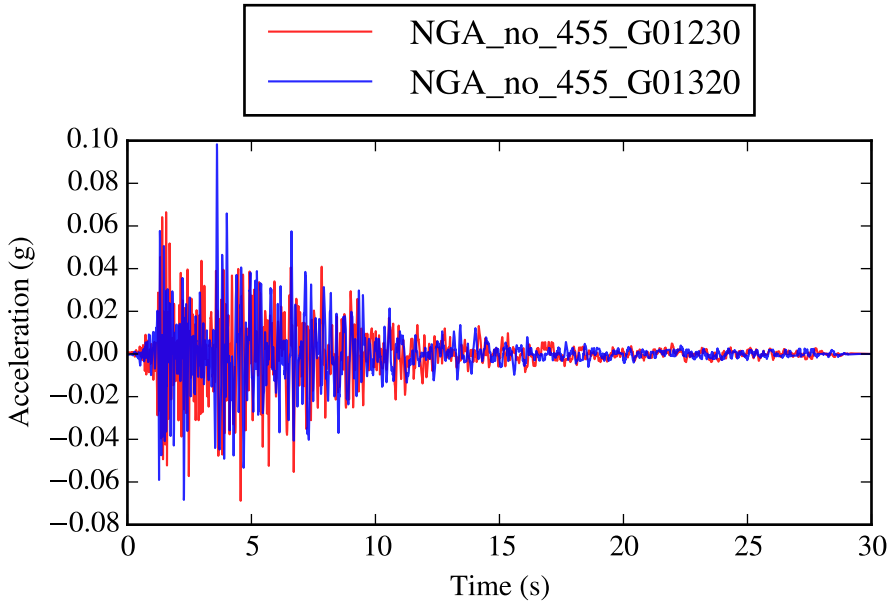


Figure F.138: WUS 138: *455_G01230* and *455_G01320*.

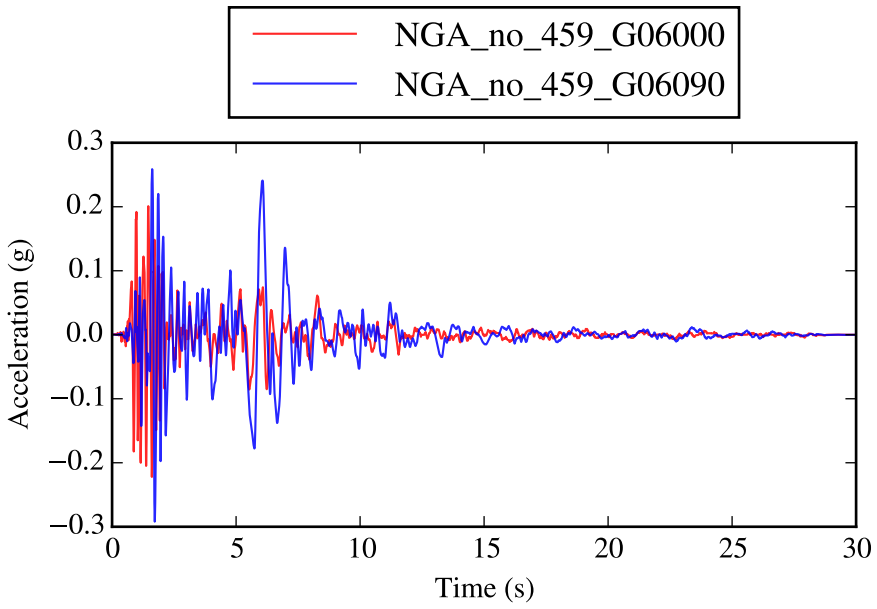


Figure F.139: WUS 139: *459_G06000* and *459_G06090*.

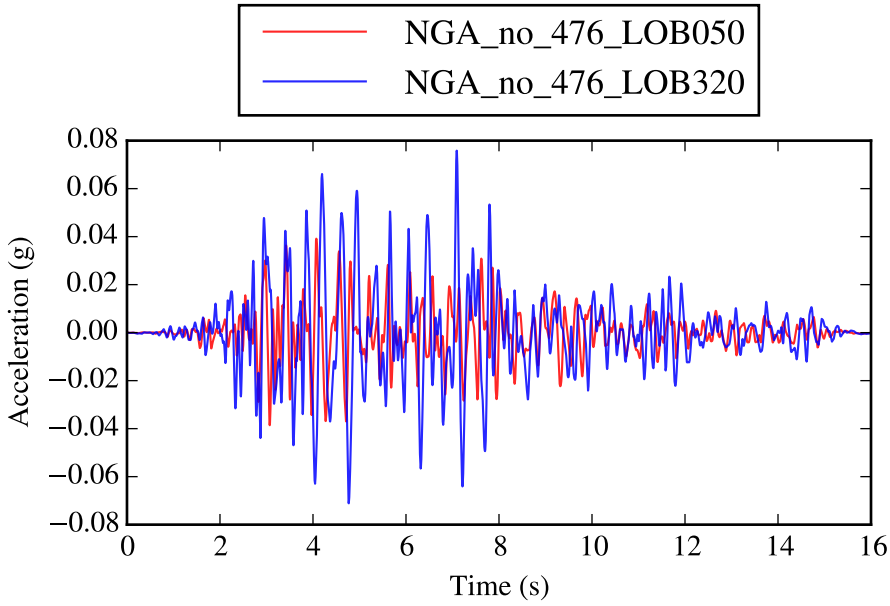


Figure F.140: WUS 140: *476_LOB050* and *476_LOB320*.

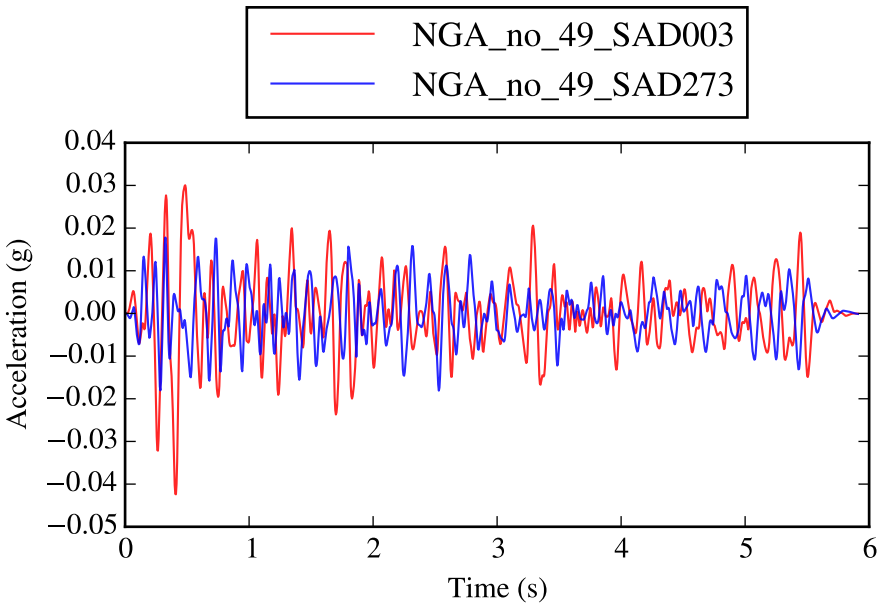


Figure F.141: WUS 141: *49_SAD003* and *49_SAD273*.

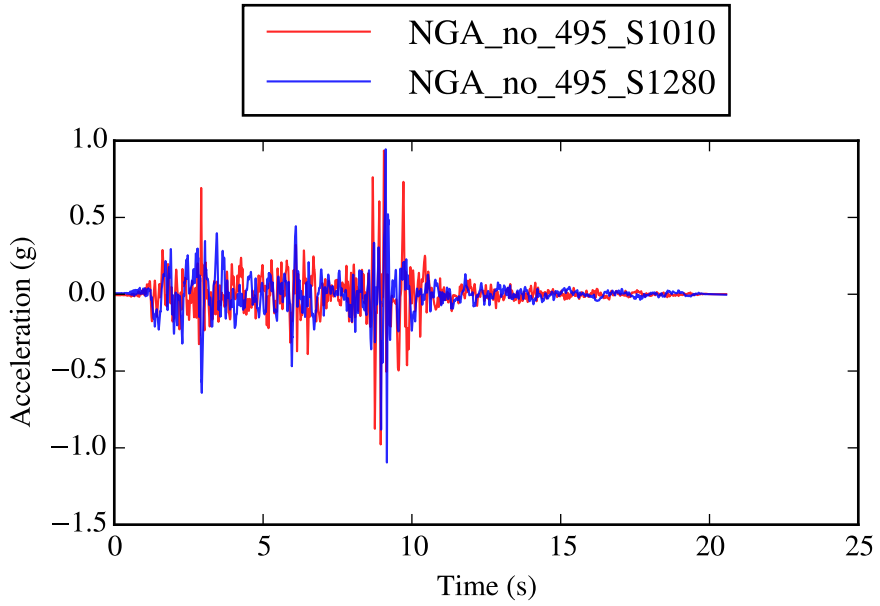


Figure F.142: WUS 142: *495_S1010* and *495_S1280*.

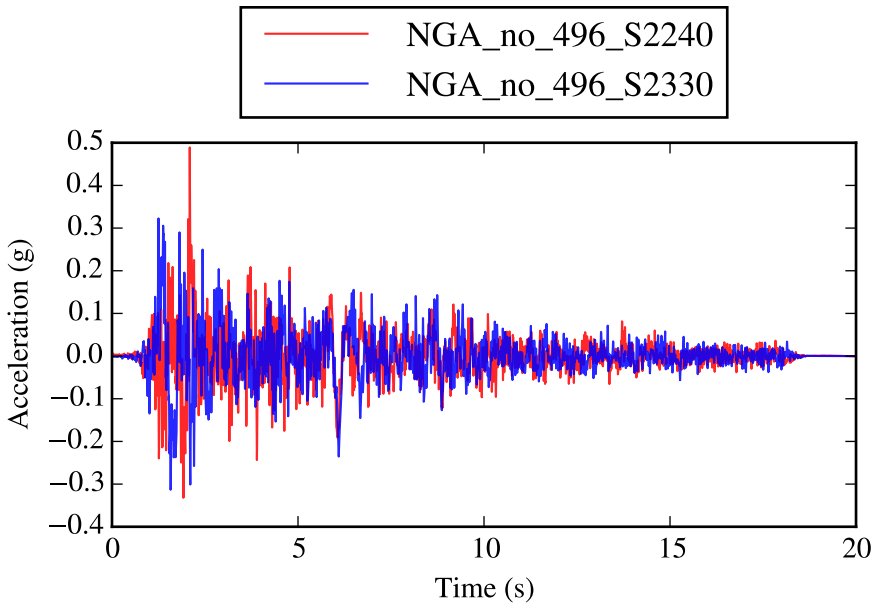


Figure F.143: WUS 143: *496_S2240* and *496_S2330*.

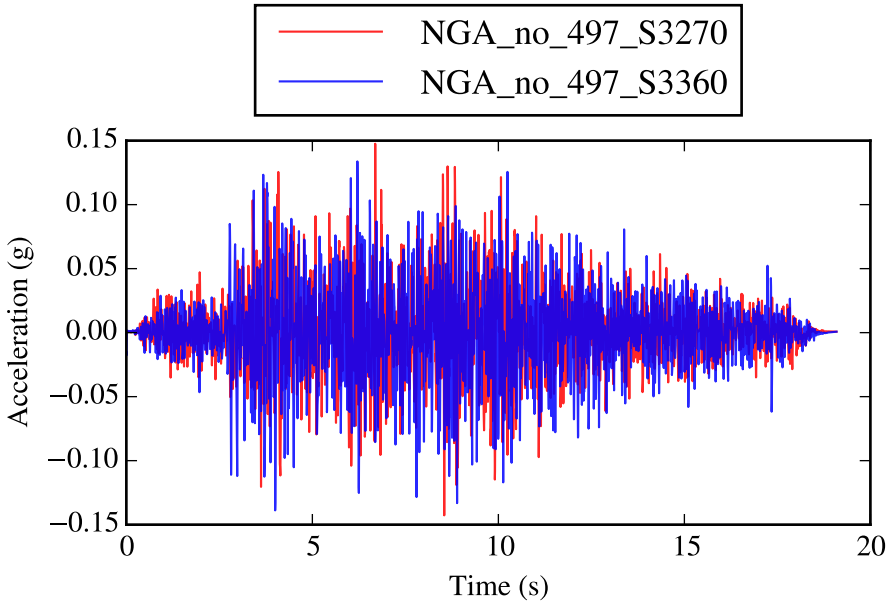


Figure F.144: WUS 144: *497_S3270* and *497_S3360*.

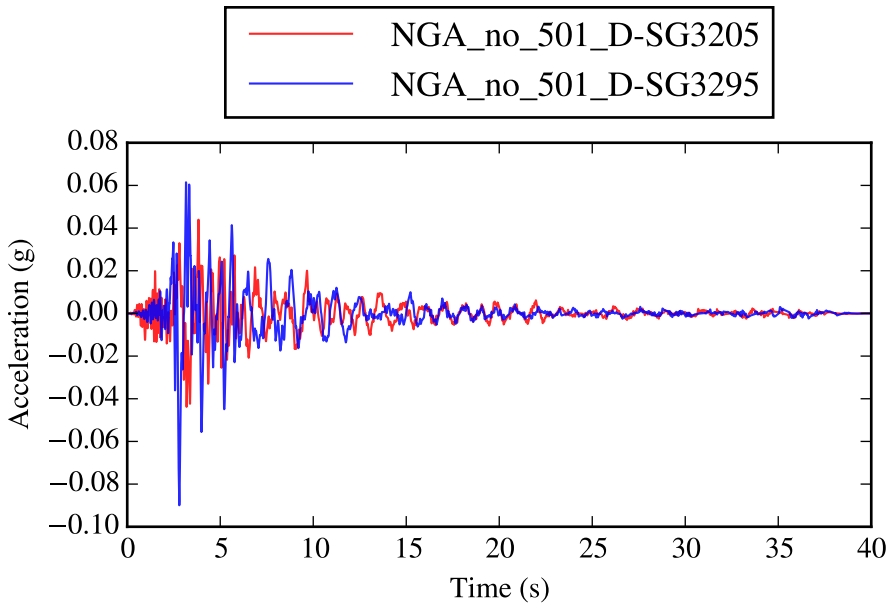


Figure F.145: WUS 145: *501_D-SG3205* and *501_D-SG3295*.

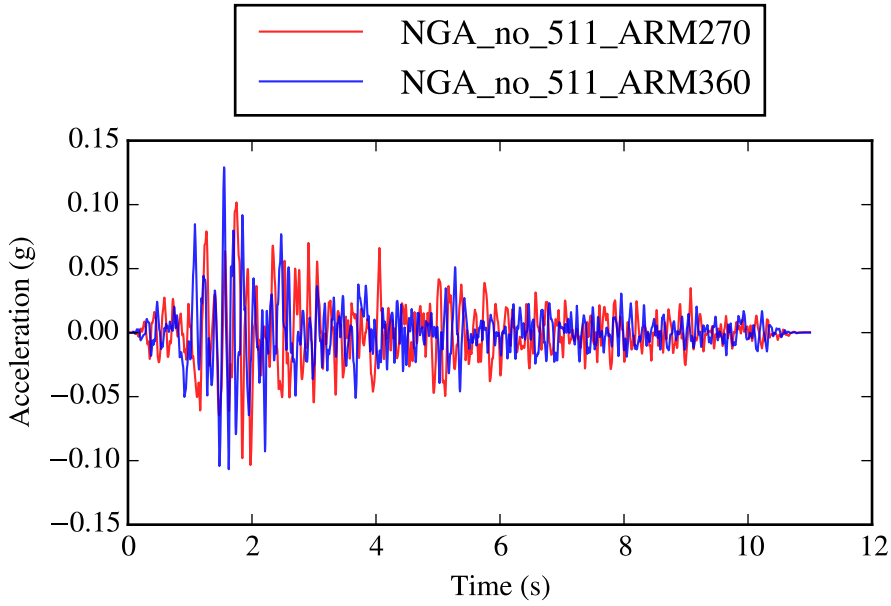


Figure F.146: WUS 146: *511_ARM270* and *511_ARM360*.

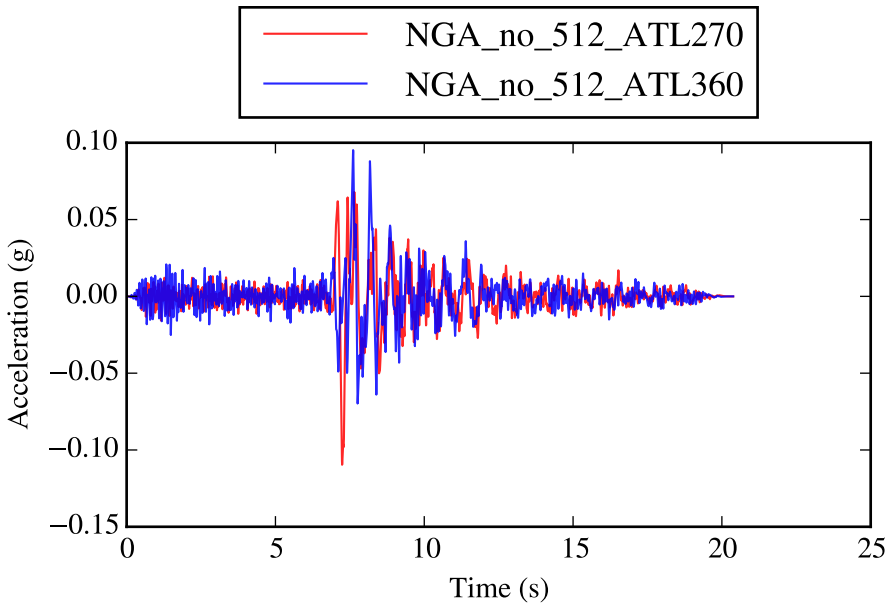


Figure F.147: WUS 147: *512_ATL270* and *512_ATL360*.

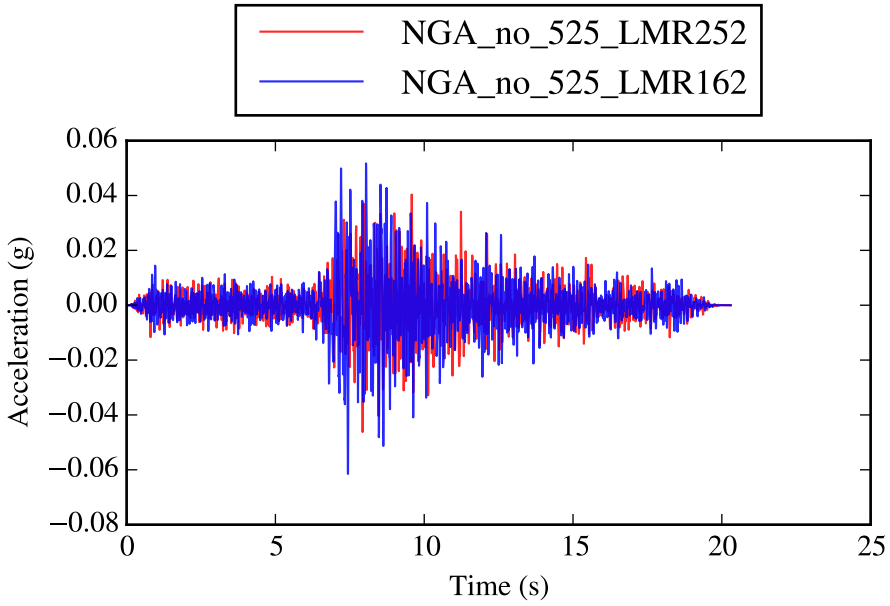


Figure F.148: WUS 148: *525_LMR252* and *525_LMR162*.

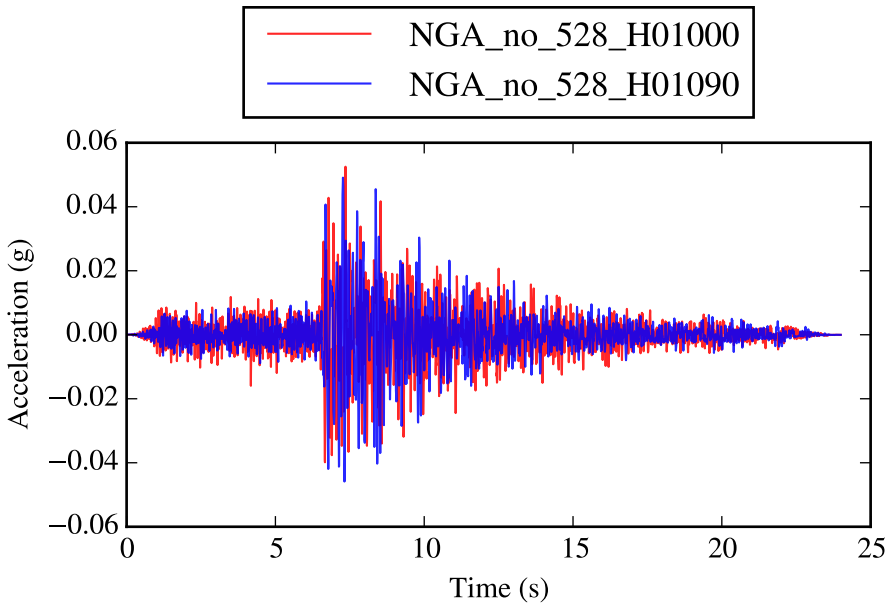


Figure F.149: WUS 149: *528_H01000* and *528_H01090*.

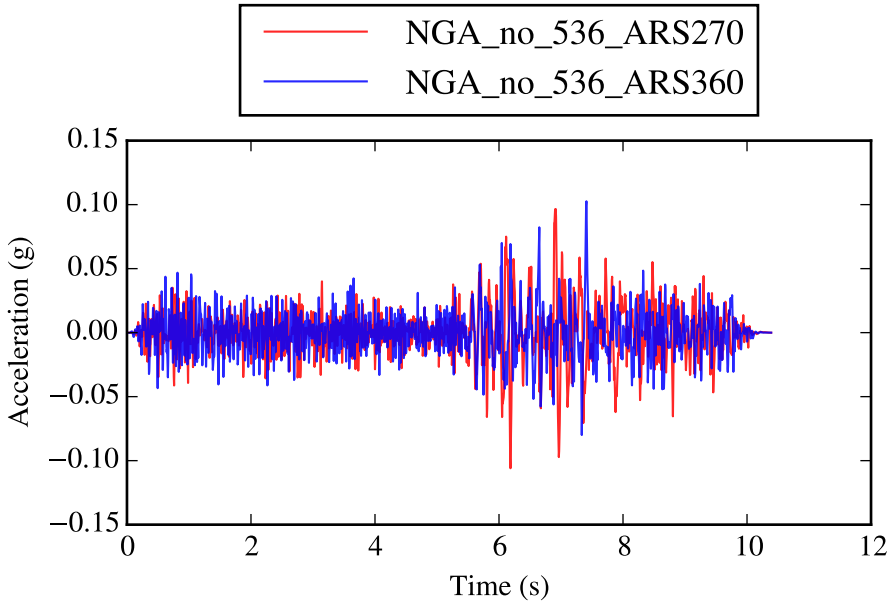


Figure F.150: WUS 150: *536_ARS270* and *536_ARS360*.

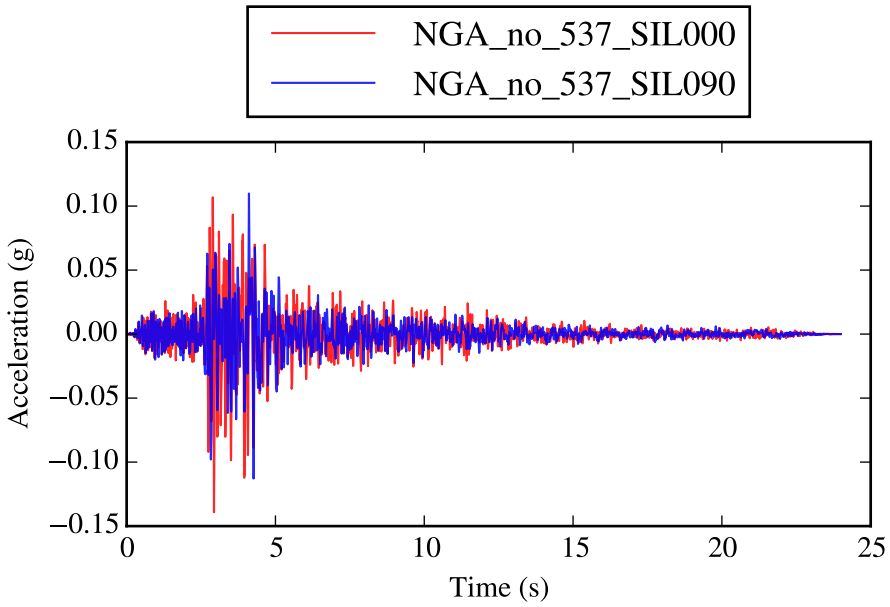


Figure F.151: WUS 151: *537_SIL000* and *537_SIL090*.

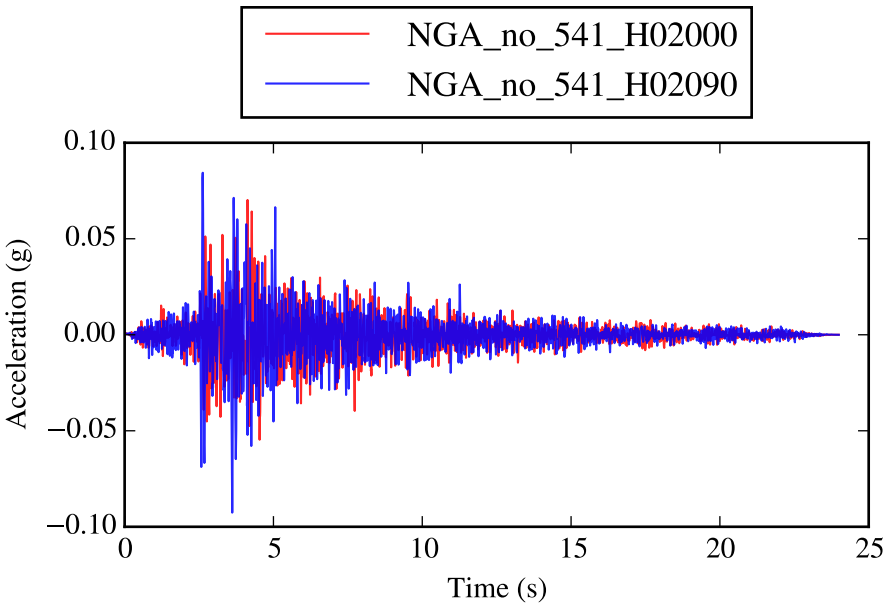


Figure F.152: WUS 152: *541_H02000* and *541_H02090*.

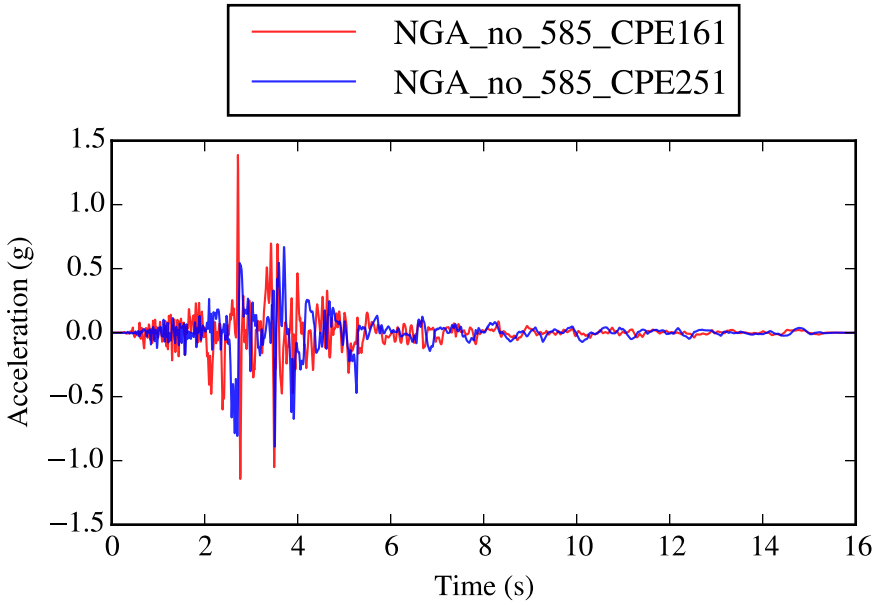


Figure F.153: WUS 153: *585_CPE161* and *585_CPE251*.

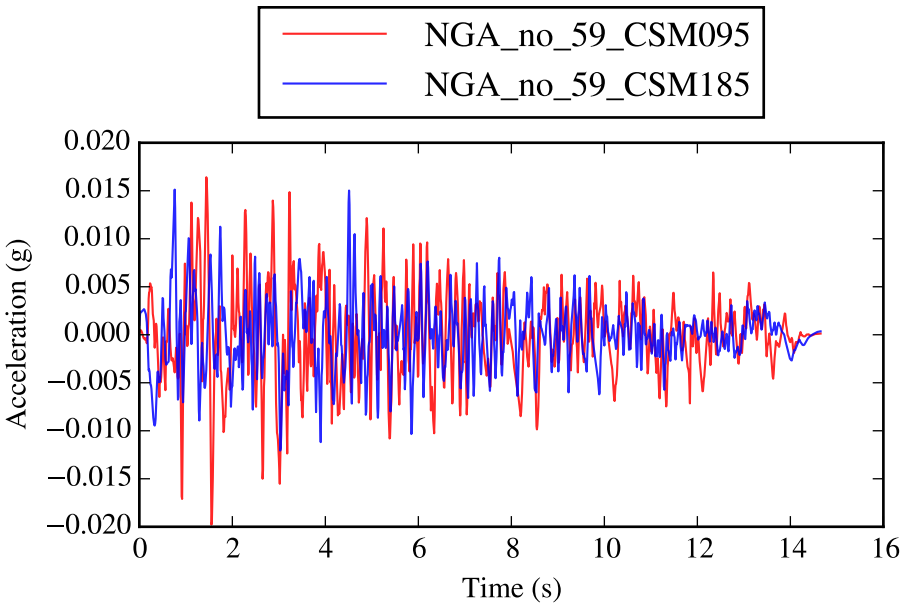


Figure F.154: WUS 154: *59_CSM095* and *59_CSM185*.

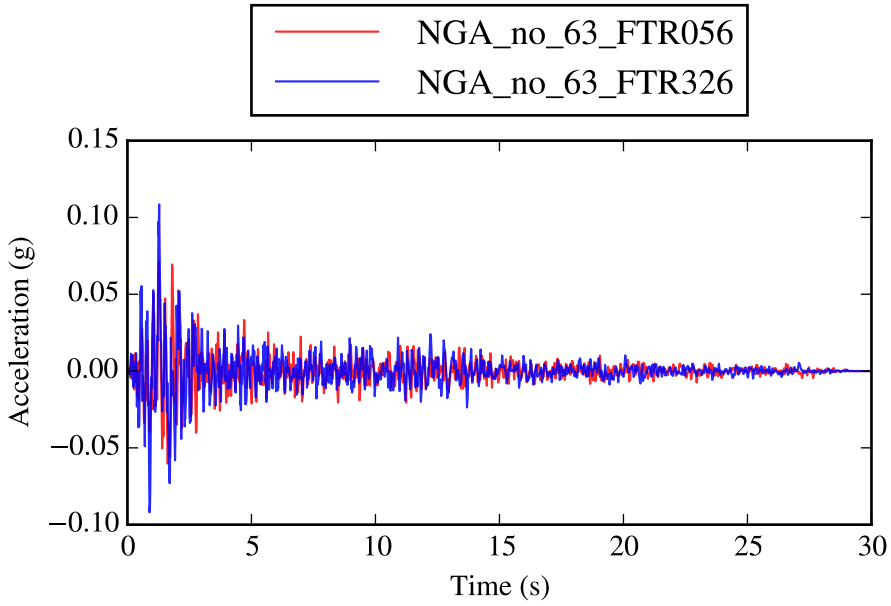


Figure F.155: WUS 155: *63_FTR056* and *63_FTR326*.

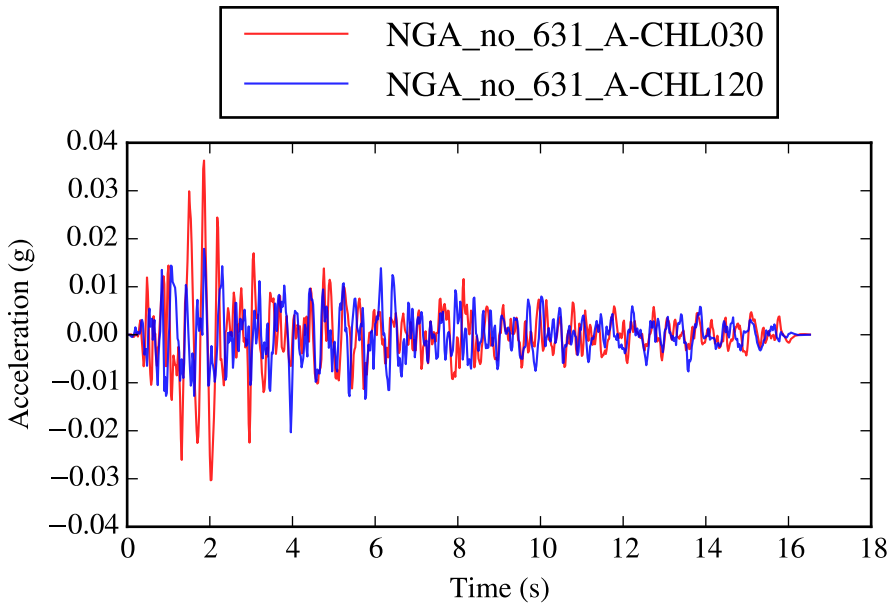


Figure F.156: WUS 156: *631_A-CHL030* and *631_A-CHL120*.

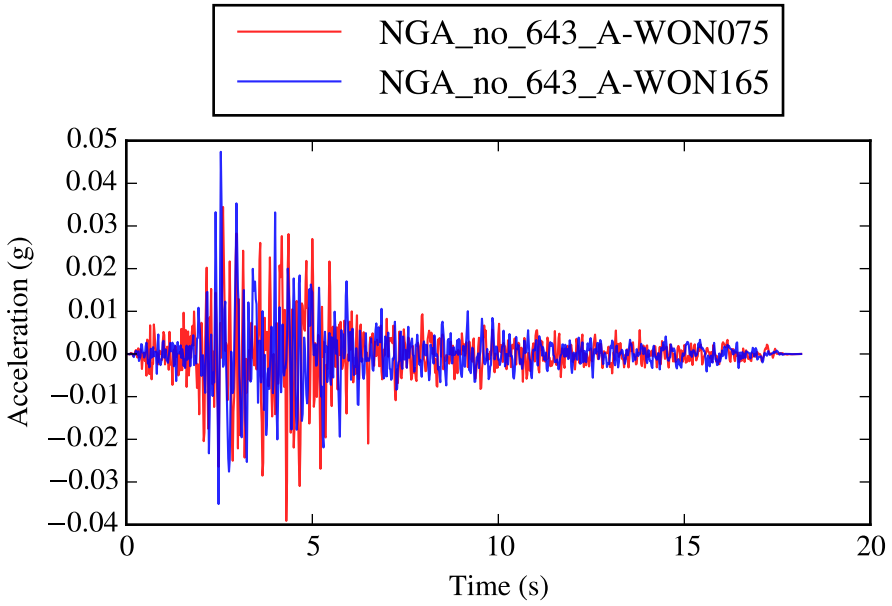


Figure F.157: WUS 157: *643_A-WON075* and *643_A-WON165*.

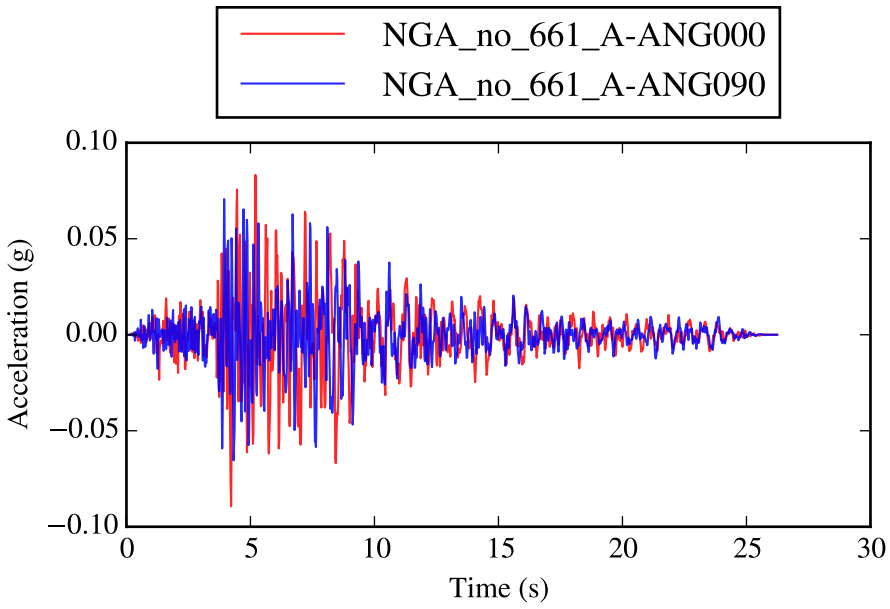


Figure F.158: WUS 158: *661_A-ANG000* and *661_A-ANG090*.

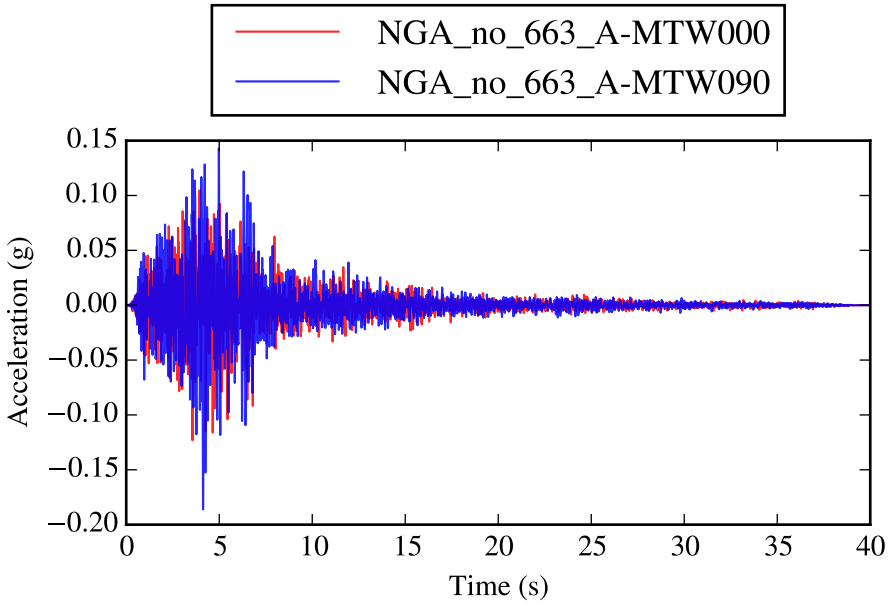


Figure F.159: WUS 159: *663_A-MTW000* and *663_A-MTW090*.

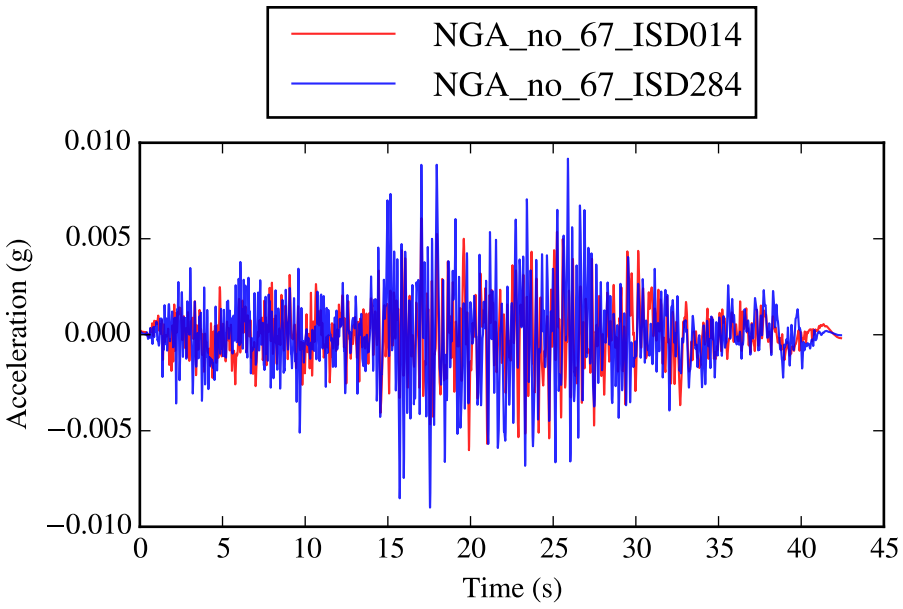


Figure F.160: WUS 160: *67_ISD014* and *67_ISD284*.

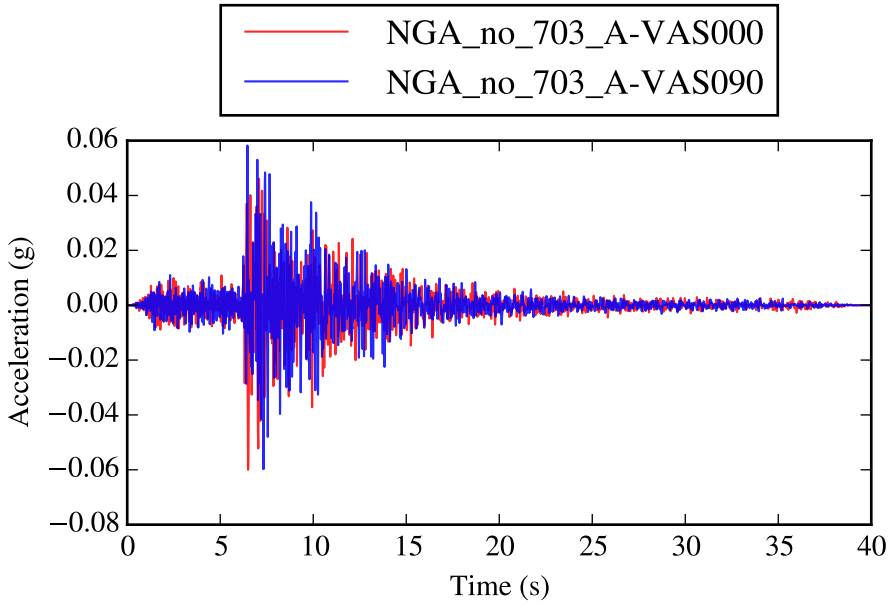


Figure F.161: WUS 161: *703_A-VAS000* and *703_A-VAS090*.

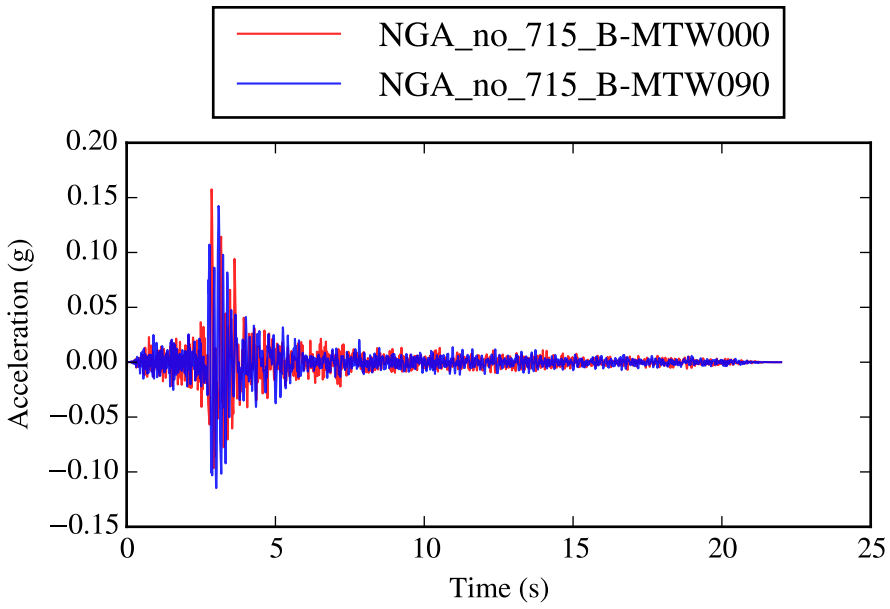


Figure F.162: WUS 162: *715_B-MTW000* and *715_B-MTW090*.

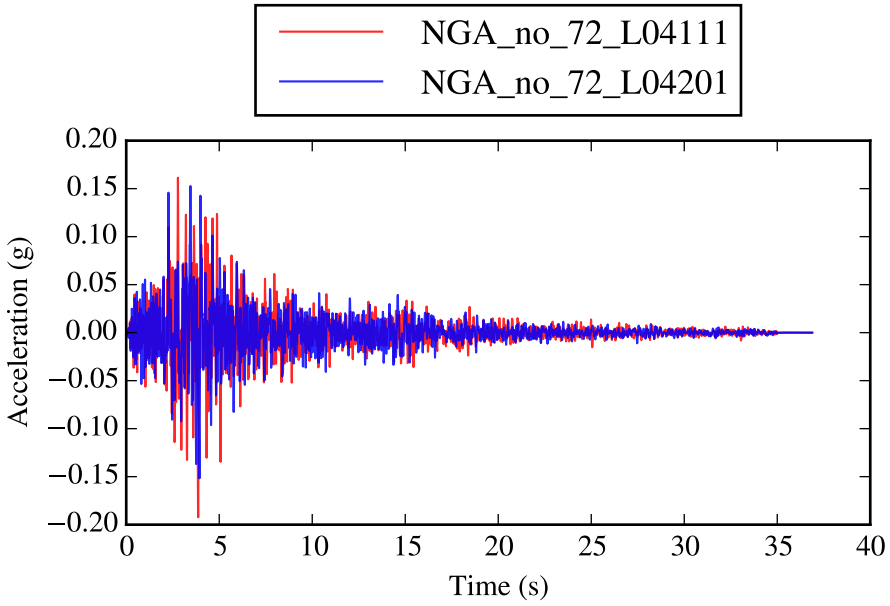


Figure F.163: WUS 163: *72_L04111* and *72_L04201*.

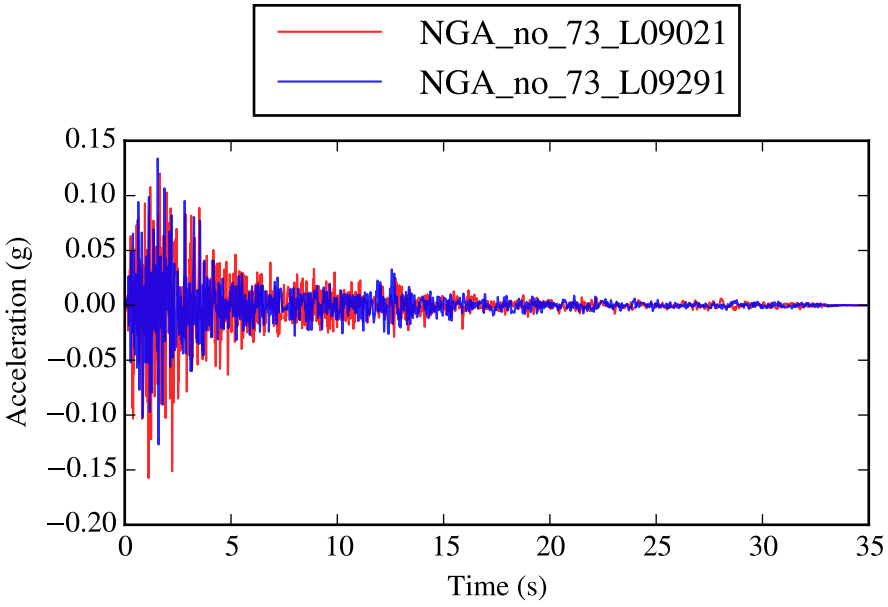


Figure F.164: WUS 164: *73_L09021* and *73_L09291*.

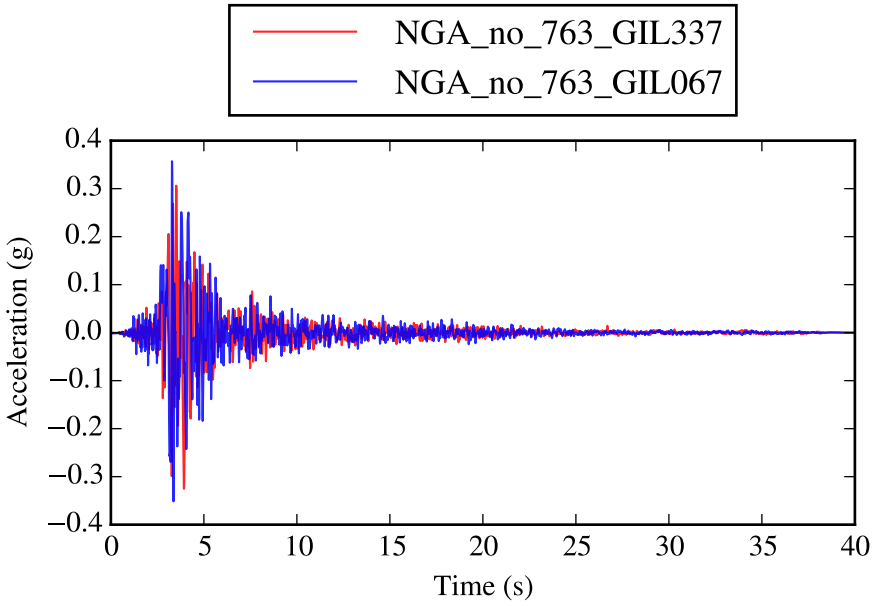


Figure F.165: WUS 165: *763_GIL337* and *763_GIL067*.

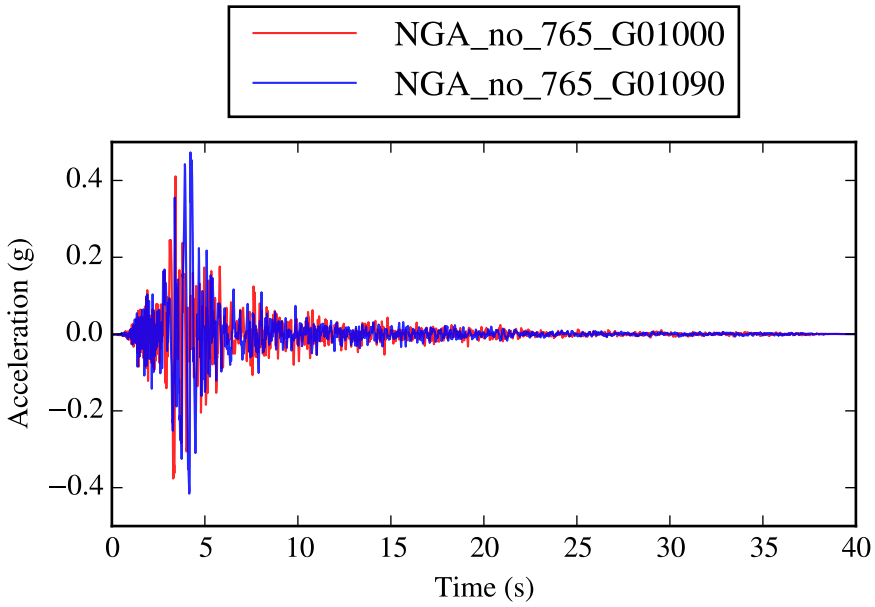


Figure F.166: WUS 166: *765_G01000* and *765_G01090*.

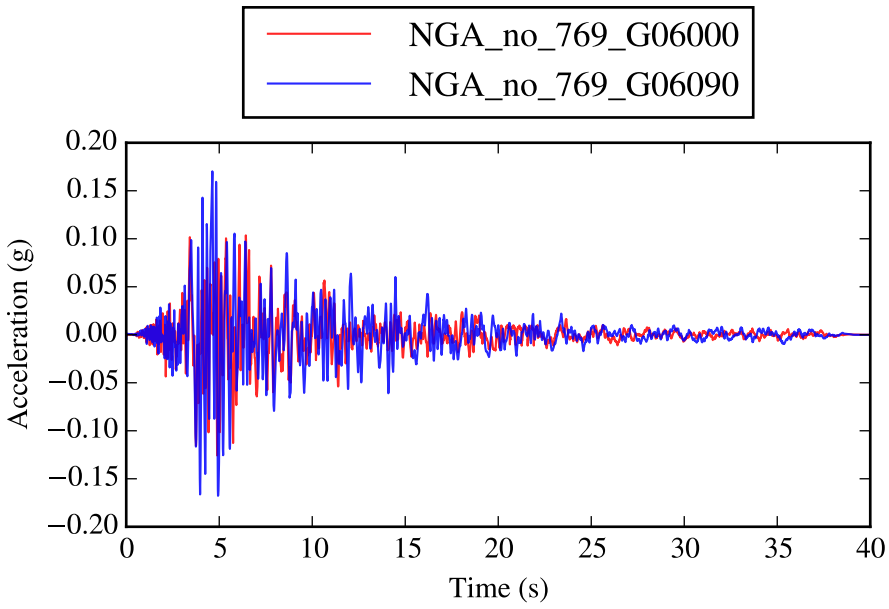


Figure F.167: WUS 167: *769_G06000* and *769_G06090*.

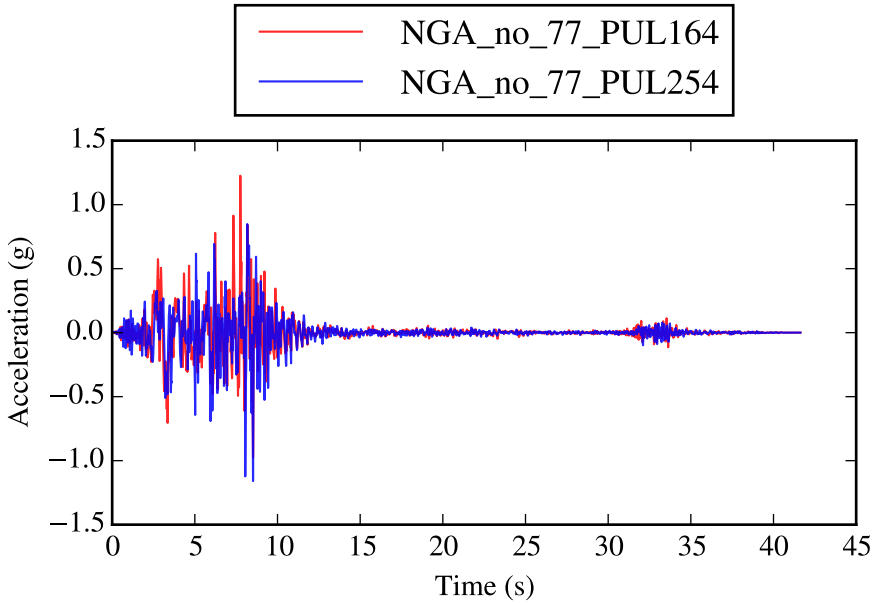


Figure F.168: WUS 168: *77_PUL164* and *77_PUL254*.

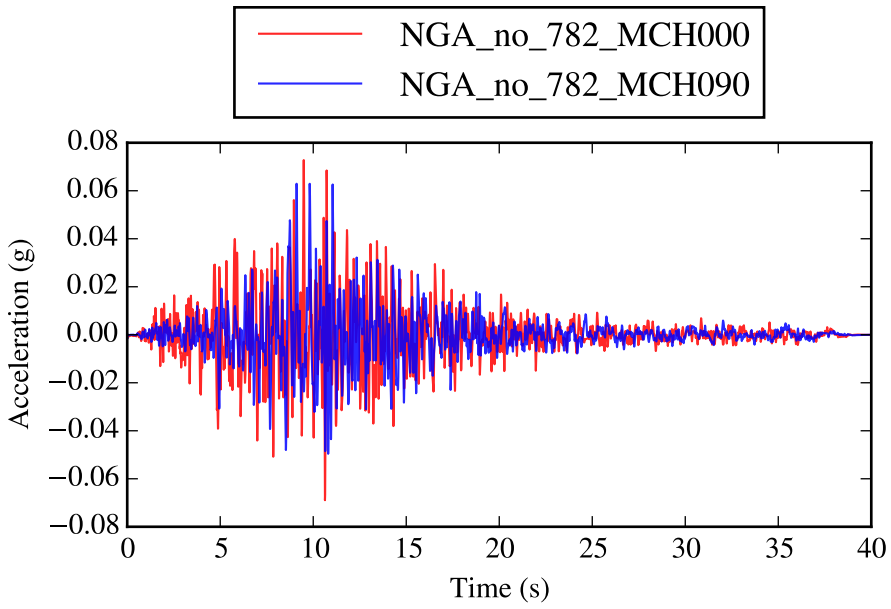


Figure F.169: WUS 169: *782_MCH000* and *782_MCH090*.

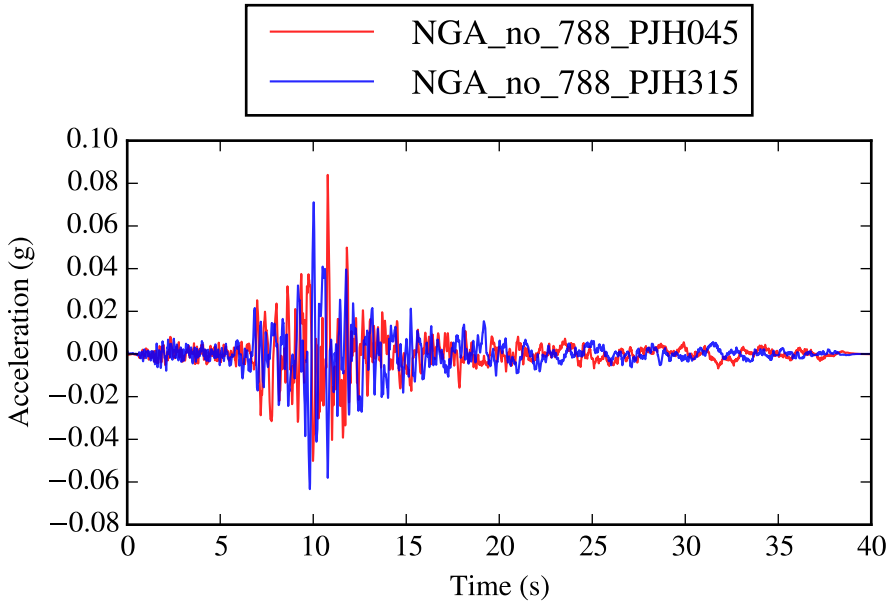


Figure F.170: WUS 170: *788_PJH045* and *788_PJH315*.

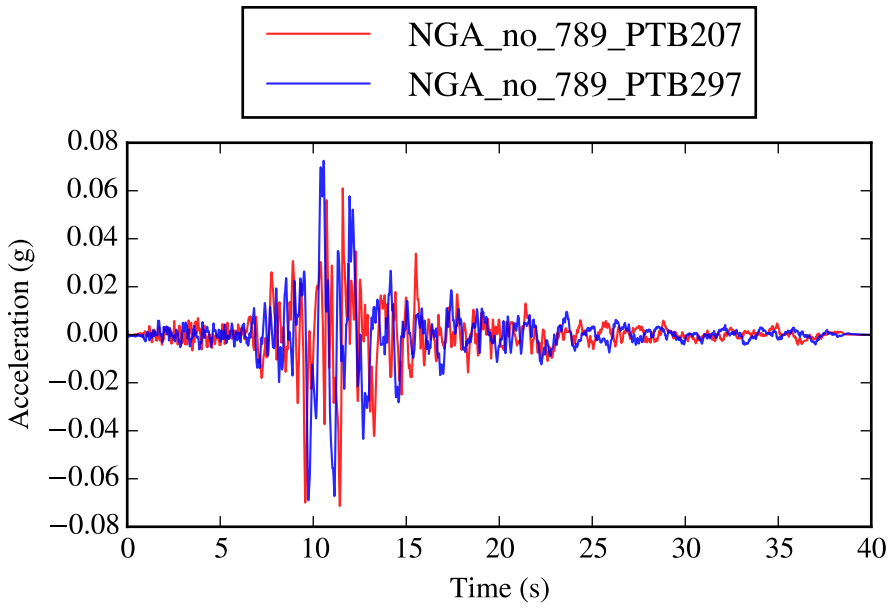


Figure F.171: WUS 171: *789_PTB207* and *789_PTB297*.

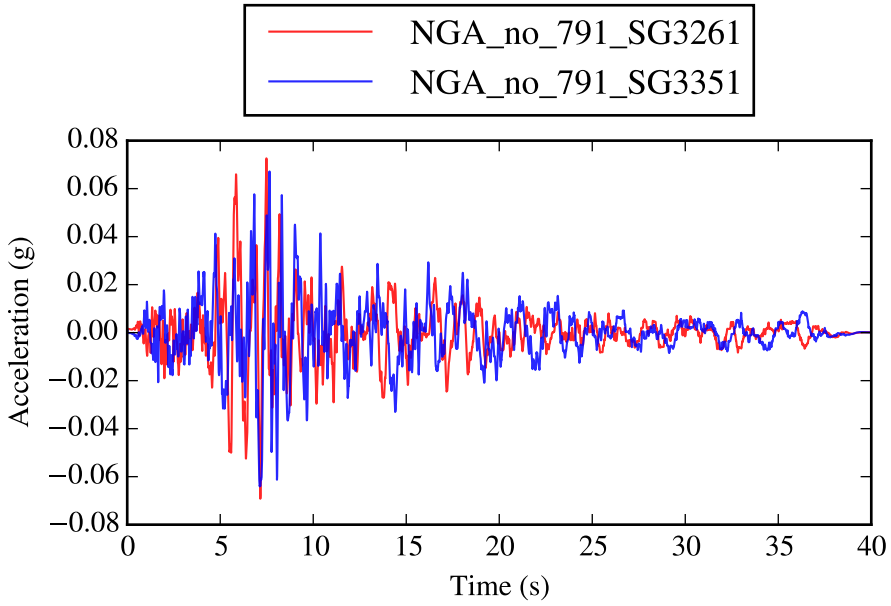


Figure F.172: WUS 172: *791_SG3261* and *791_SG3351*.

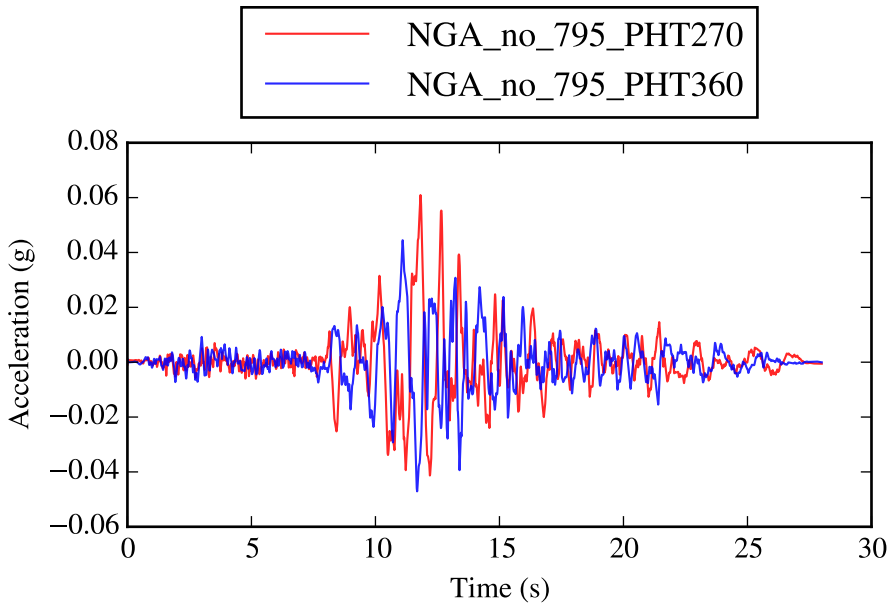


Figure F.173: WUS 173: *795_PHT270* and *795_PHT360*.

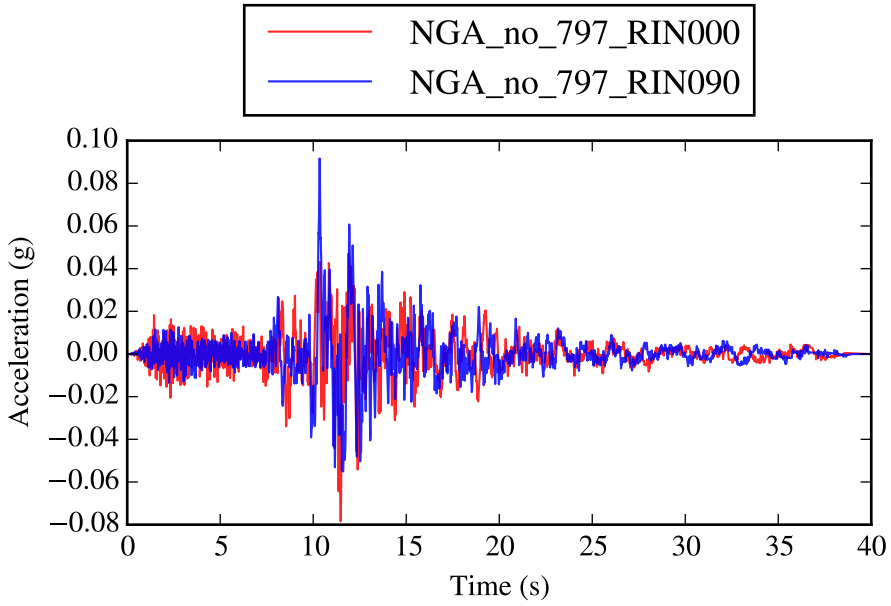


Figure F.174: WUS 174: *797_RIN000* and *797_RIN090*.

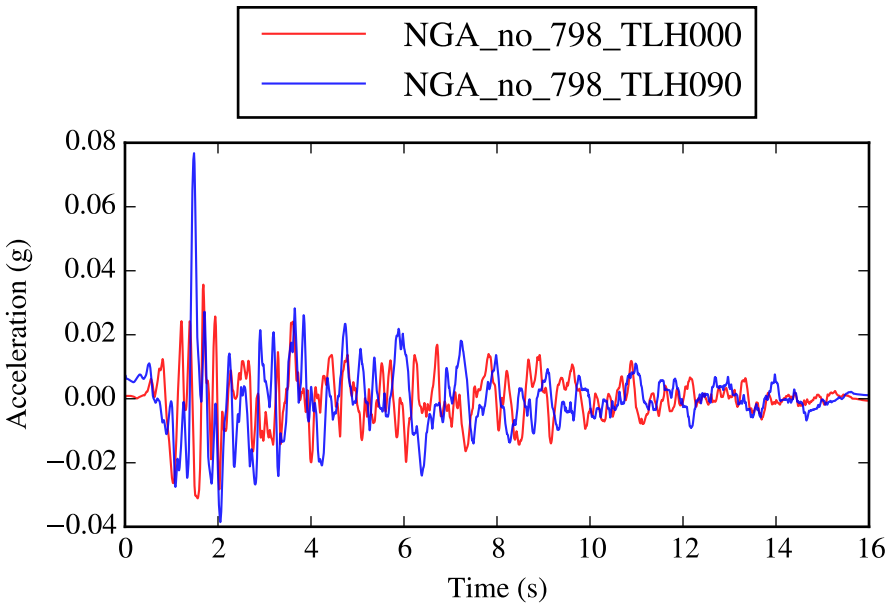


Figure F.175: WUS 175: *798_TLH000* and *798_TLH090*.

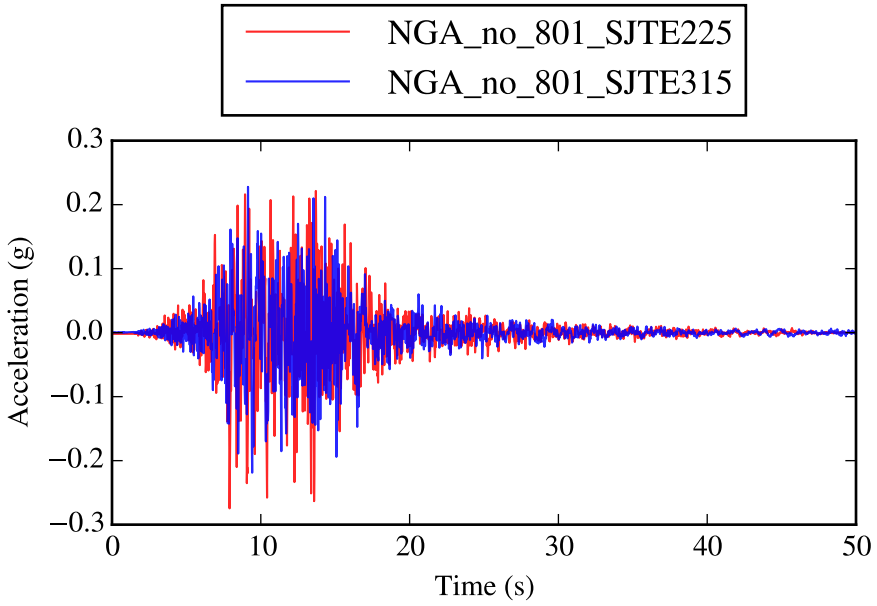


Figure F.176: WUS 176: *801_SJTE225* and *801_SJTE315*.

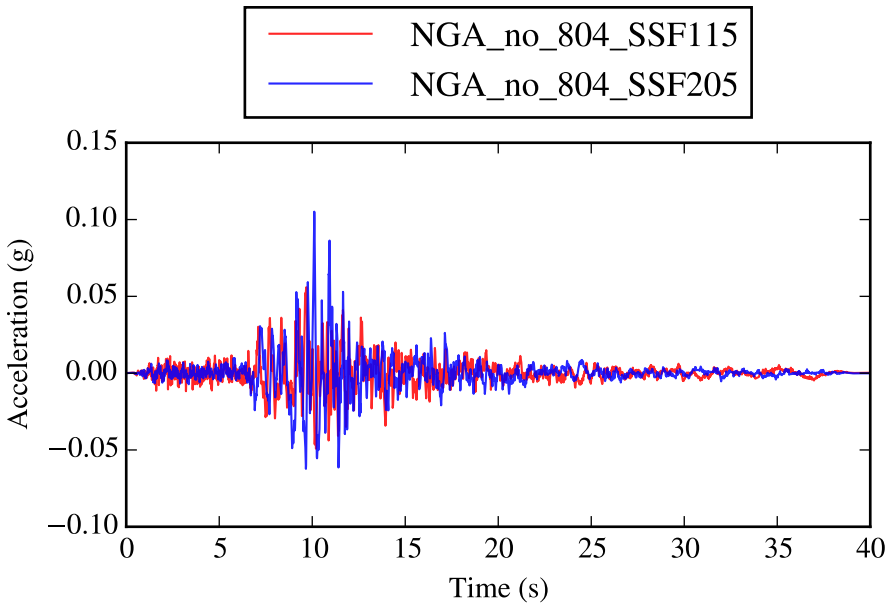


Figure F.177: WUS 177: *804_SSF115* and *804_SSF205*.

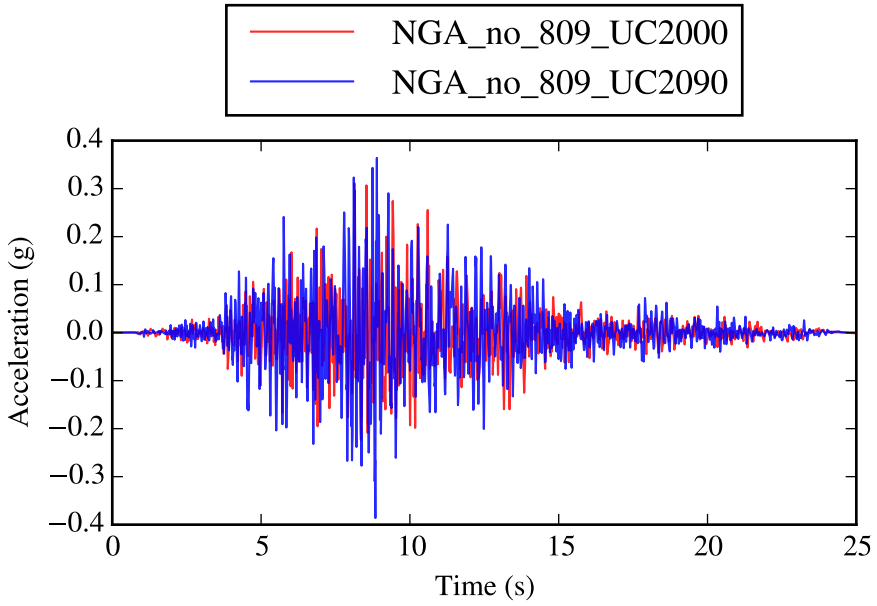


Figure F.178: WUS 178: *809_UC2000* and *809_UC2090*.

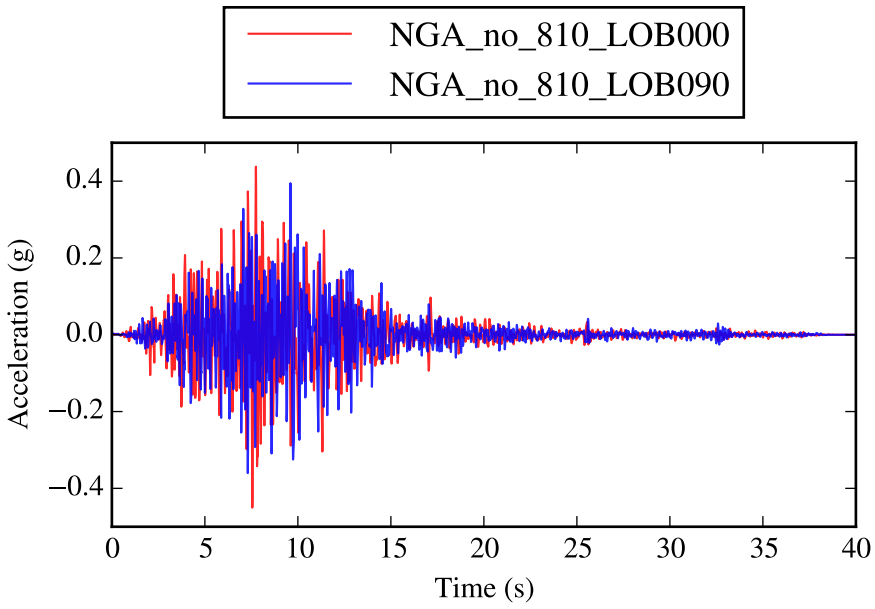


Figure F.179: WUS 179: *810_LOB000* and *810_LOB090*.

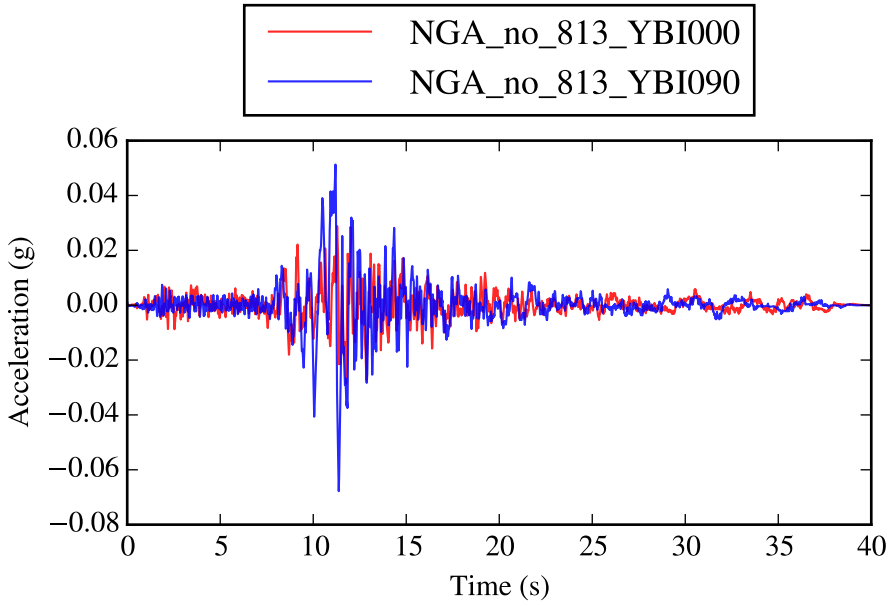


Figure F.180: WUS 180: *813_YBI000* and *813_YBI090*.

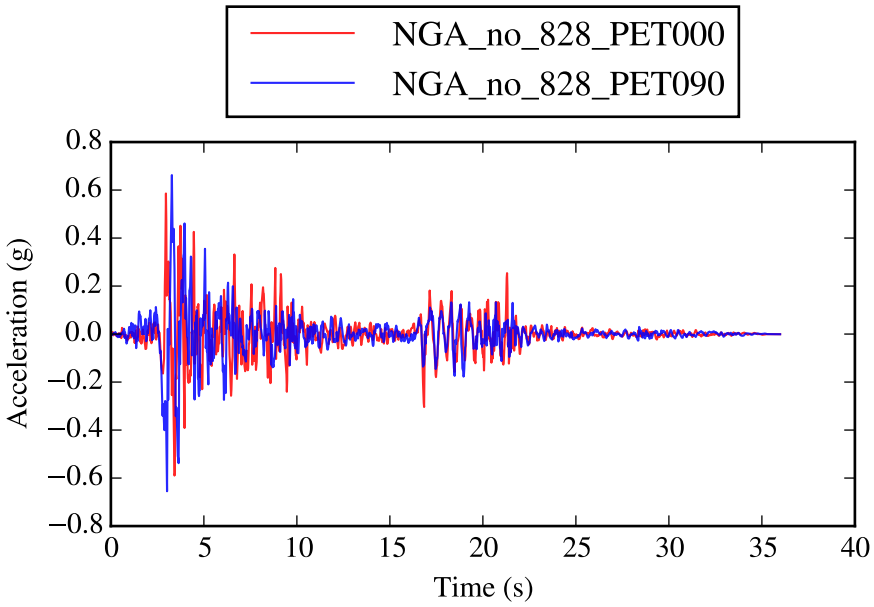


Figure F.181: WUS 181: *828_PET000* and *828_PET090*.

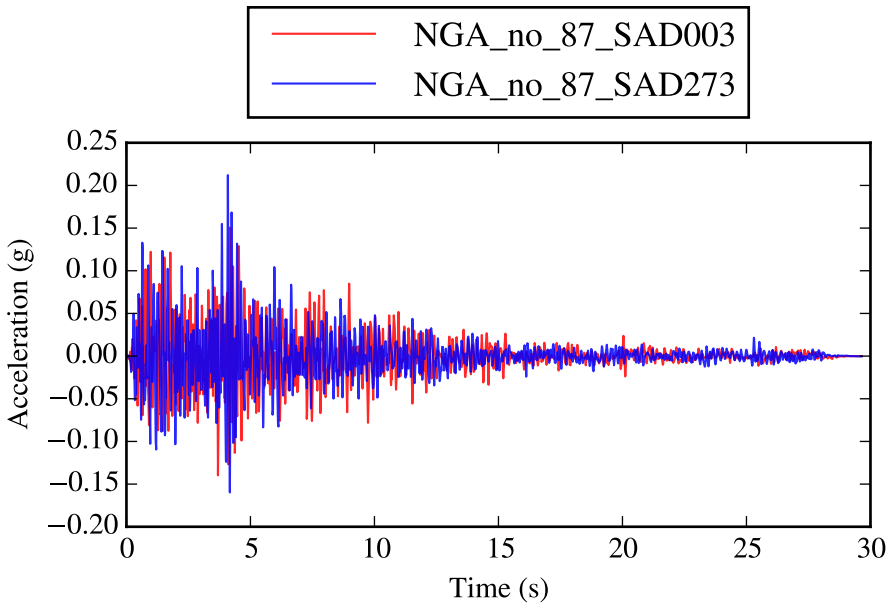


Figure F.182: WUS 182: *87_SAD003* and *87_SAD273*.

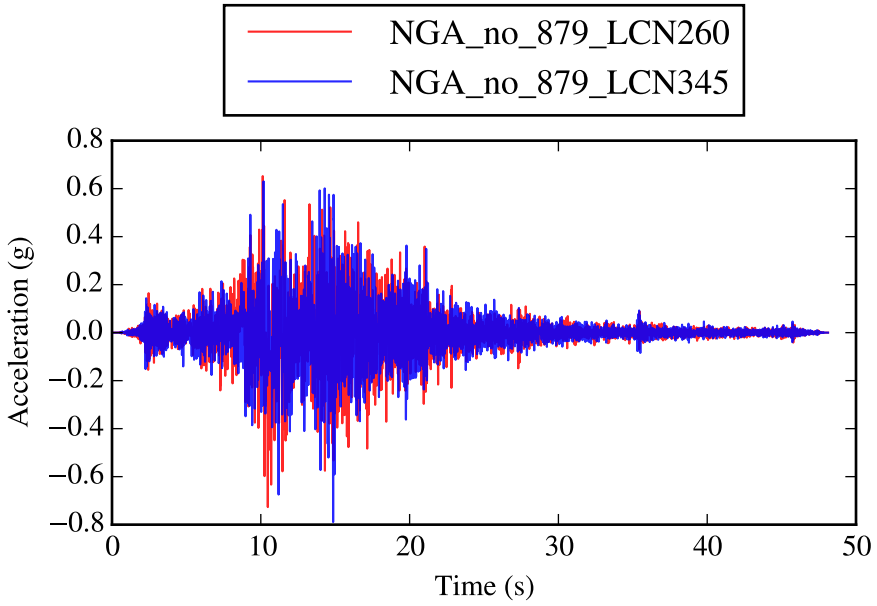


Figure F.183: WUS 183: *879_LCN260* and *879_LCN345*.

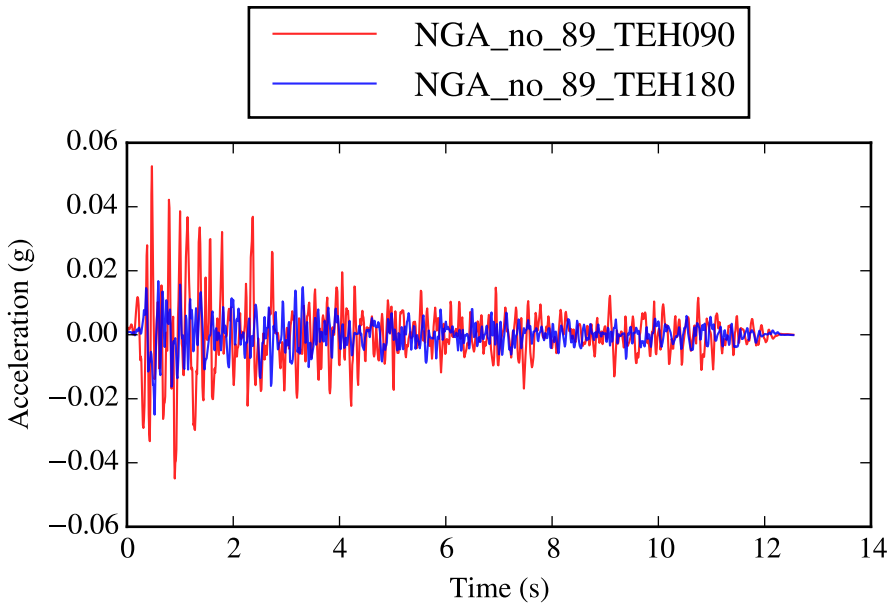


Figure F.184: WUS 184: *89_TEH090* and *89_TEH180*.

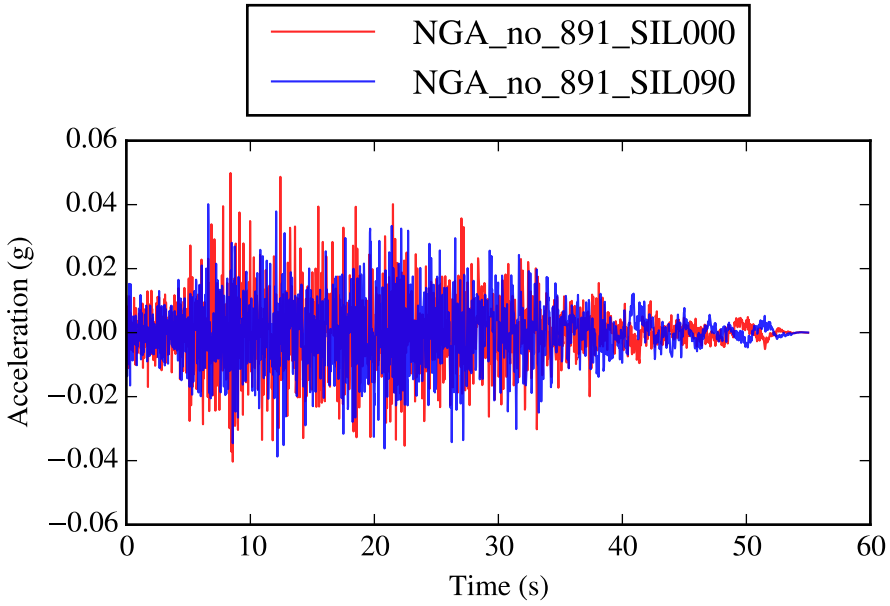


Figure F.185: WUS 185: *891_SIL000* and *891_SIL090*.

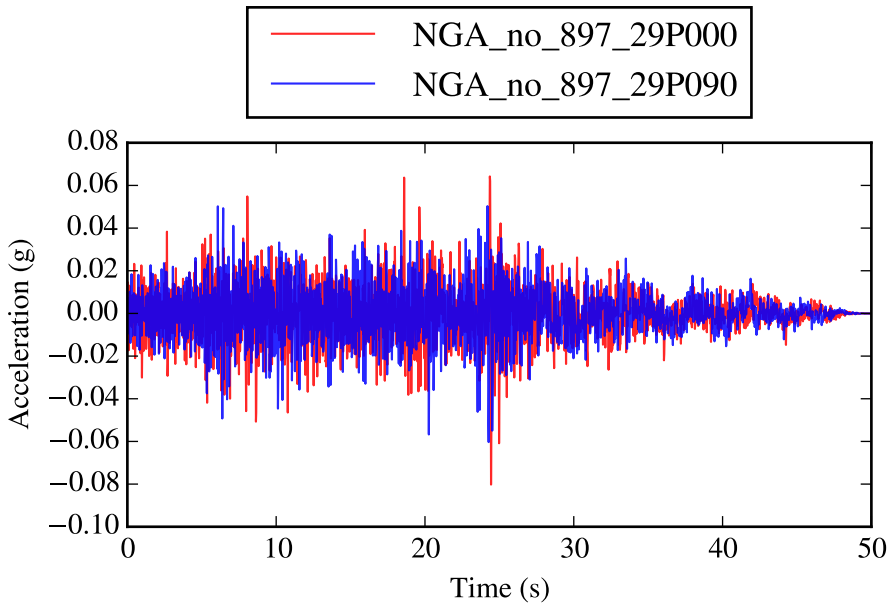


Figure F.186: WUS 186: *897_29P000* and *897_29P090*.

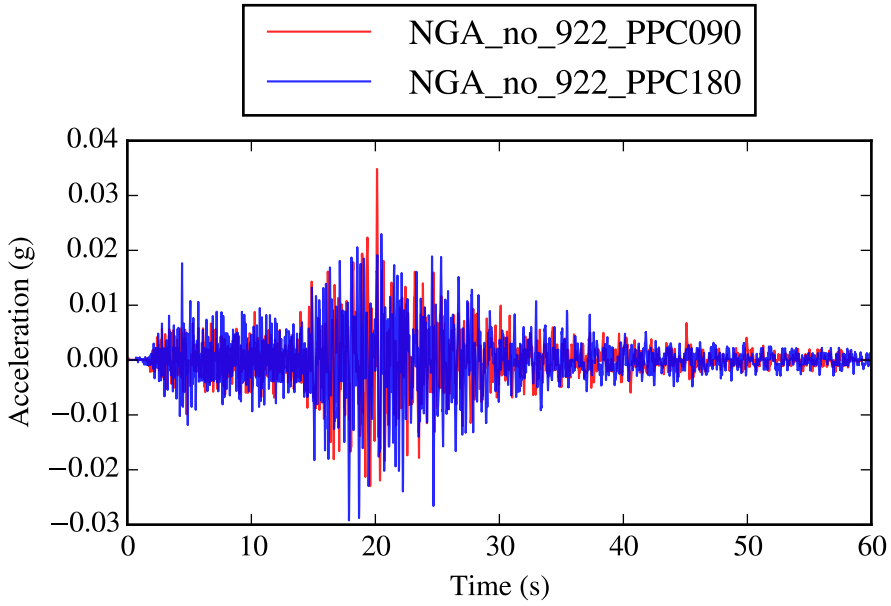


Figure F.187: WUS 187: *922_PPC090* and *922_PPC180*.

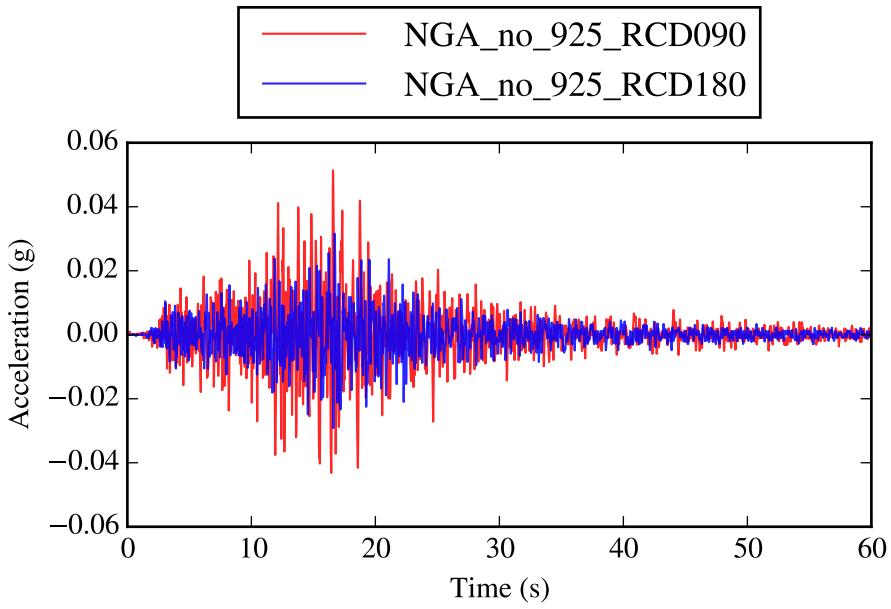


Figure F.188: WUS 188: *925_RCD090* and *925_RCD180*.

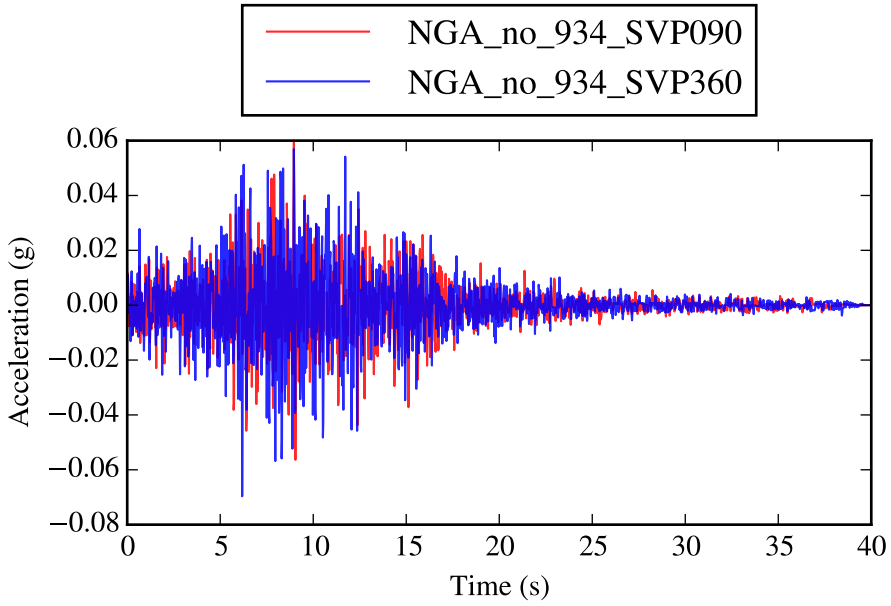


Figure F.189: WUS 189: *934_SVP090* and *934_SVP360*.

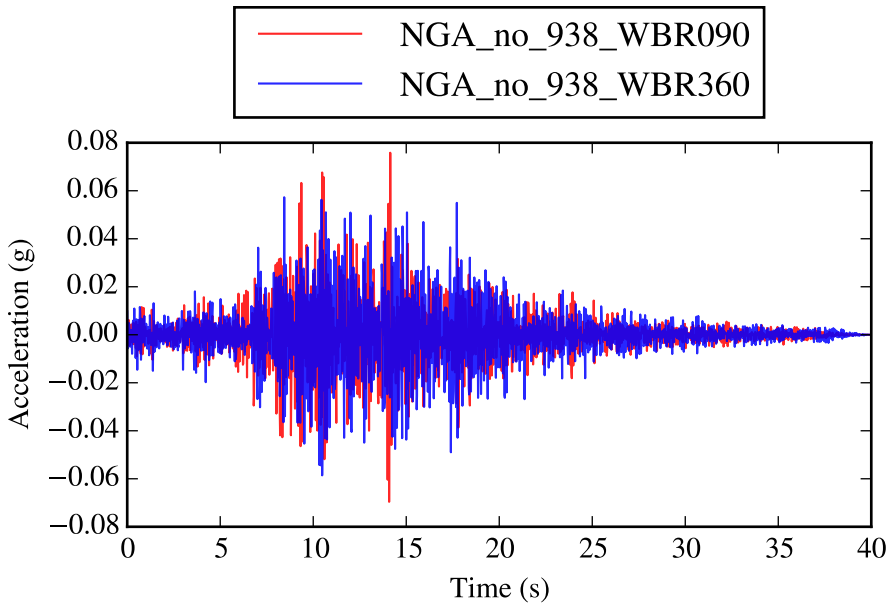


Figure F.190: WUS 190: *938_WBR090* and *938_WBR360*.

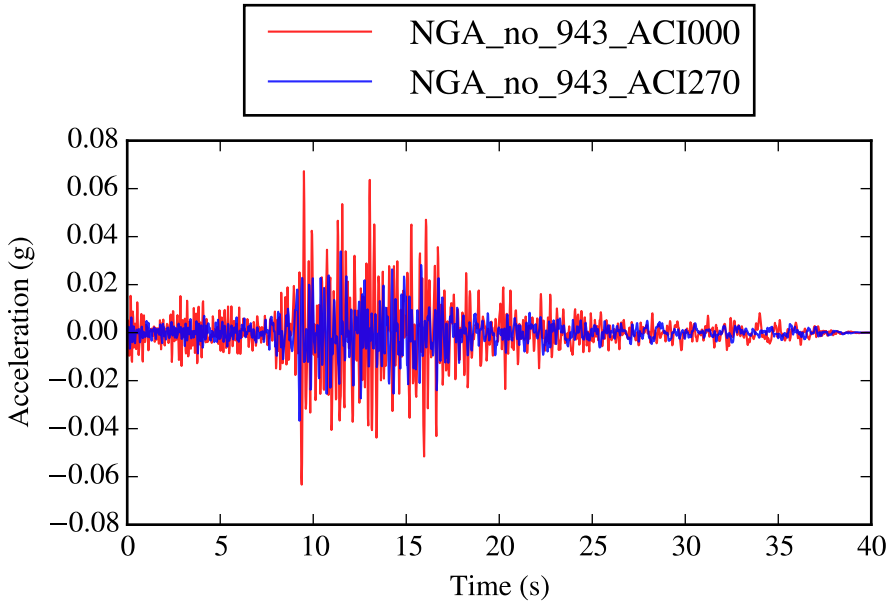


Figure F.191: WUS 191: *943_ACI000* and *943_ACI270*.

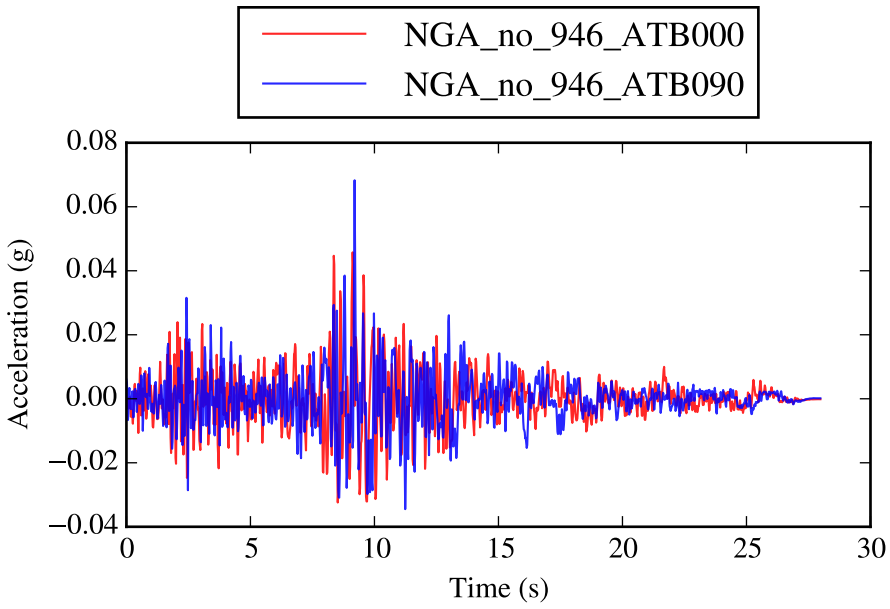


Figure F.192: WUS 192: *946_ATB000* and *946_ATB090*.

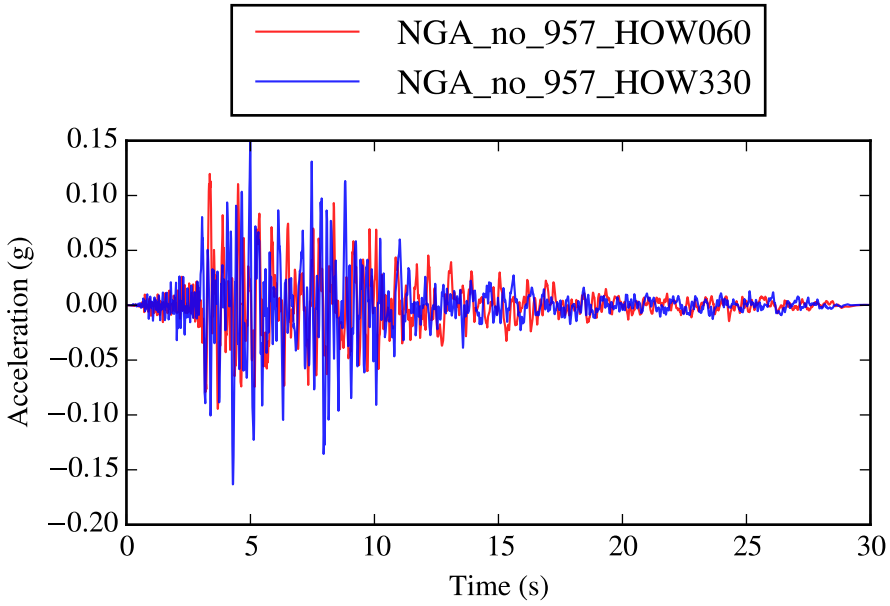


Figure F.193: WUS 193: *957_HOW060* and *957_HOW330*.

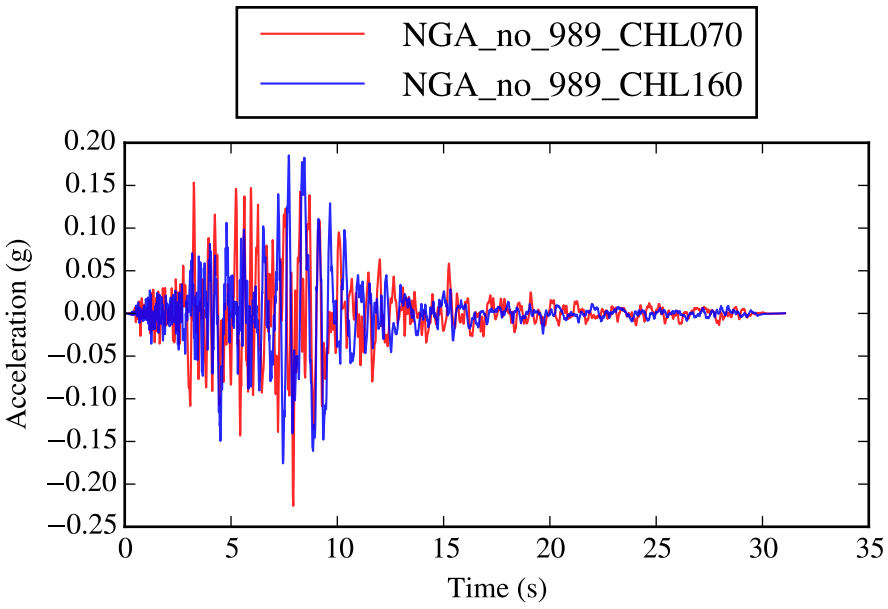


Figure F.194: WUS 194: *989_CHL070* and *989_CHL160*.

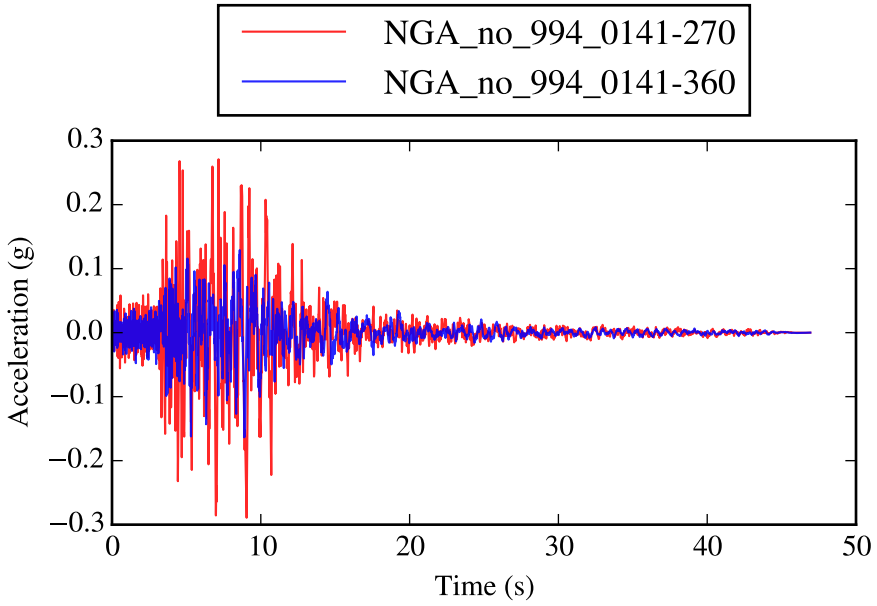


Figure F.195: WUS 195: *994_0141-270* and *994_0141-360*.

F.2 McGuire et al. (2001) Earthquake Motions (CEUS)

The following figures show the CEUS earthquake motions (McGuire et al. 2001[1]) used in the equivalent-linear site response analyses. More details on each motion are given in the electronic supplements of Chapters 4 and 5.

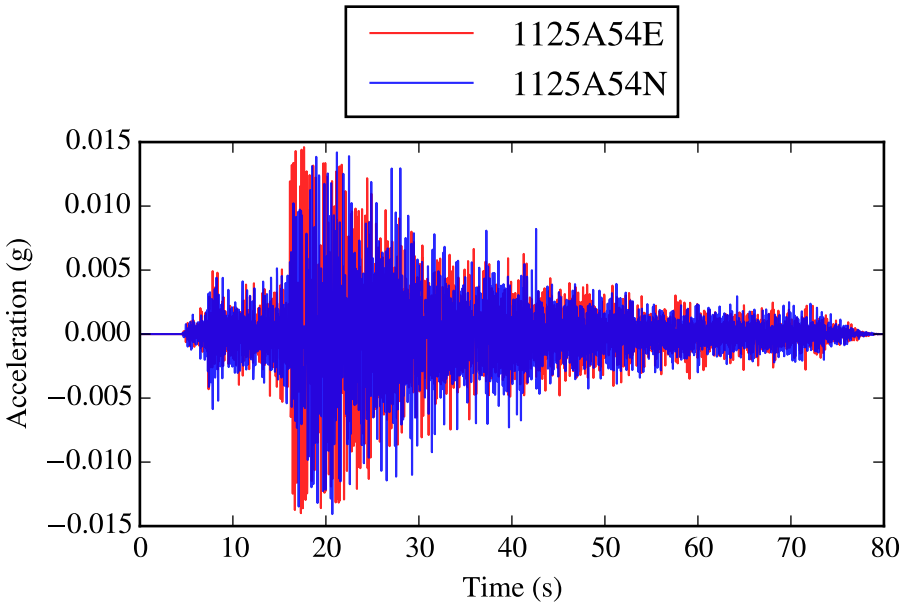


Figure F.196: CEUS 1: *1125A54E* and *1125A54N*.

[1]McGuire, R. K., Silva, W. J., and Costantino, C. J. (2001). *Technical Basis for Revision of Regulatory Guidance on Design Ground Motions: Hazard- and Risk-consistent Ground Motion Spectra Guidelines.* Division of Engineering Technology, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, Washington, D.C.

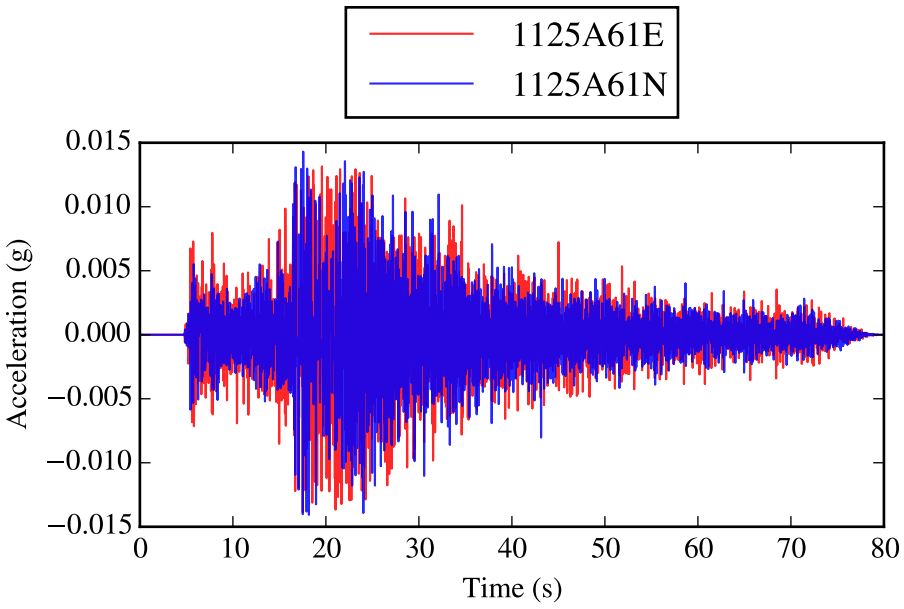


Figure F.197: CEUS 2: *1125A61E* and *1125A61N*.

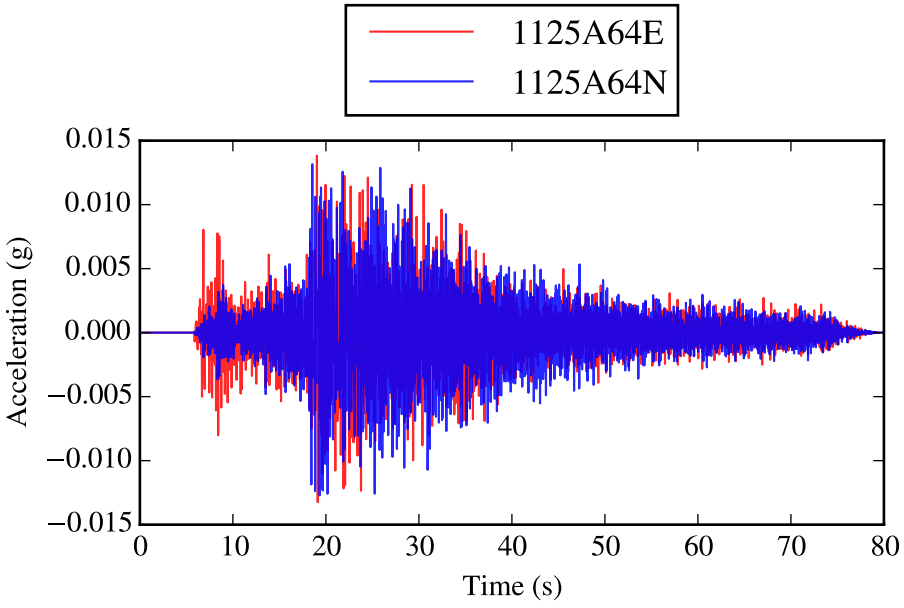


Figure F.198: CEUS 3: *1125A64E* and *1125A64N*.

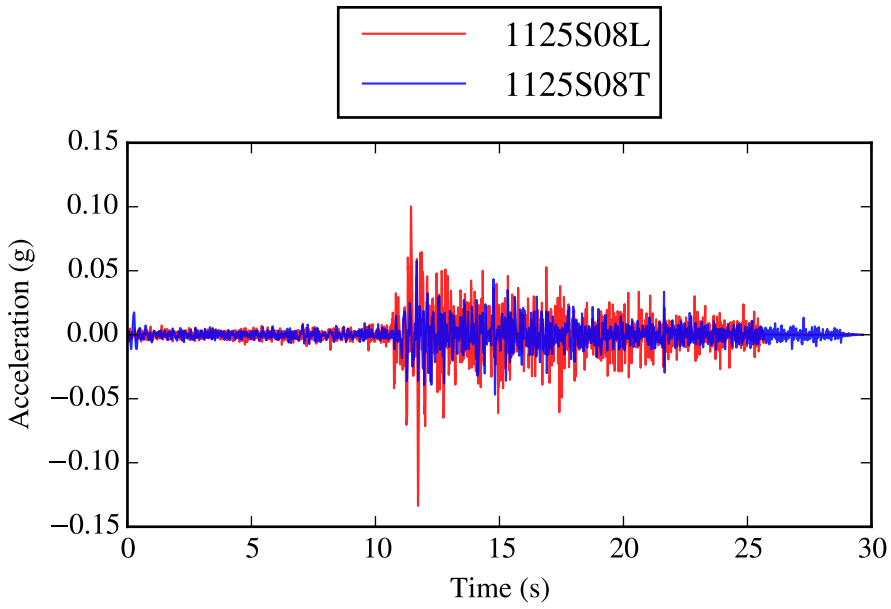


Figure F.199: CEUS 4: *1125S08L* and *1125S08T*.

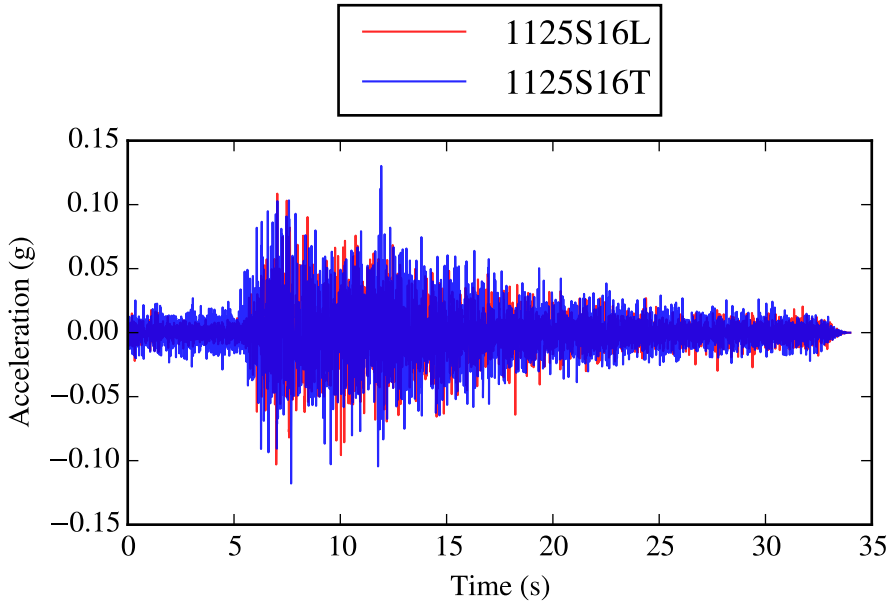


Figure F.200: CEUS 5: *1125S16L* and *1125S16T*.

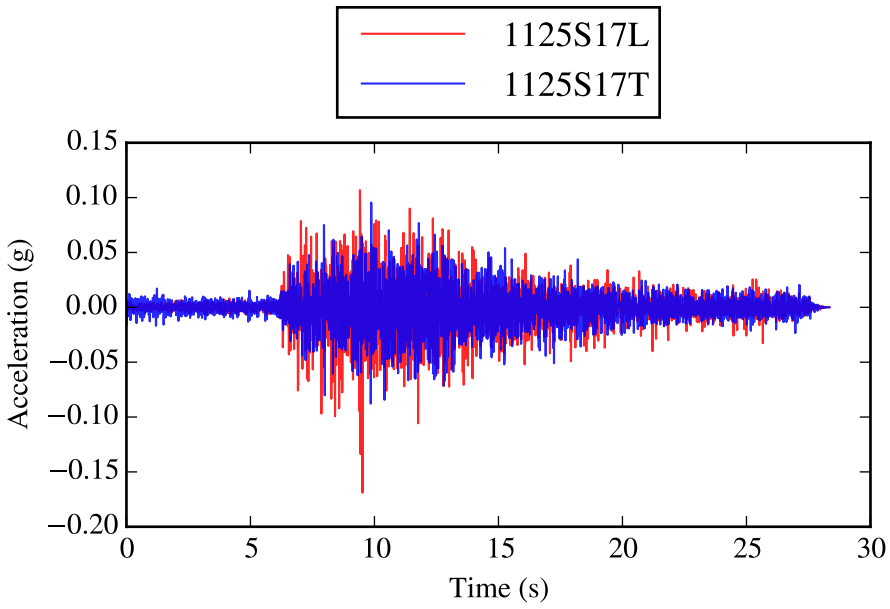


Figure F.201: CEUS 6: *1125S17L* and *1125S17T*.

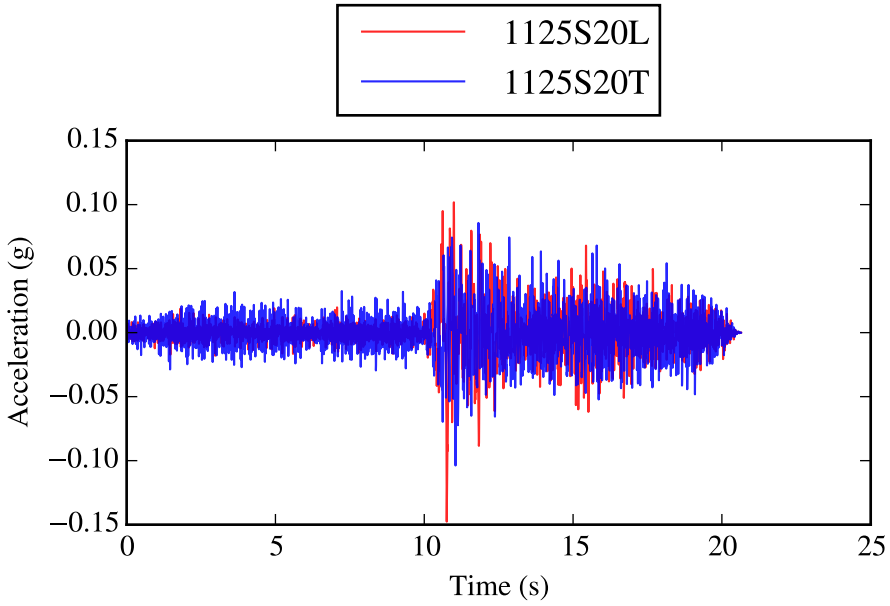


Figure F.202: CEUS 7: *1125S20L* and *1125S20T*.

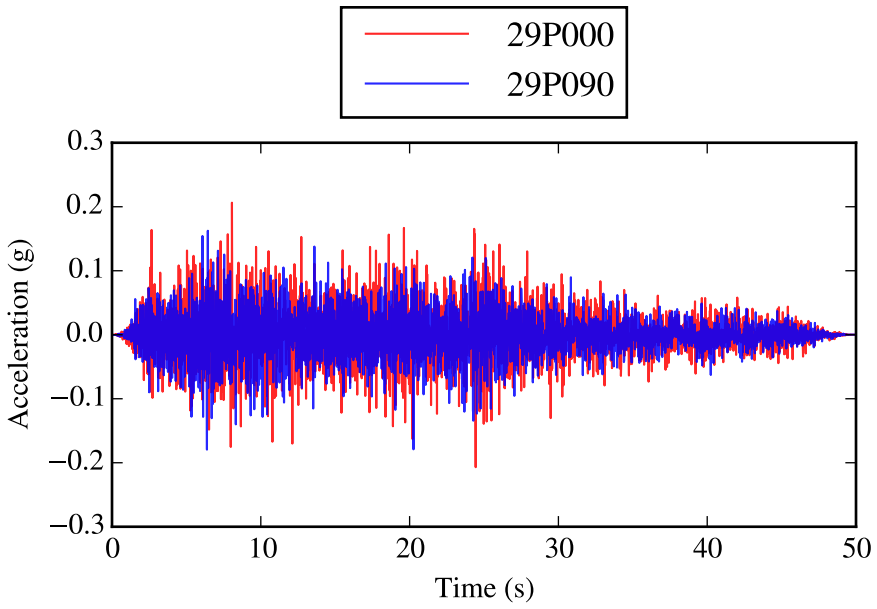


Figure F.203: CEUS 8: *29P000* and *29P090*.

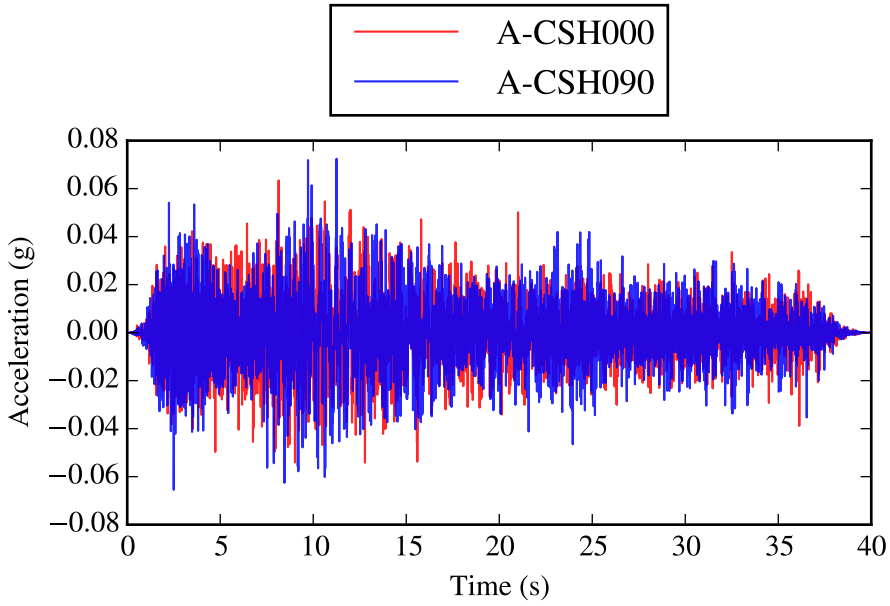


Figure F.204: CEUS 9: *A-CSH000* and *A-CSH090*.

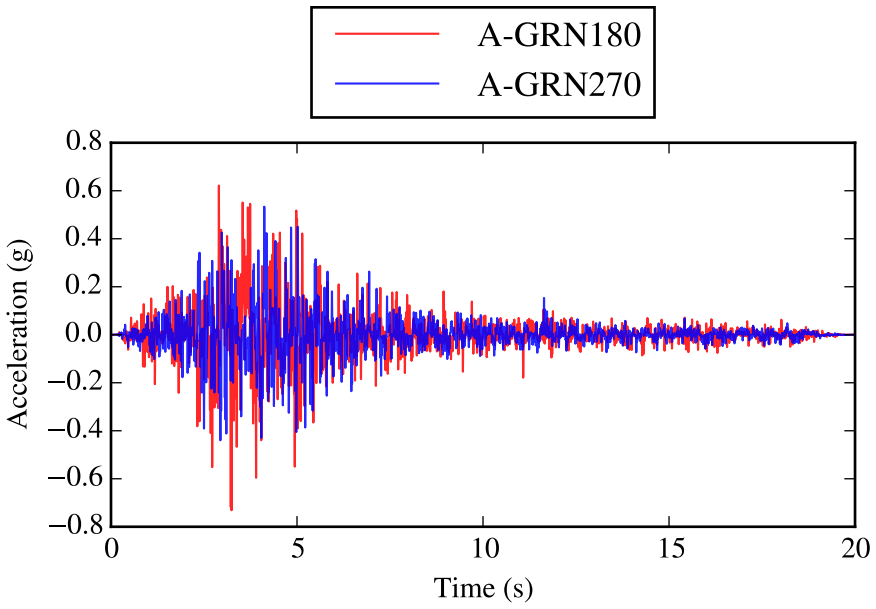


Figure F.205: CEUS 10: *A-GRN180* and *A-GRN270*.

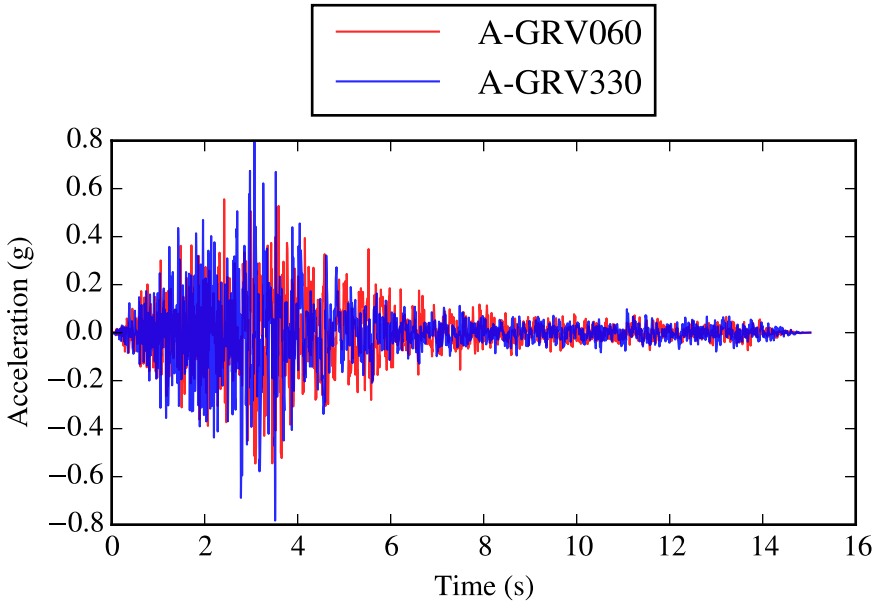


Figure F.206: CEUS 11: *A-GRV060* and *A-GRV330*.

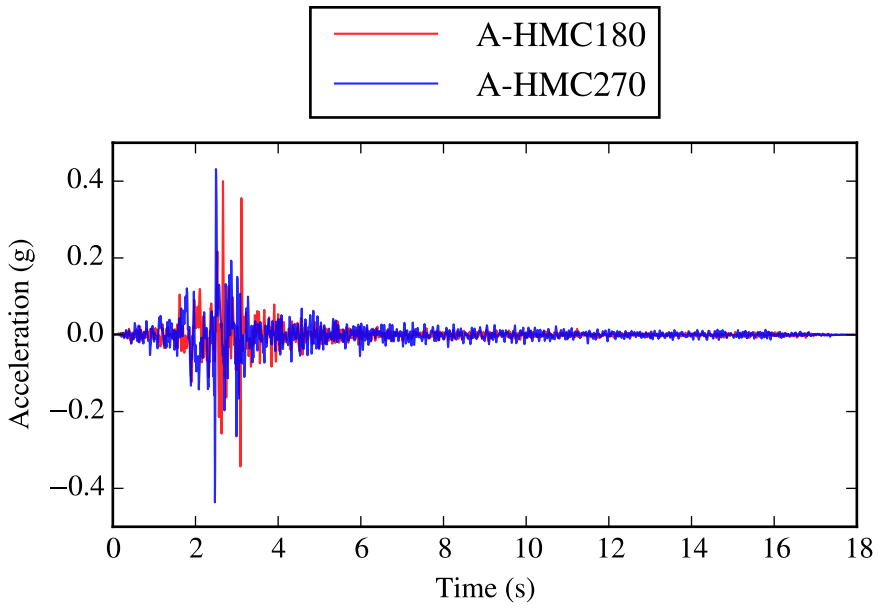


Figure F.207: CEUS 12: *A-HMC180* and *A-HMC270*.

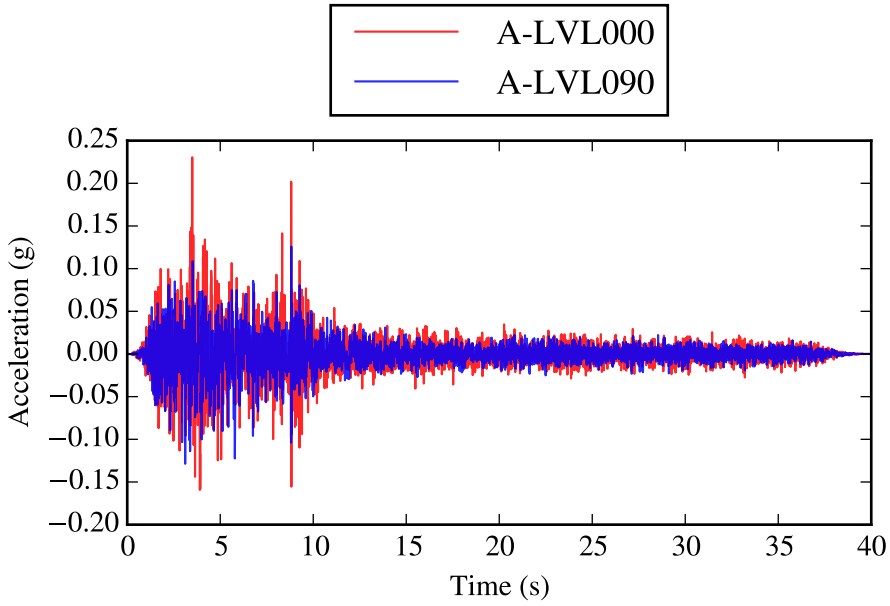


Figure F.208: CEUS 13: *A-LVL000* and *A-LVL090*.

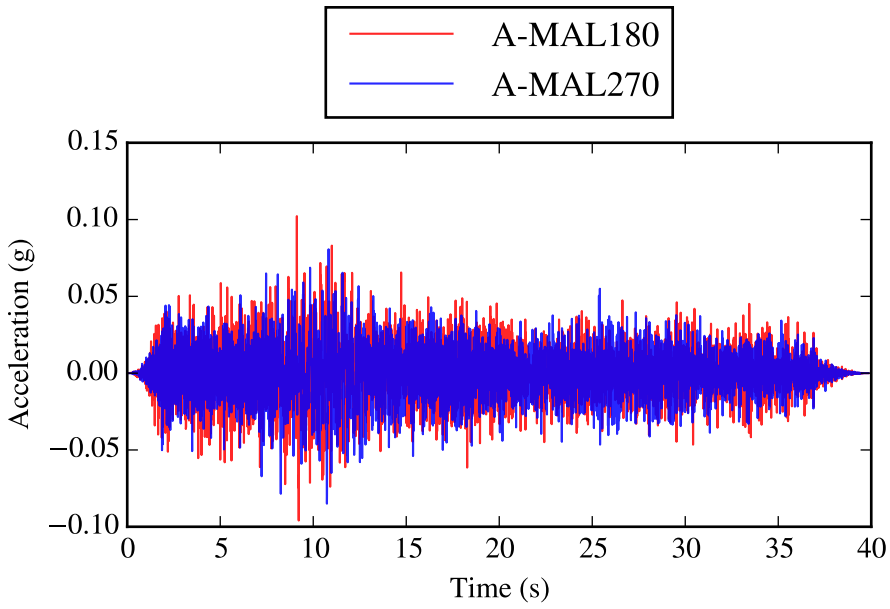


Figure F.209: CEUS 14: *A-MAL180* and *A-MAL270*.

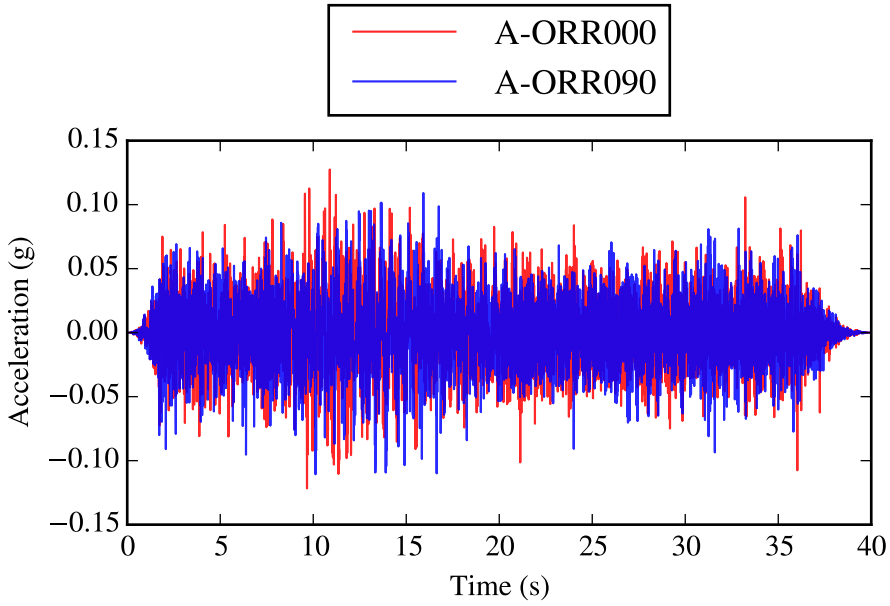


Figure F.210: CEUS 15: *A-ORR000* and *A-ORR090*.

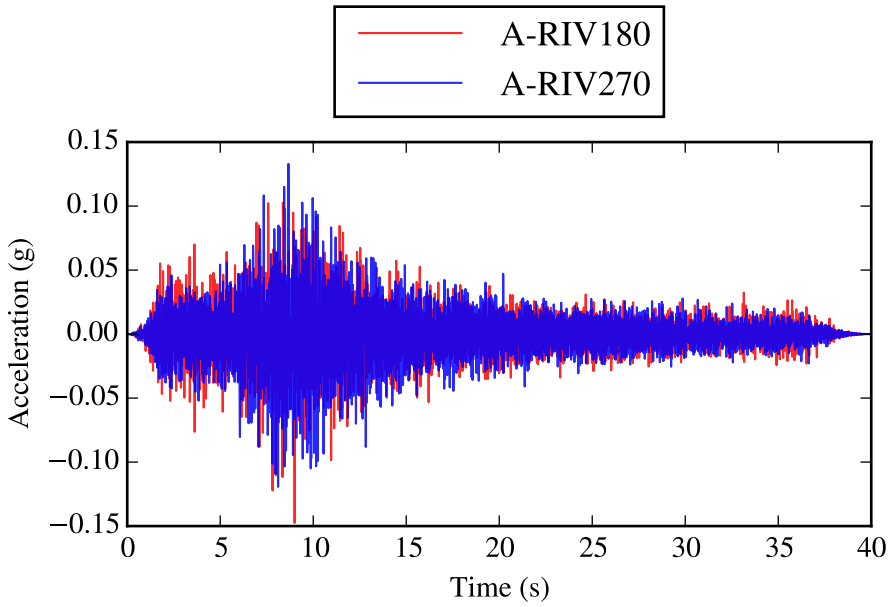


Figure F.211: CEUS 16: *A-RIV180* and *A-RIV270*.

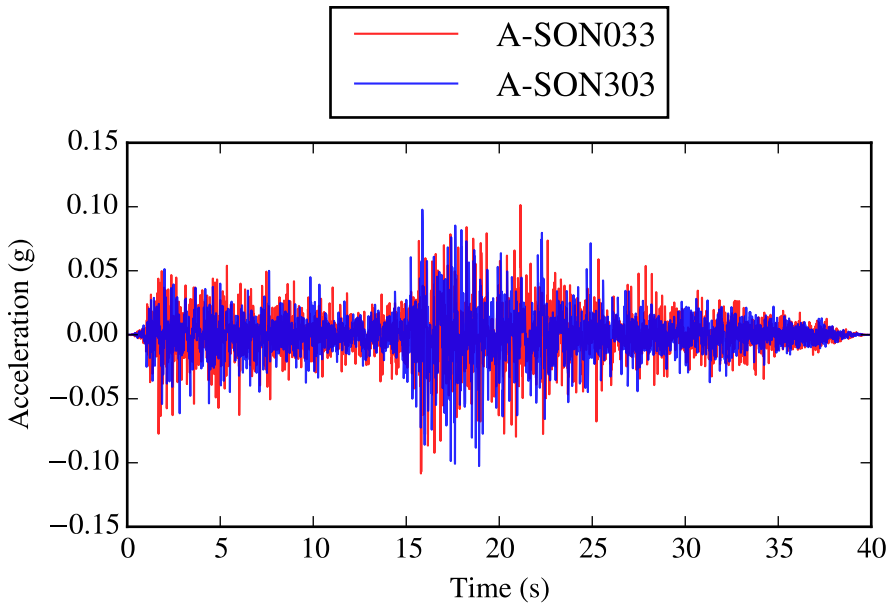


Figure F.212: CEUS 17: *A-SON033* and *A-SON303*.

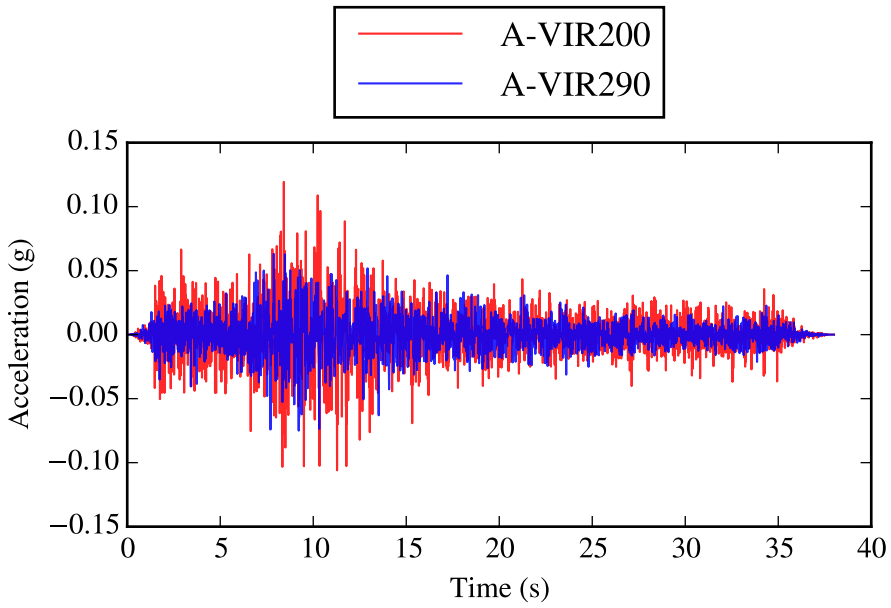


Figure F.213: CEUS 18: *A-VIR200* and *A-VIR290*.

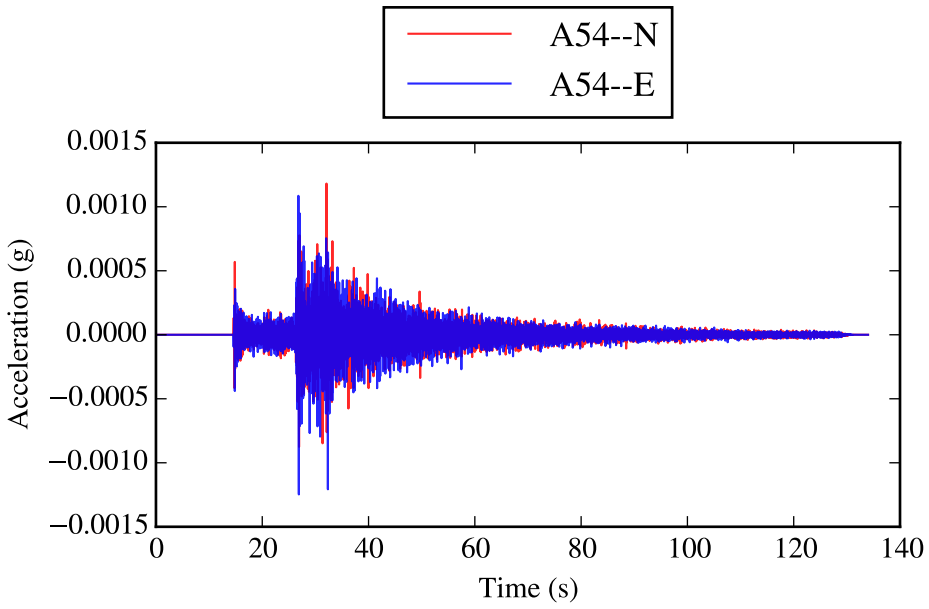


Figure F.214: CEUS 19: *A54–N* and *A54–E*.

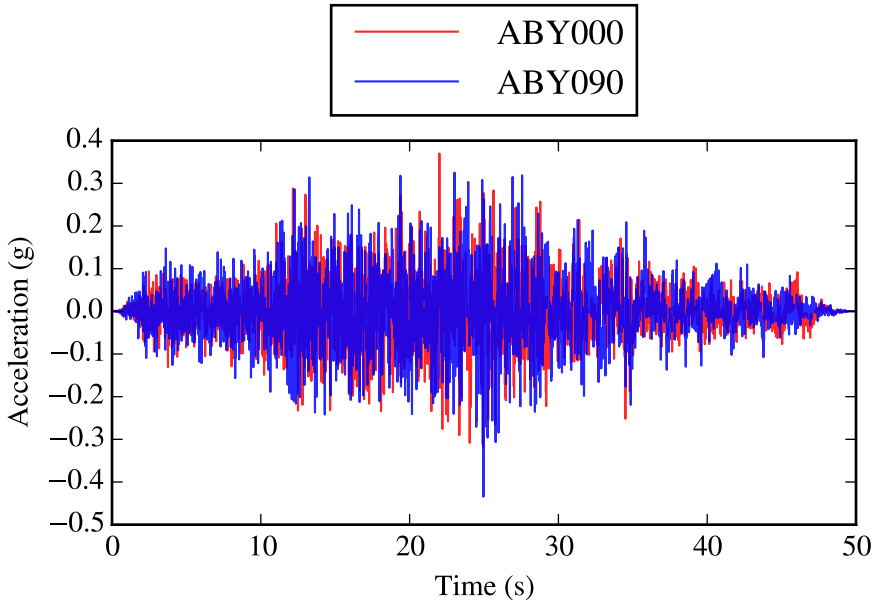


Figure F.215: CEUS 20: *ABY000* and *ABY090*.

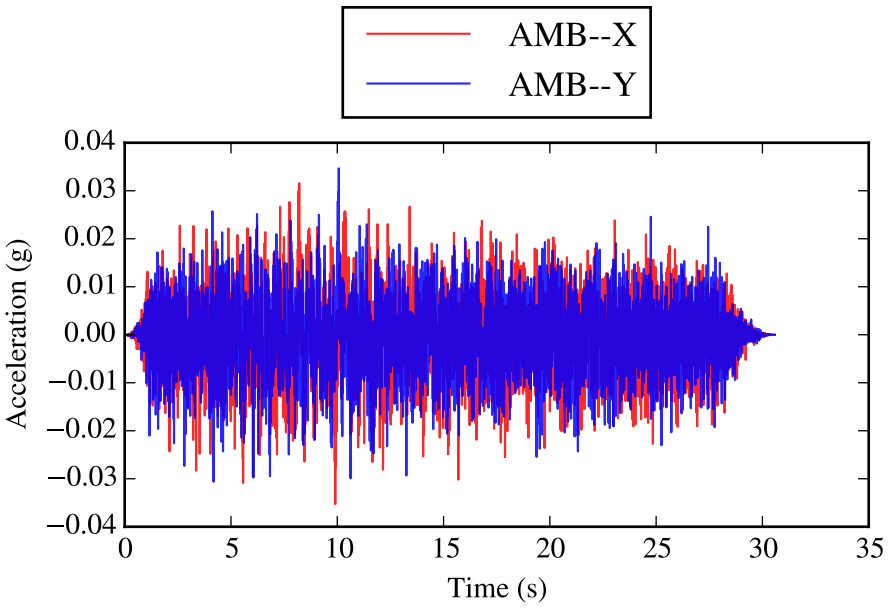


Figure F.216: CEUS 21: *AMB–X* and *AMB–Y*.

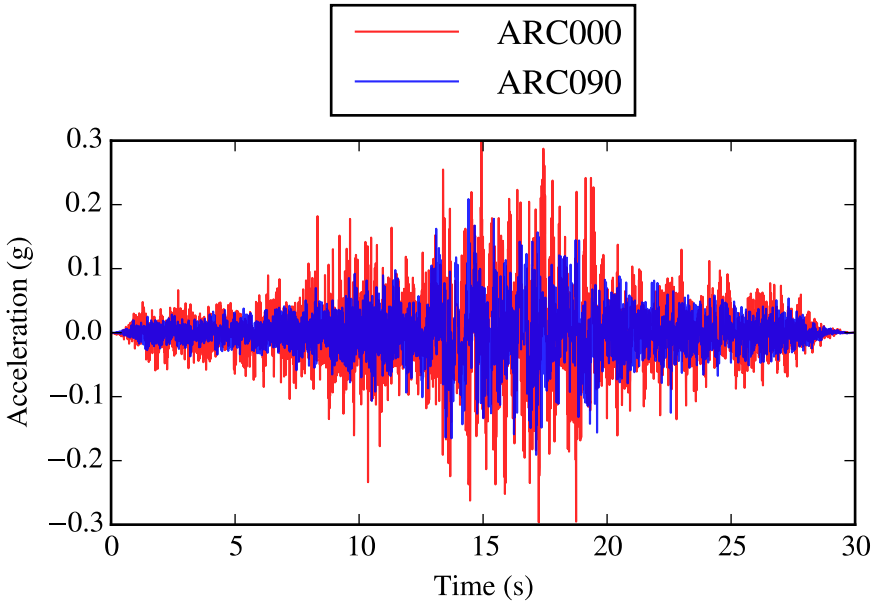


Figure F.217: CEUS 22: *ARC000* and *ARC090*.

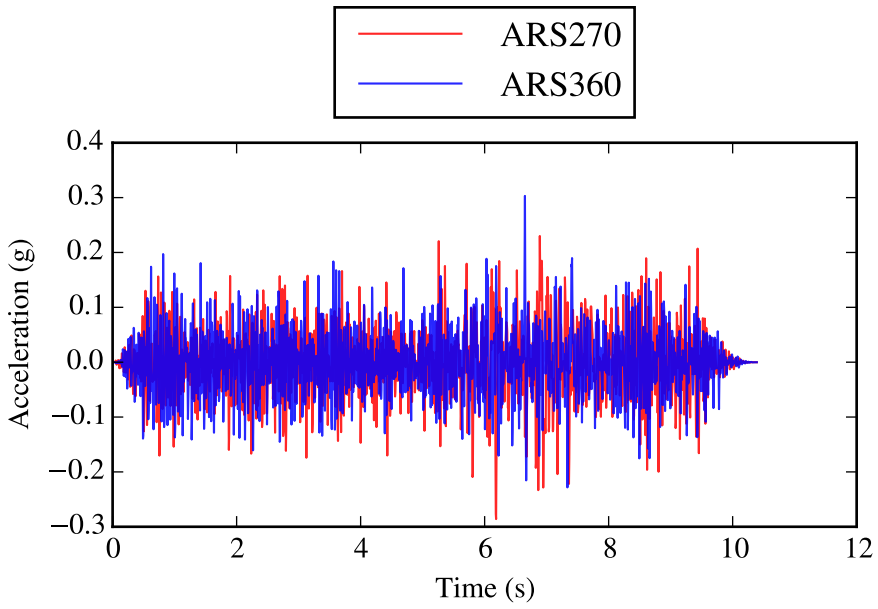


Figure F.218: CEUS 23: *ARS270* and *ARS360*.

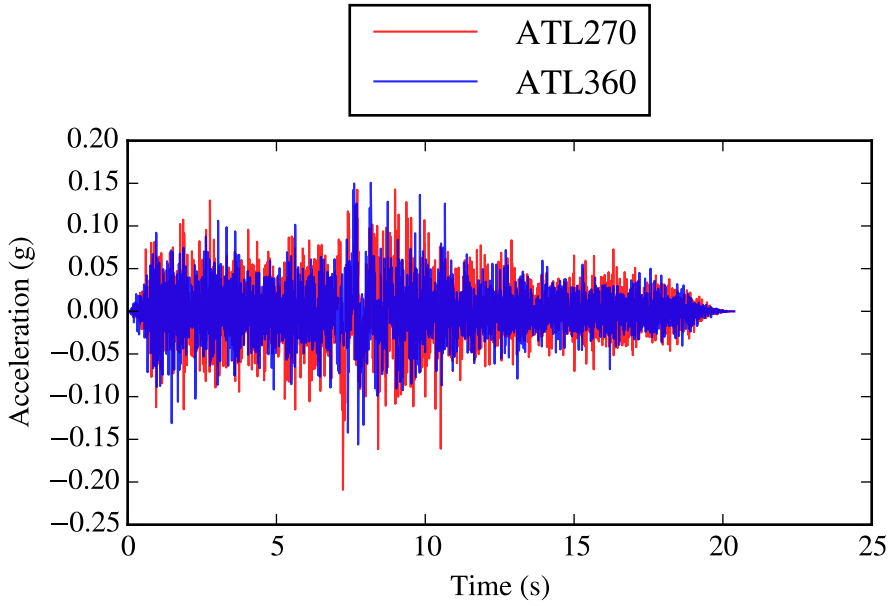


Figure F.219: CEUS 24: *ATL270* and *ATL360*.

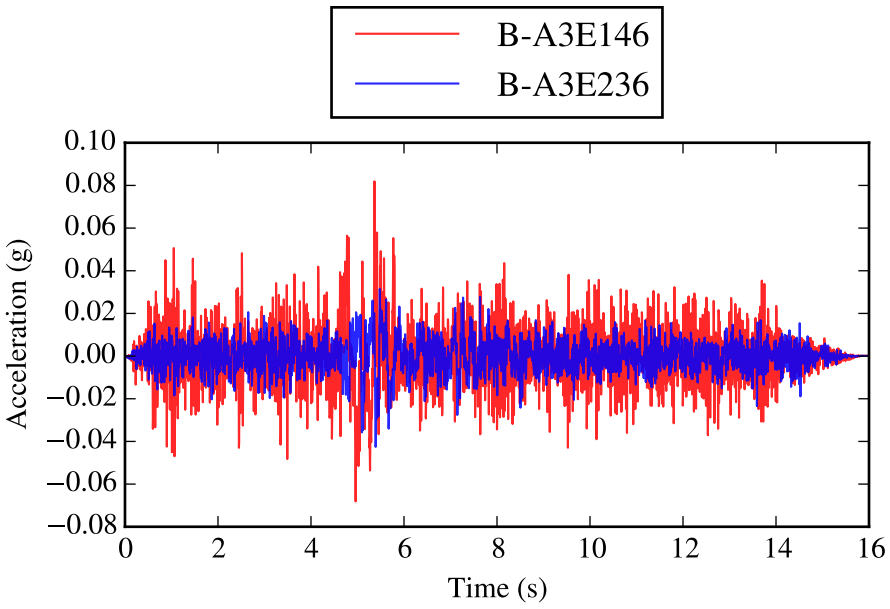


Figure F.220: CEUS 25: *B-A3E146* and *B-A3E236*.

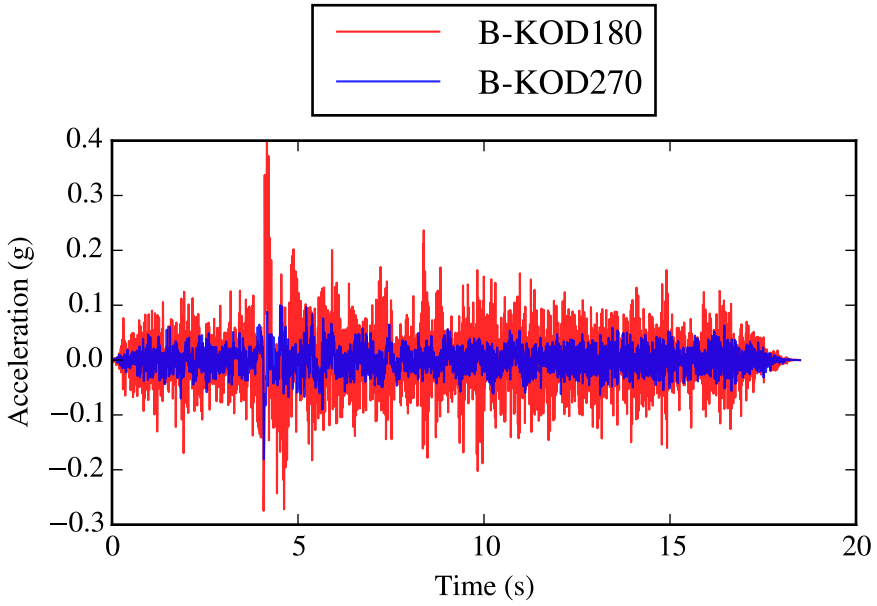


Figure F.221: CEUS 26: *B-KOD180* and *B-KOD270*.

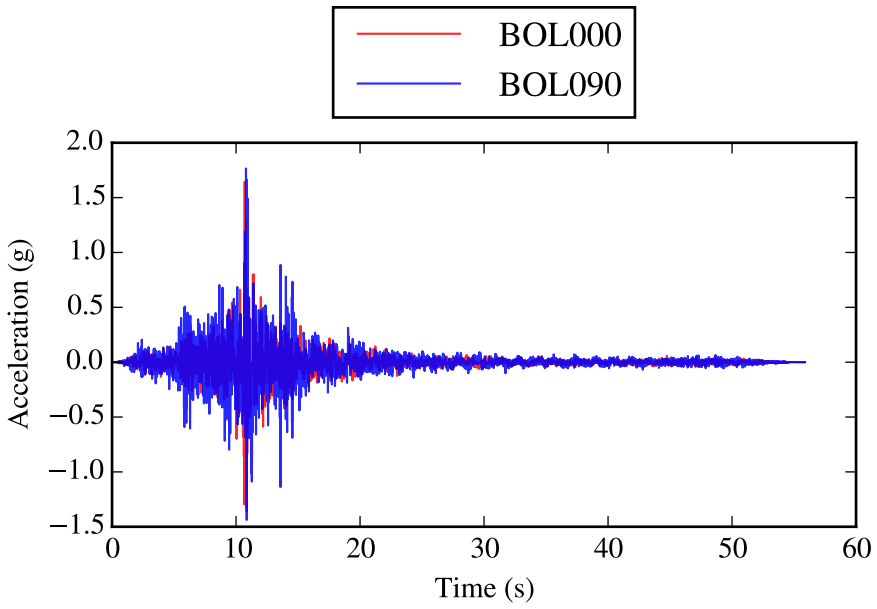


Figure F.222: CEUS 27: *BOL000* and *BOL090*.

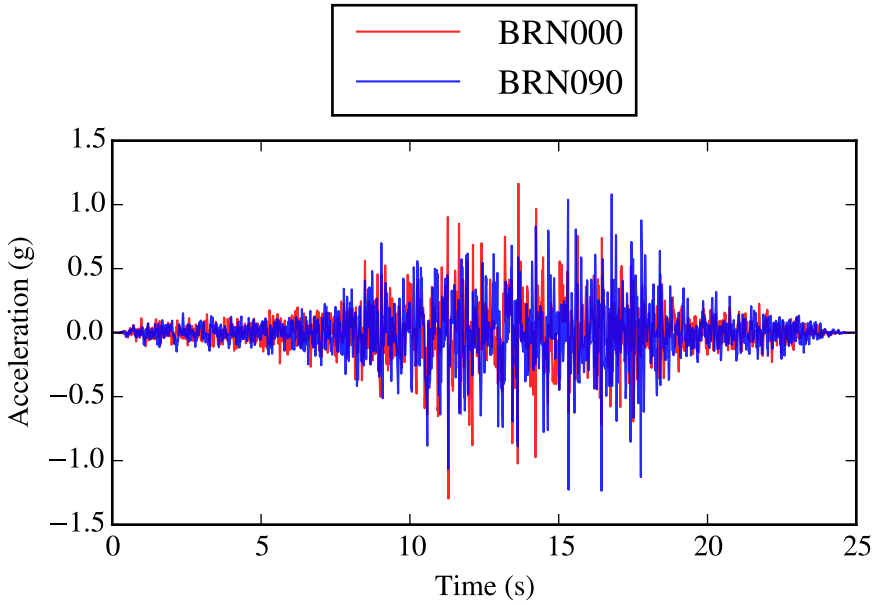


Figure F.223: CEUS 28: *BRN000* and *BRN090*.

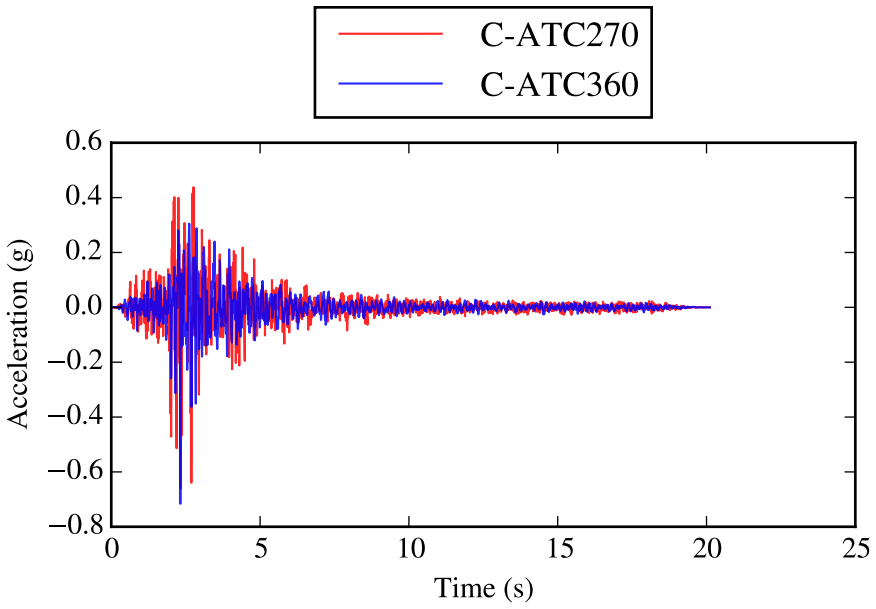


Figure F.224: CEUS 29: *C-ATC270* and *C-ATC360*.

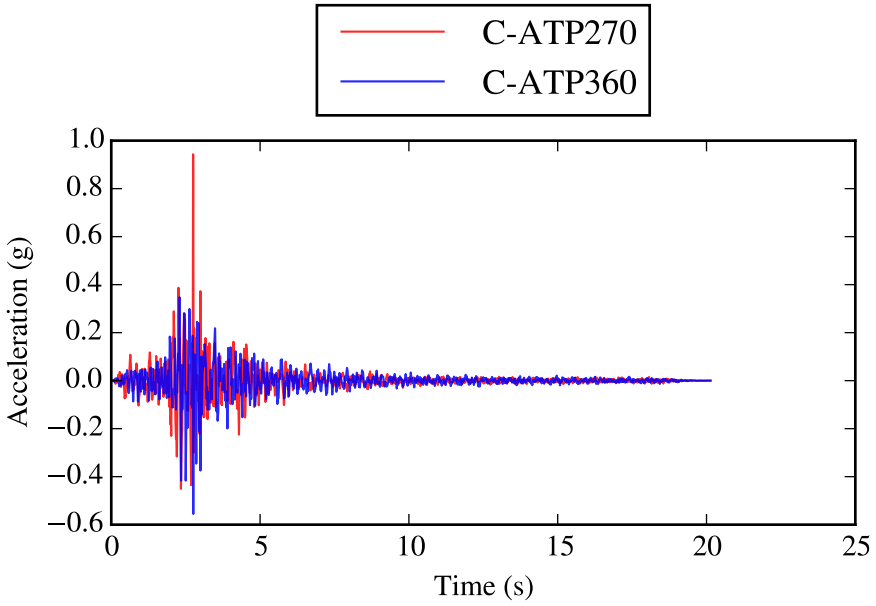


Figure F.225: CEUS 30: *C-ATP270* and *C-ATP360*.

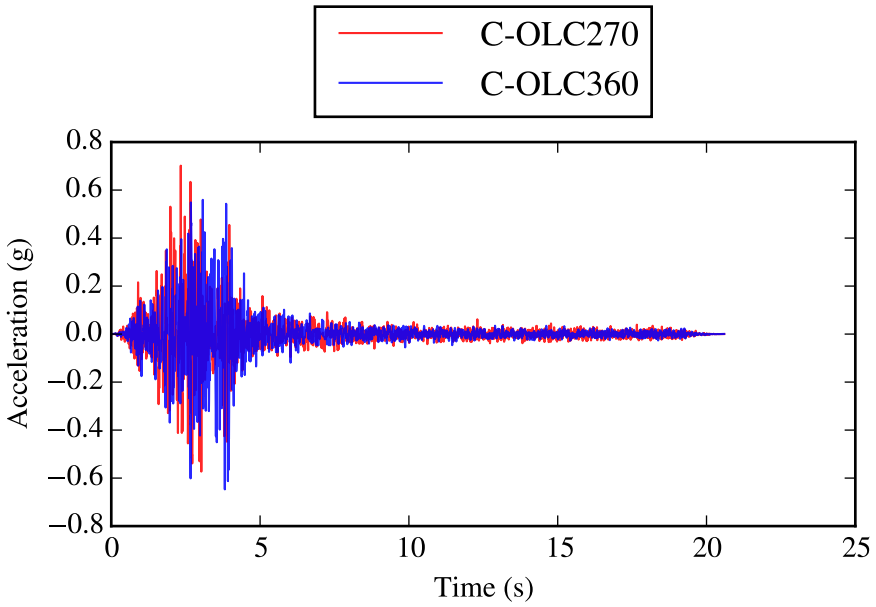


Figure F.226: CEUS 31: *C-OLC270* and *C-OLC360*.

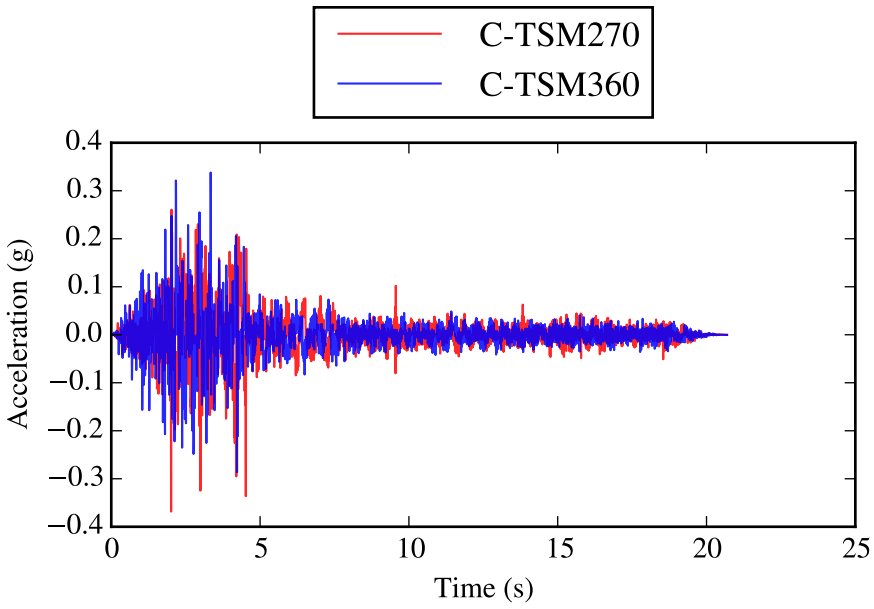


Figure F.227: CEUS 32: *C-TSM270* and *C-TSM360*.

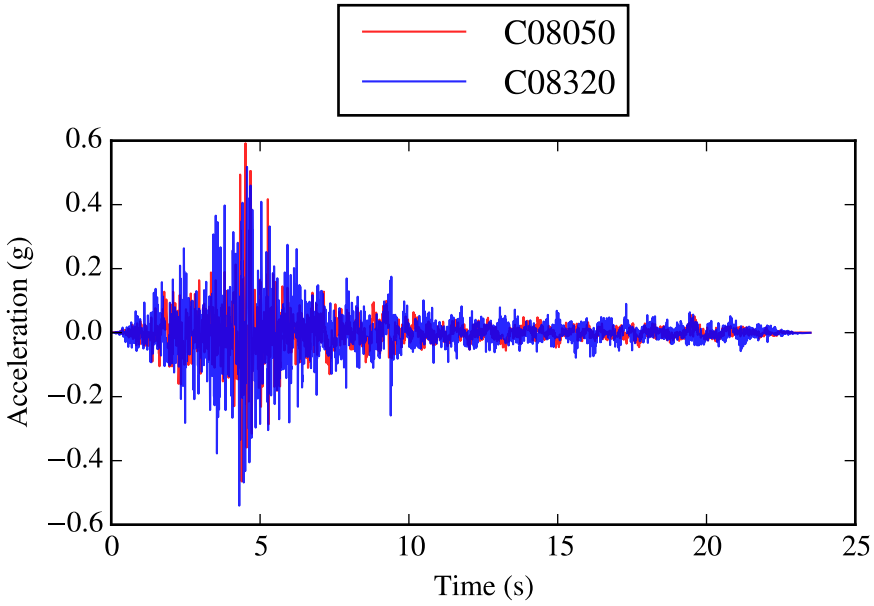


Figure F.228: CEUS 33: *C08050* and *C08320*.

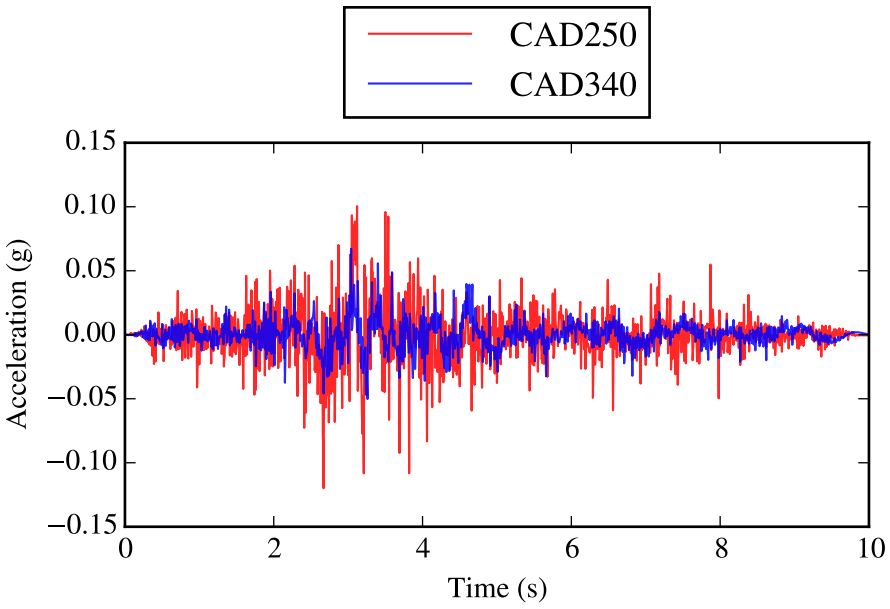


Figure F.229: CEUS 34: *CAD250* and *CAD340*.

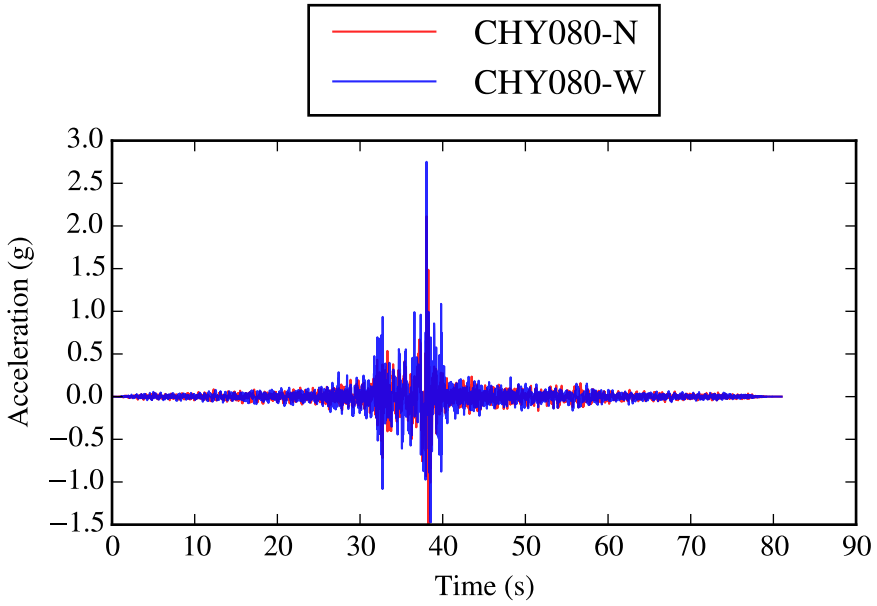


Figure F.230: CEUS 35: *CHY080-N* and *CHY080-W*.

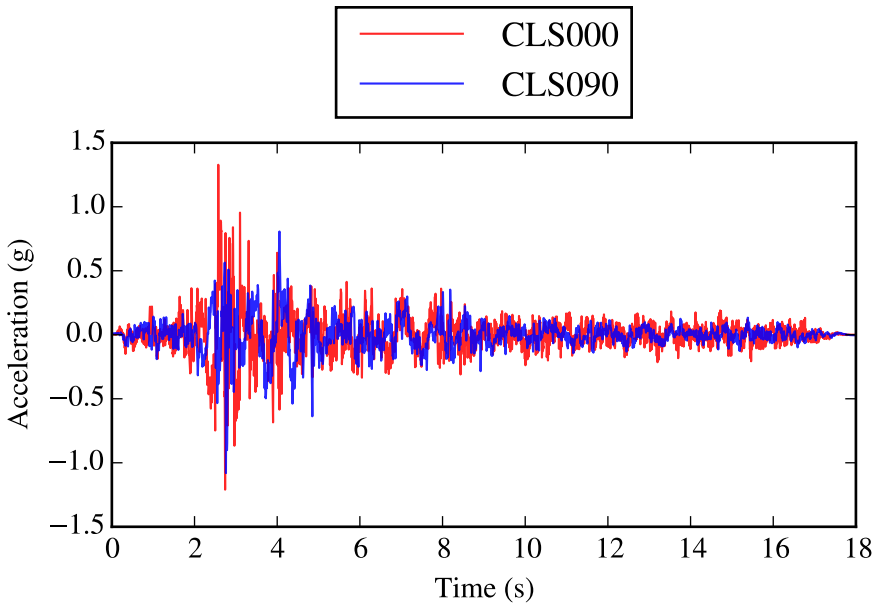


Figure F.231: CEUS 36: *CLS000* and *CLS090*.

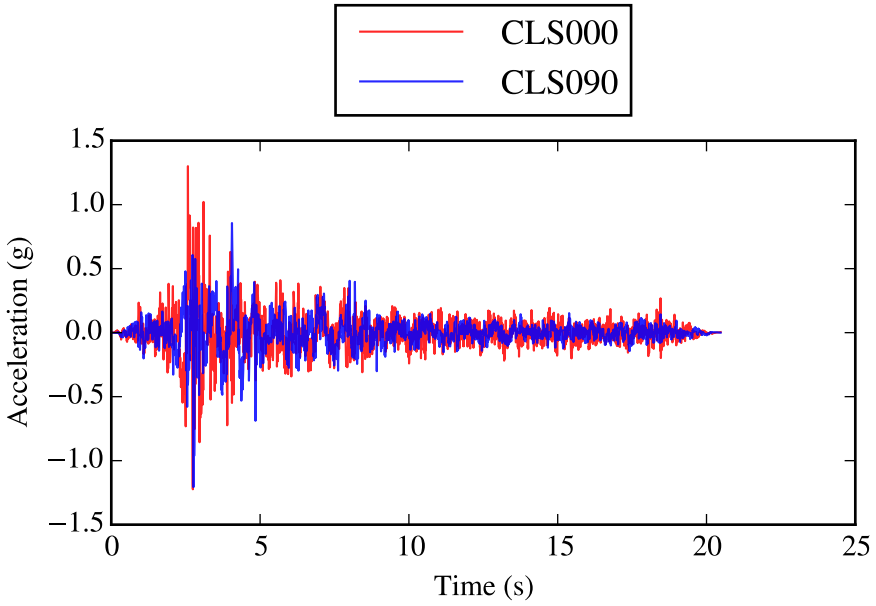


Figure F.232: CEUS 37: *CLS000* and *CLS090*.

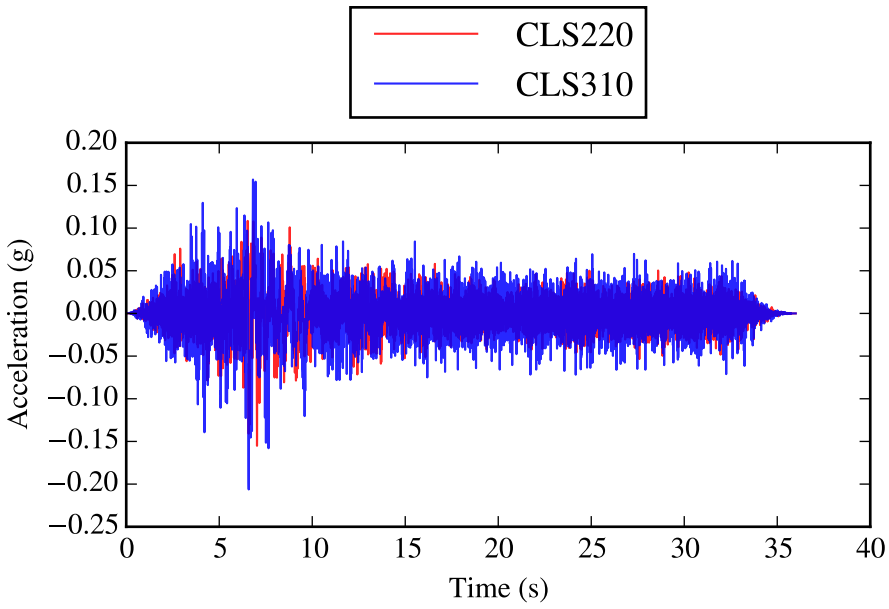


Figure F.233: CEUS 38: *CLS220* and *CLS310*.

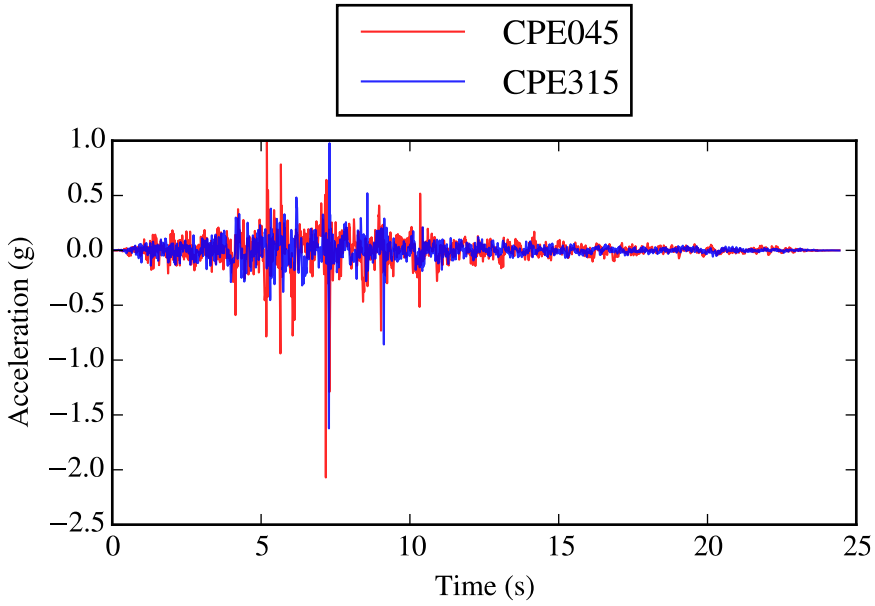


Figure F.234: CEUS 39: *CPE045* and *CPE315*.

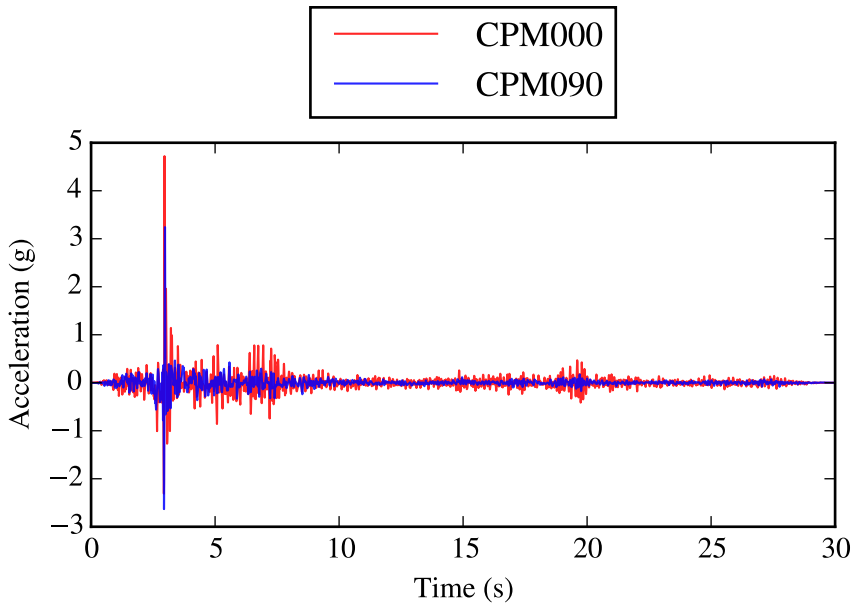


Figure F.235: CEUS 40: *CPM000* and *CPM090*.

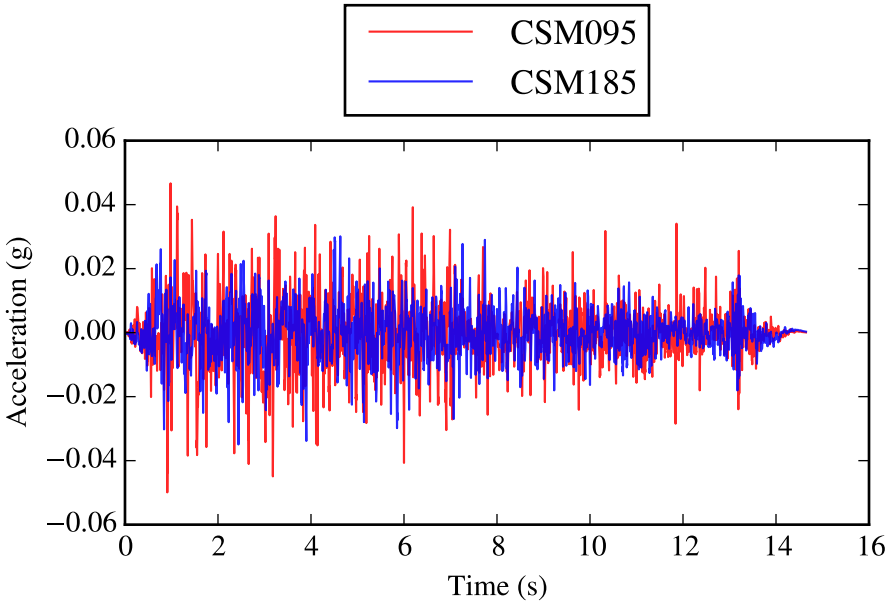


Figure F.236: CEUS 41: *CSM095* and *CSM185*.

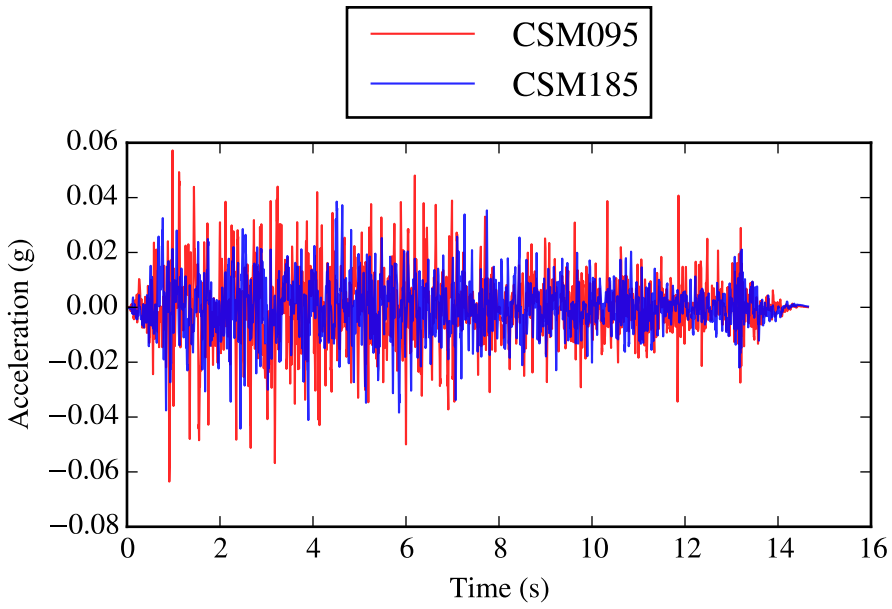


Figure F.237: CEUS 42: *CSM095* and *CSM185*.

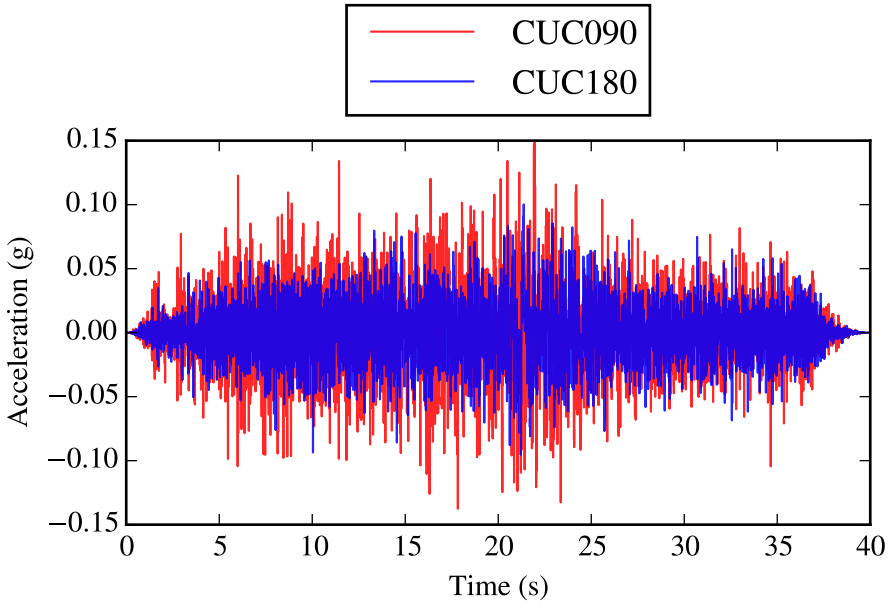


Figure F.238: CEUS 43: *CUC090* and *CUC180*.

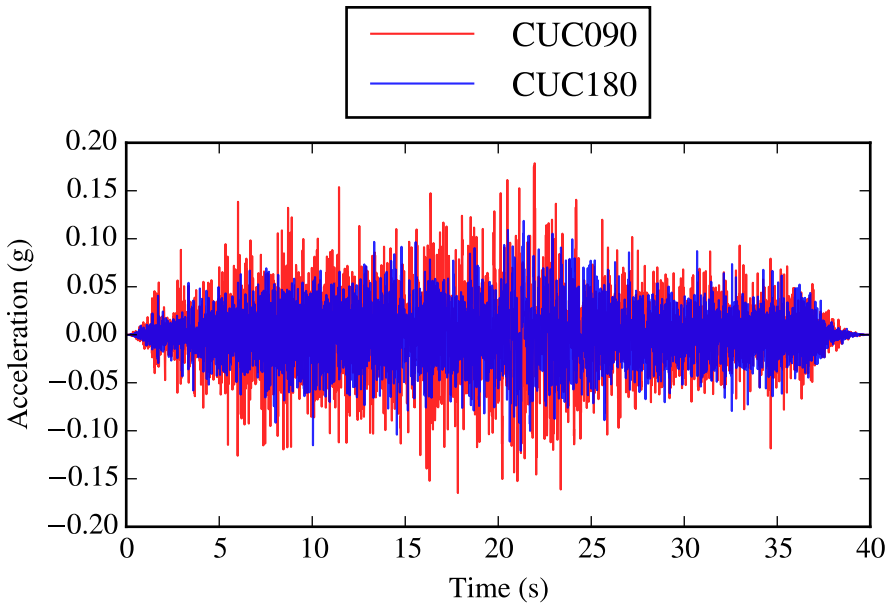


Figure F.239: CEUS 44: *CUC090* and *CUC180*.

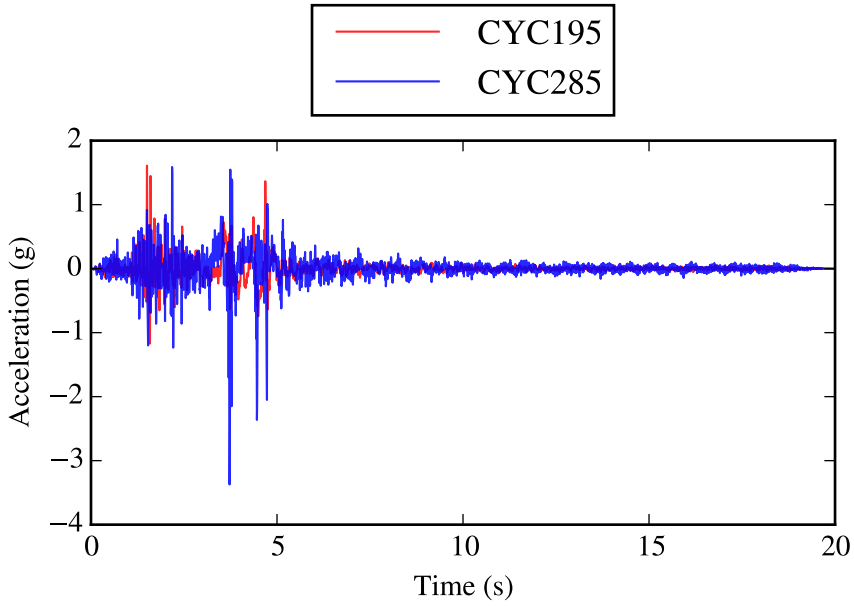


Figure F.240: CEUS 45: *CYC195* and *CYC285*.

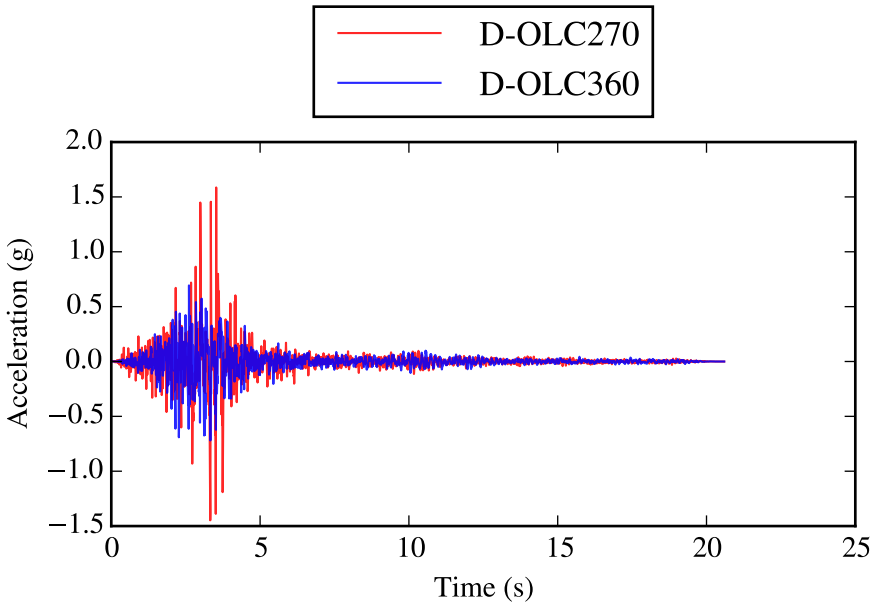


Figure F.241: CEUS 46: *D-OLC270* and *D-OLC360*.

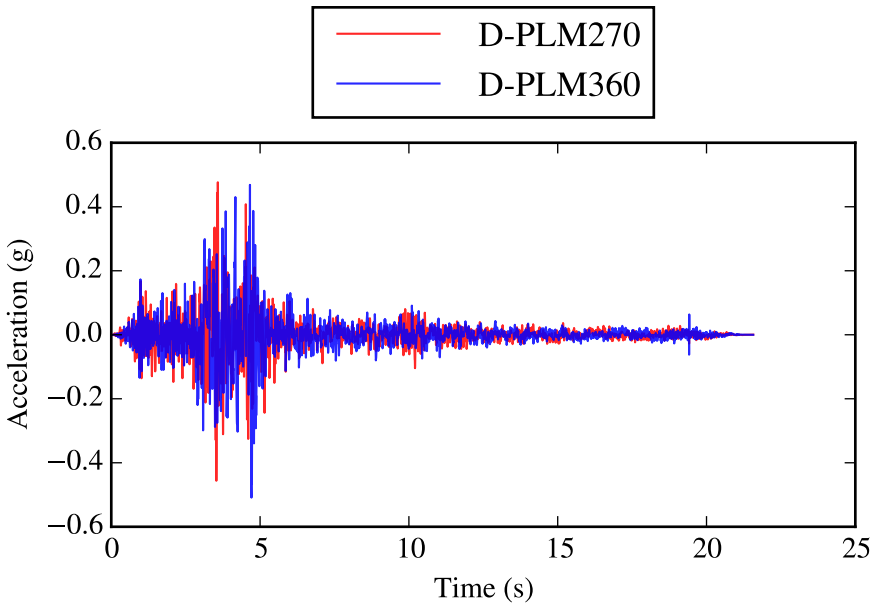


Figure F.242: CEUS 47: *D-PLM270* and *D-PLM360*.

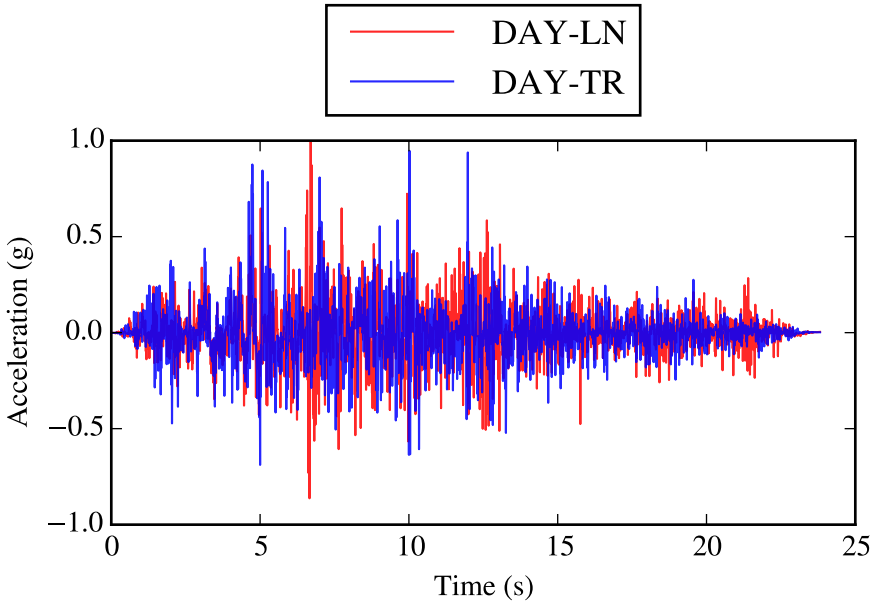


Figure F.243: CEUS 48: *DAY-LN* and *DAY-TR*.

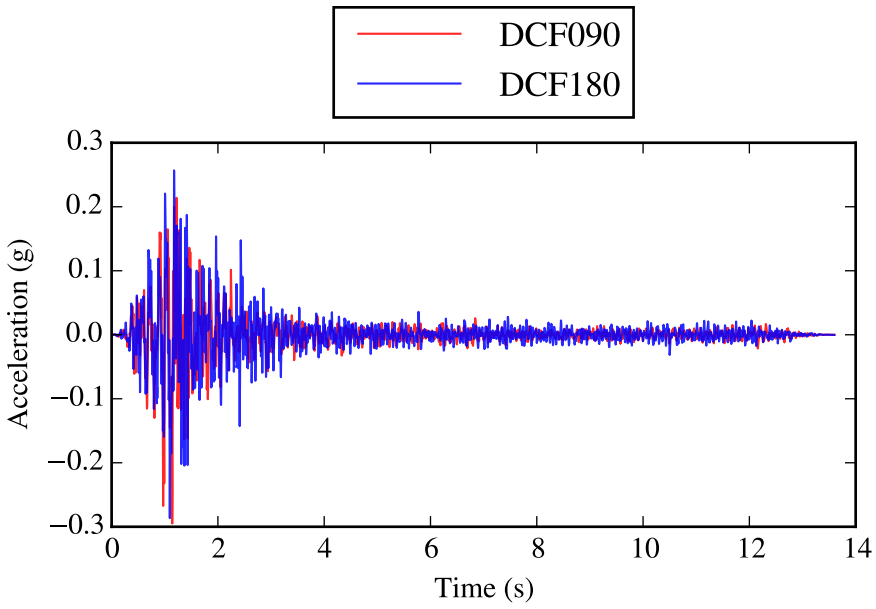


Figure F.244: CEUS 49: *DCF090* and *DCF180*.

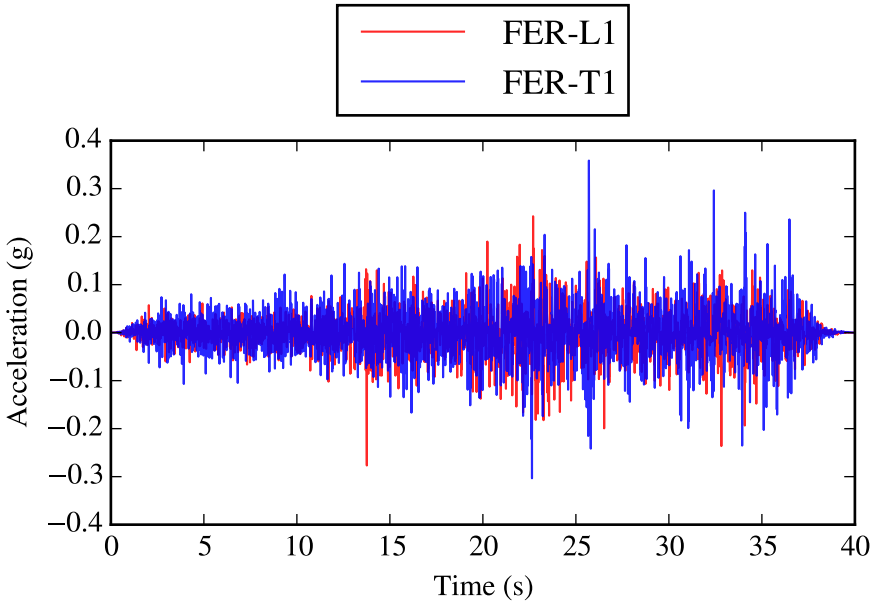


Figure F.245: CEUS 50: *FER-L1* and *FER-T1*.

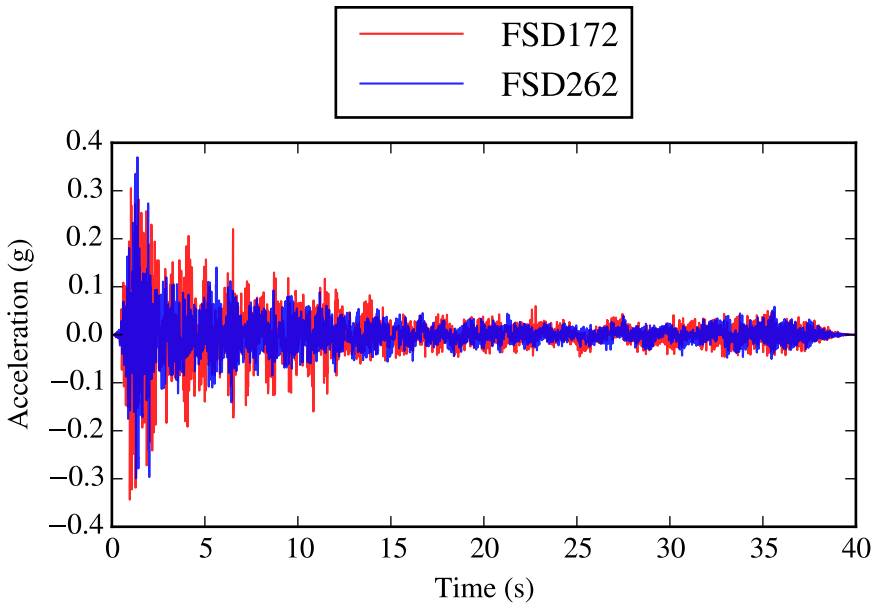


Figure F.246: CEUS 51: *FSD172* and *FSD262*.

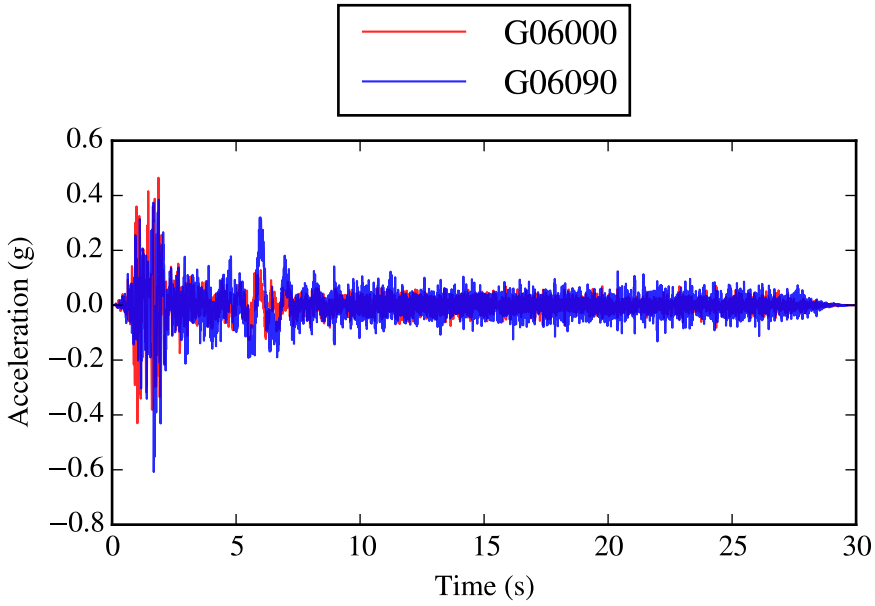


Figure F.247: CEUS 52: *G06000* and *G06090*.

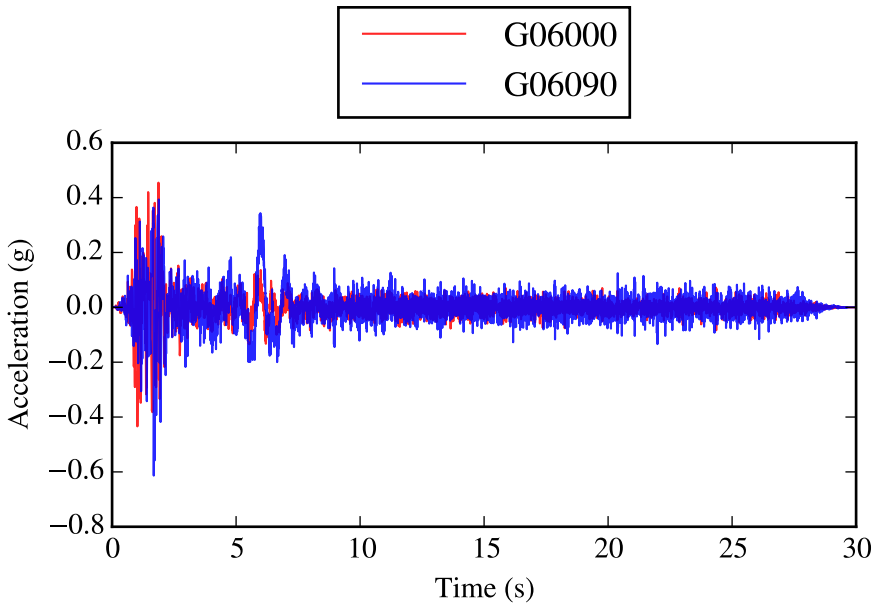


Figure F.248: CEUS 53: *G06000* and *G06090*.

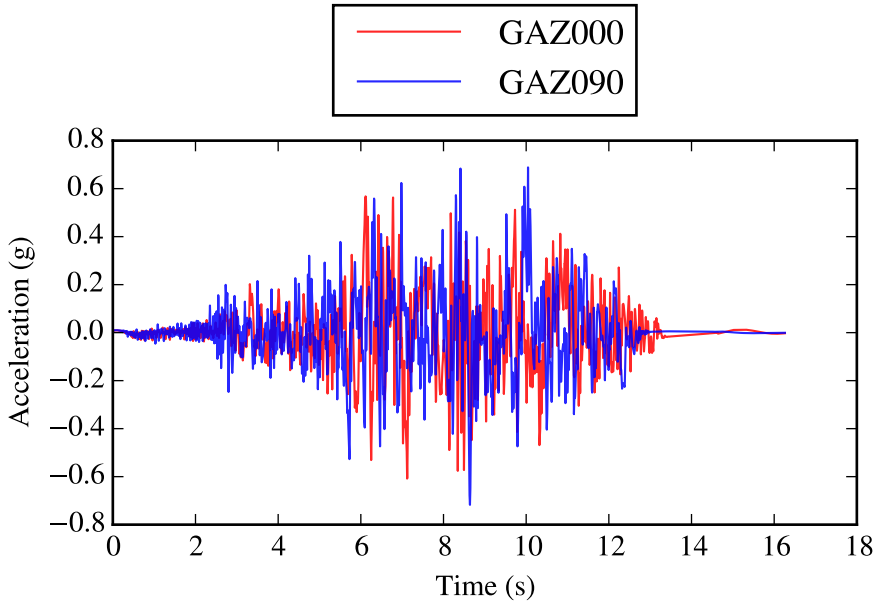


Figure F.249: CEUS 54: *GAZ000* and *GAZ090*.

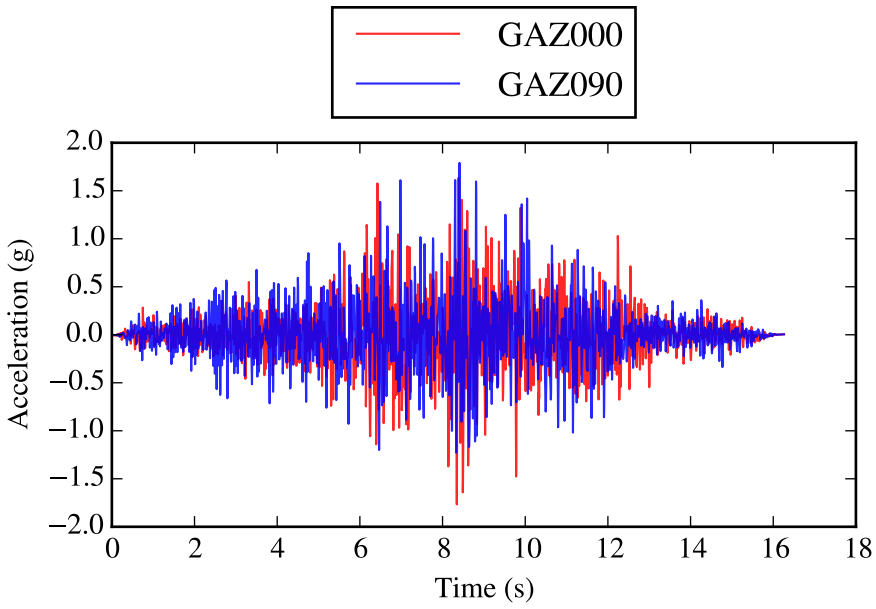


Figure F.250: CEUS 55: *GAZ000* and *GAZ090*.

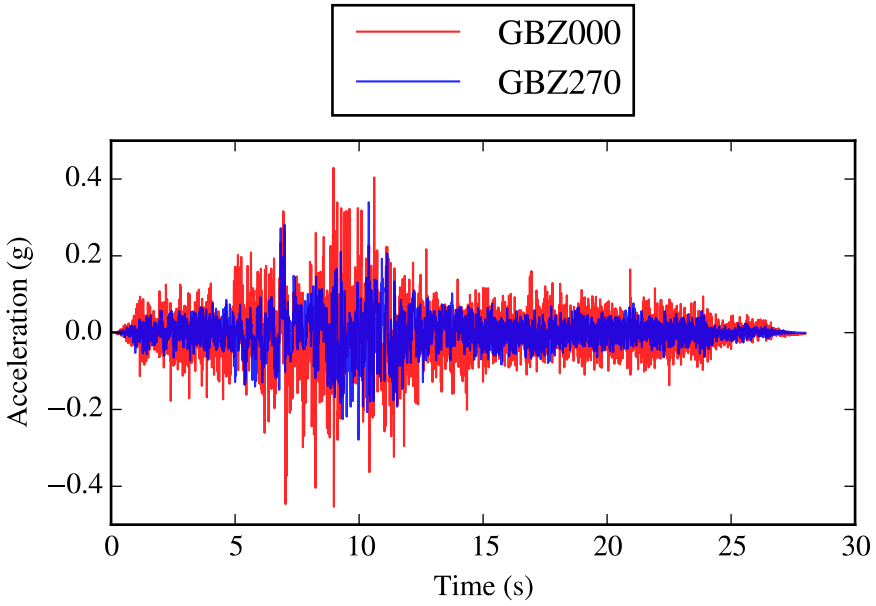


Figure F.251: CEUS 56: *GBZ000* and *GBZ270*.

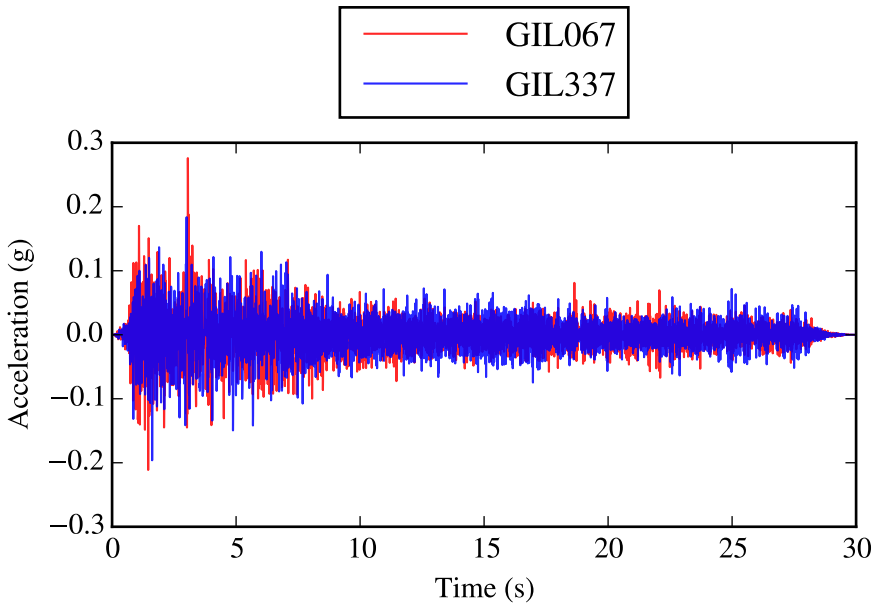


Figure F.252: CEUS 57: *GIL067* and *GIL337*.

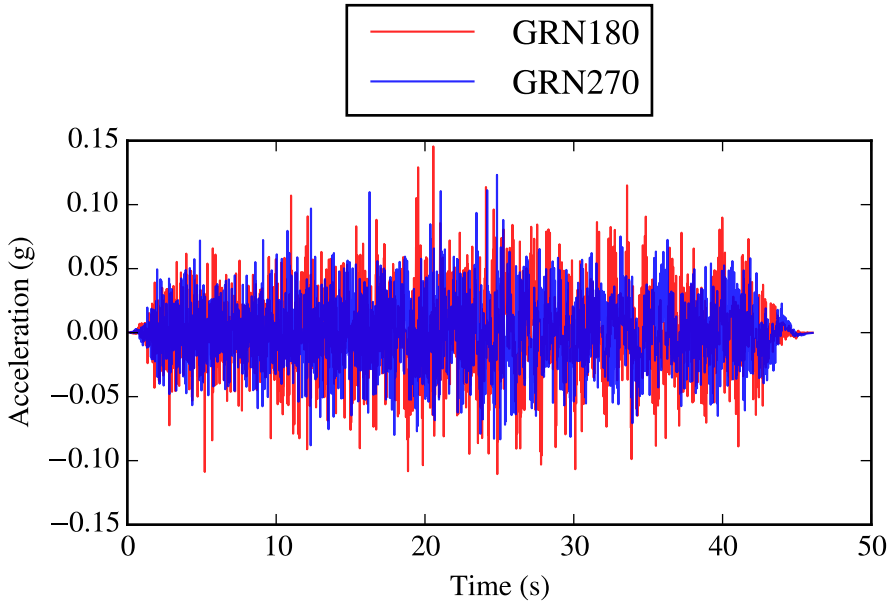


Figure F.253: CEUS 58: *GRN180* and *GRN270*.

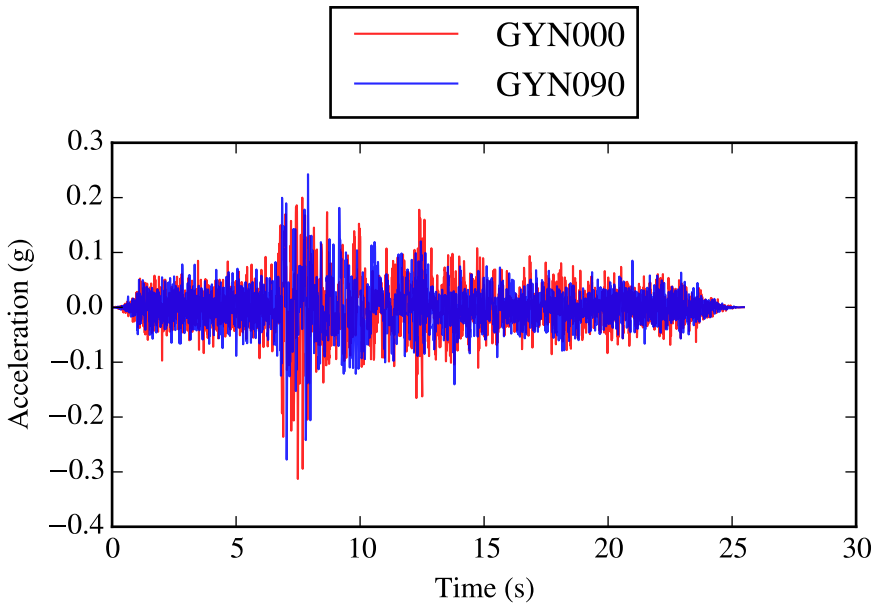


Figure F.254: CEUS 59: *GYN000* and *GYN090*.

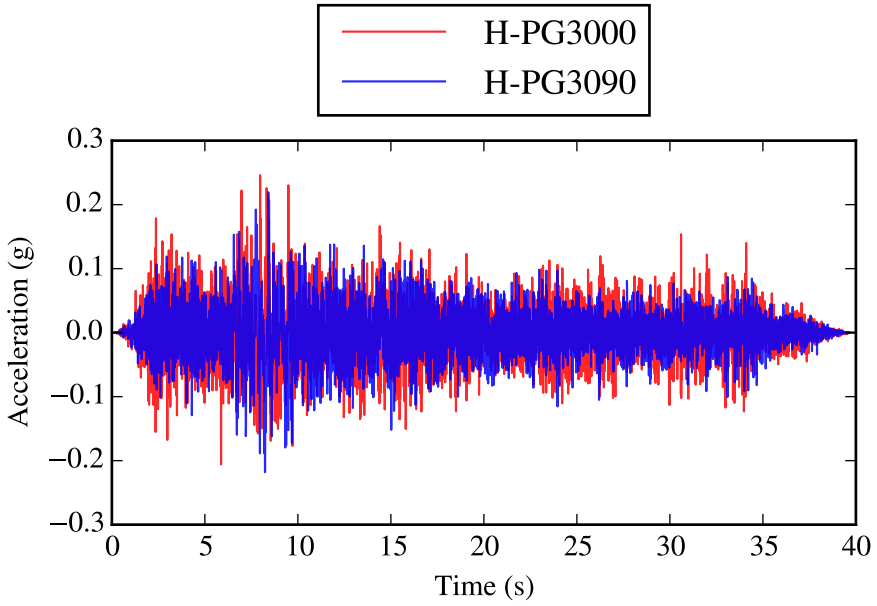


Figure F.255: CEUS 60: *H-PG3000* and *H-PG3090*.

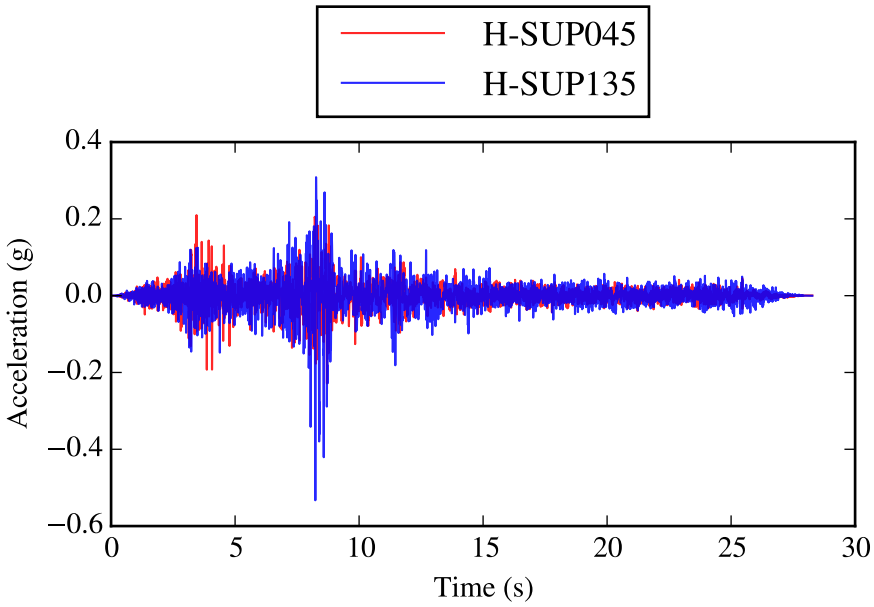


Figure F.256: CEUS 61: *H-SUP045* and *H-SUP135*.

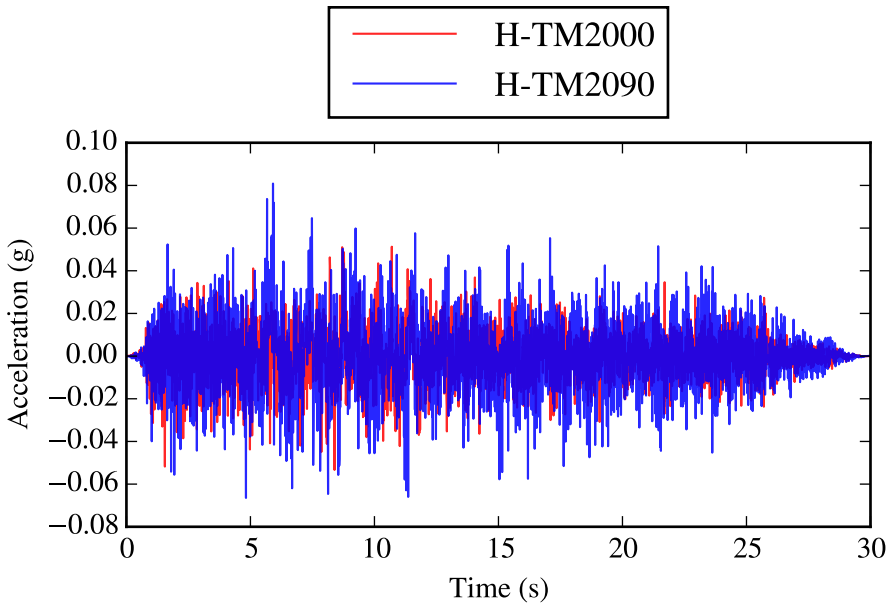


Figure F.257: CEUS 62: *H-TM2000* and *H-TM2090*.

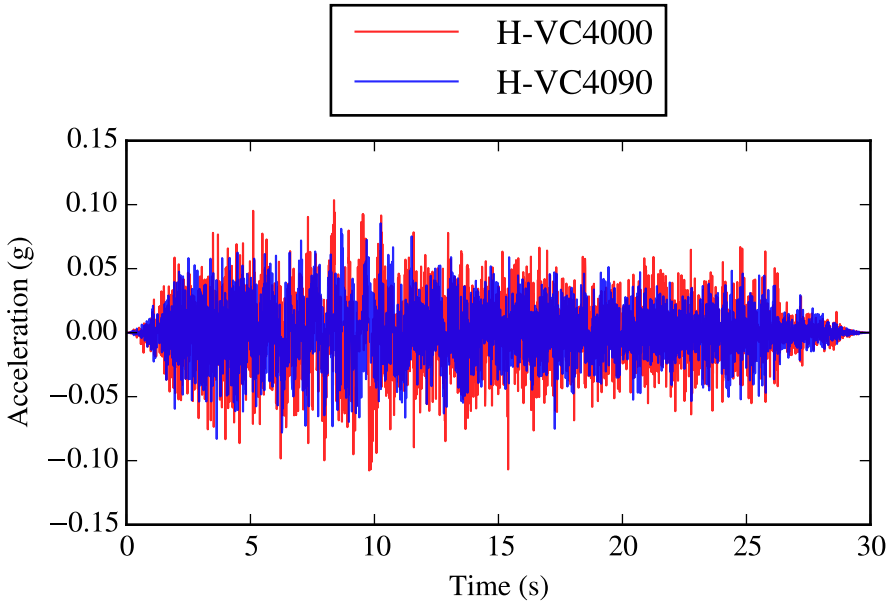


Figure F.258: CEUS 63: *H-VC4000* and *H-VC4090*.

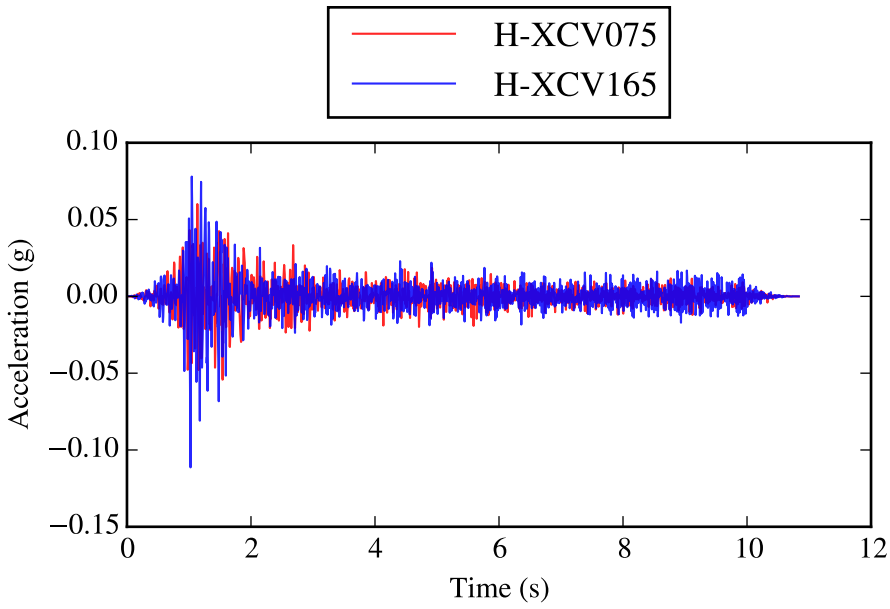


Figure F.259: CEUS 64: *H-XCV075* and *H-XCV165*.

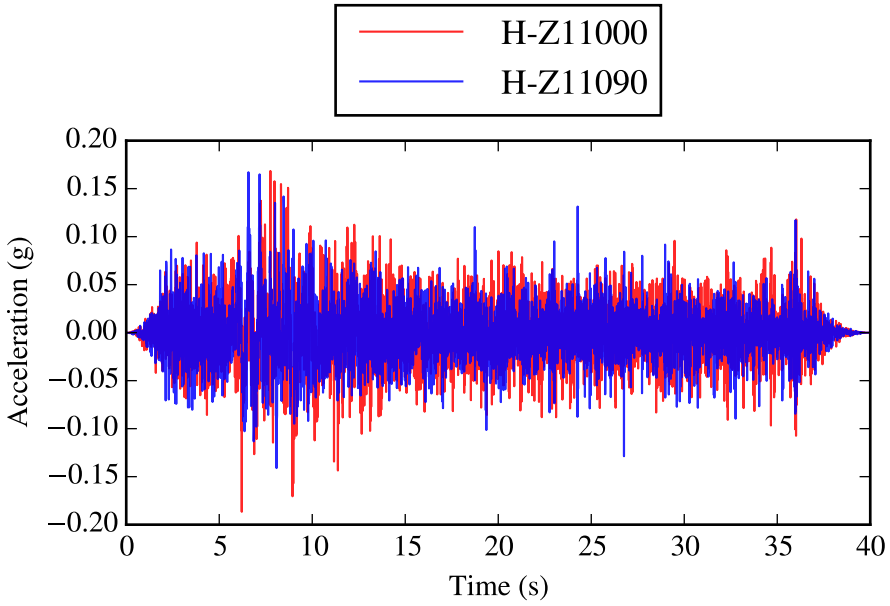


Figure F.260: CEUS 65: *H-Z11000* and *H-Z11090*.

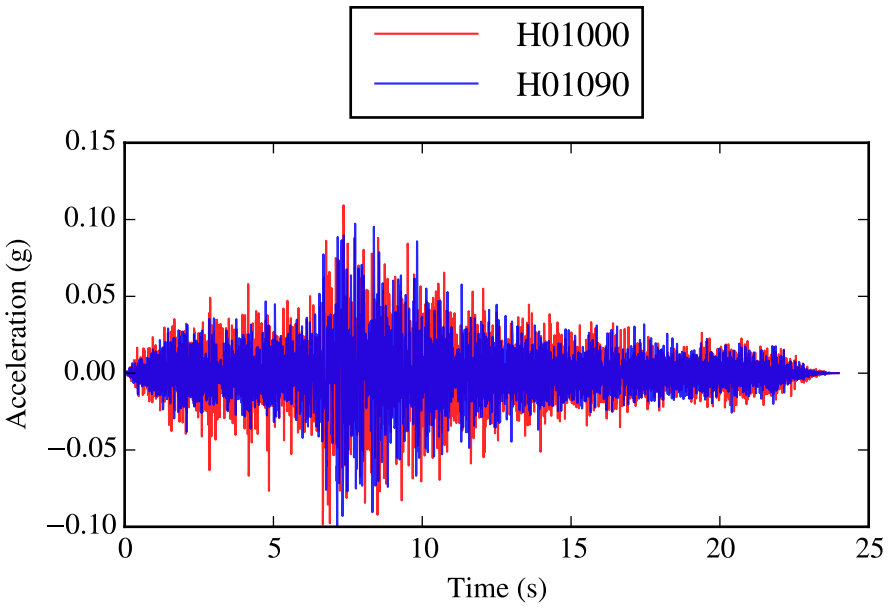


Figure F.261: CEUS 66: *H01000* and *H01090*.

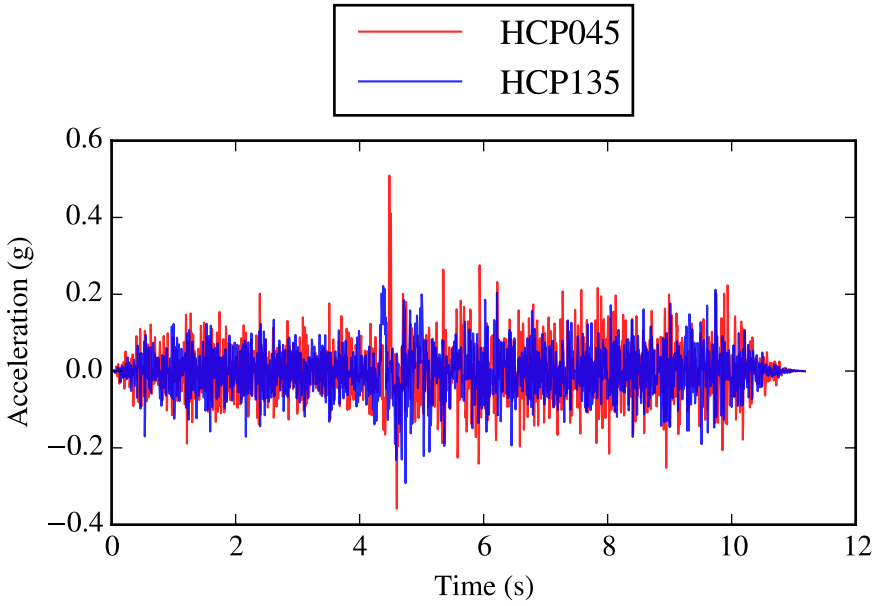


Figure F.262: CEUS 67: *HCP045* and *HCP135*.

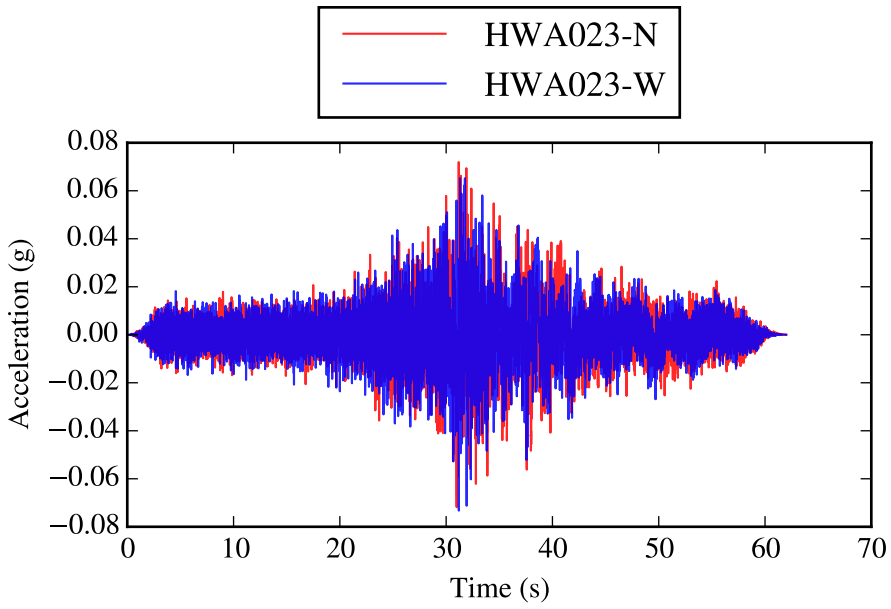


Figure F.263: CEUS 68: *HWA023-N* and *HWA023-W*.

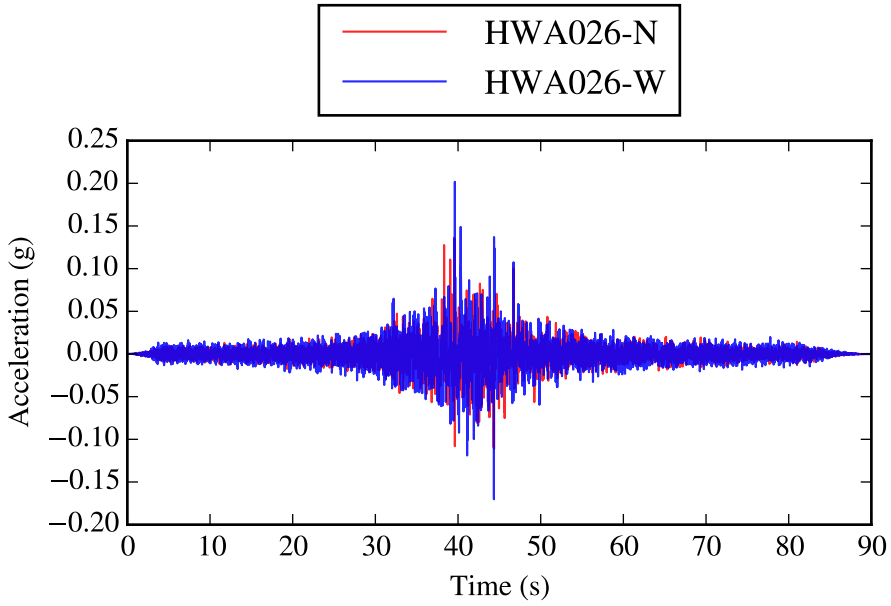


Figure F.264: CEUS 69: *HWA026-N* and *HWA026-W*.

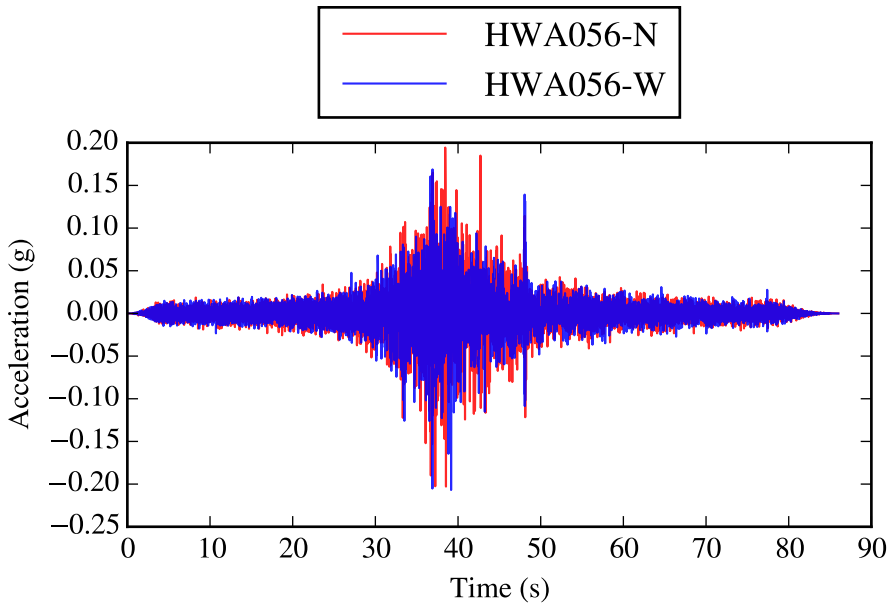


Figure F.265: CEUS 70: *HWA056-N* and *HWA056-W*.

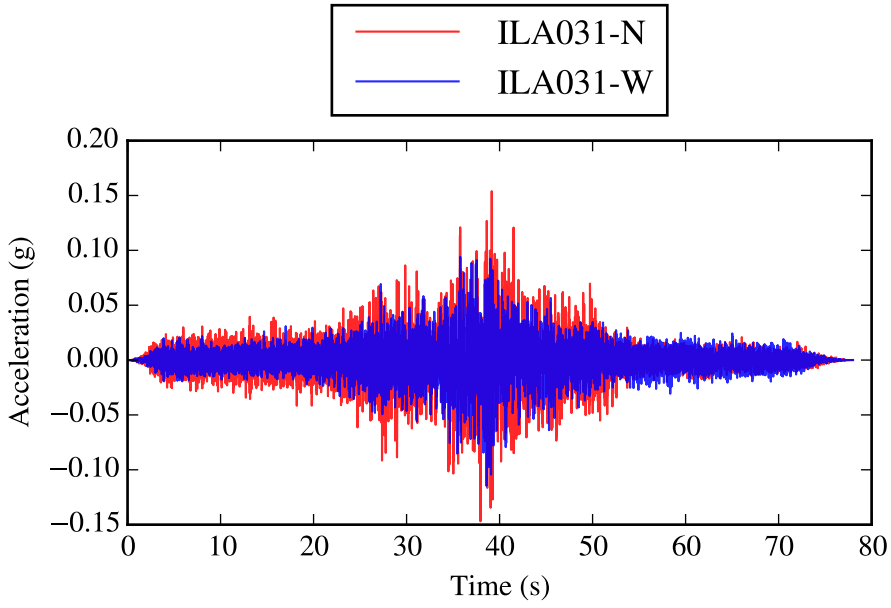


Figure F.266: CEUS 71: *ILA031-N* and *ILA031-W*.

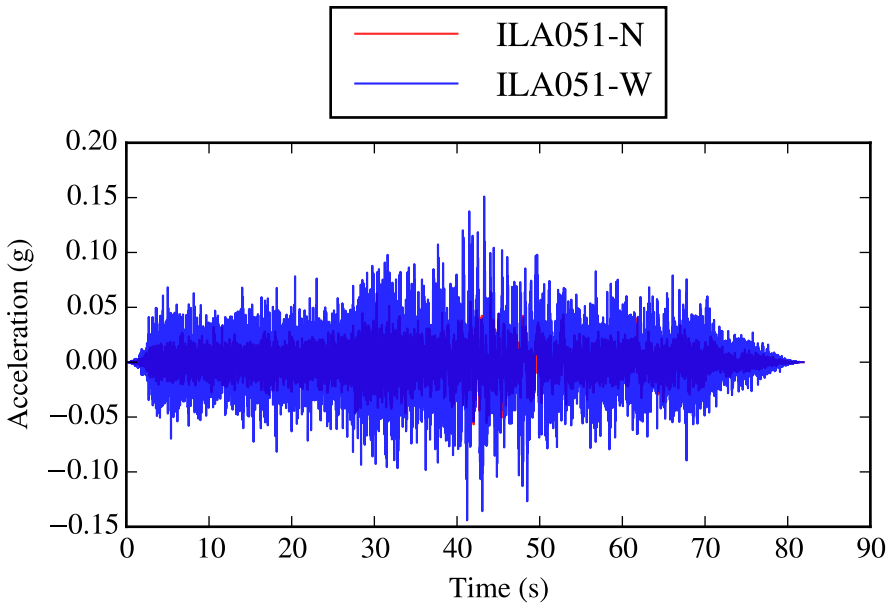


Figure F.267: CEUS 72: *ILA051-N* and *ILA051-W*.

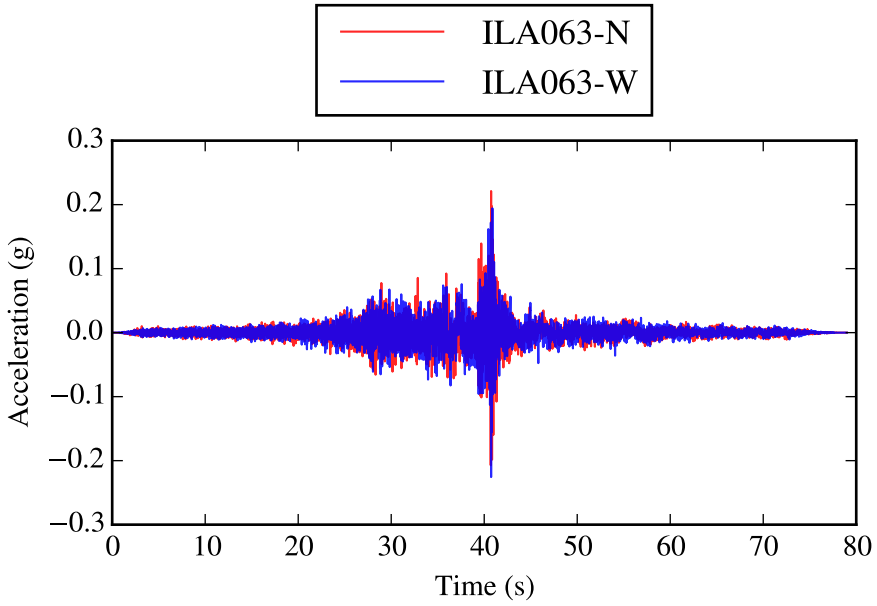


Figure F.268: CEUS 73: *ILA063-N* and *ILA063-W*.

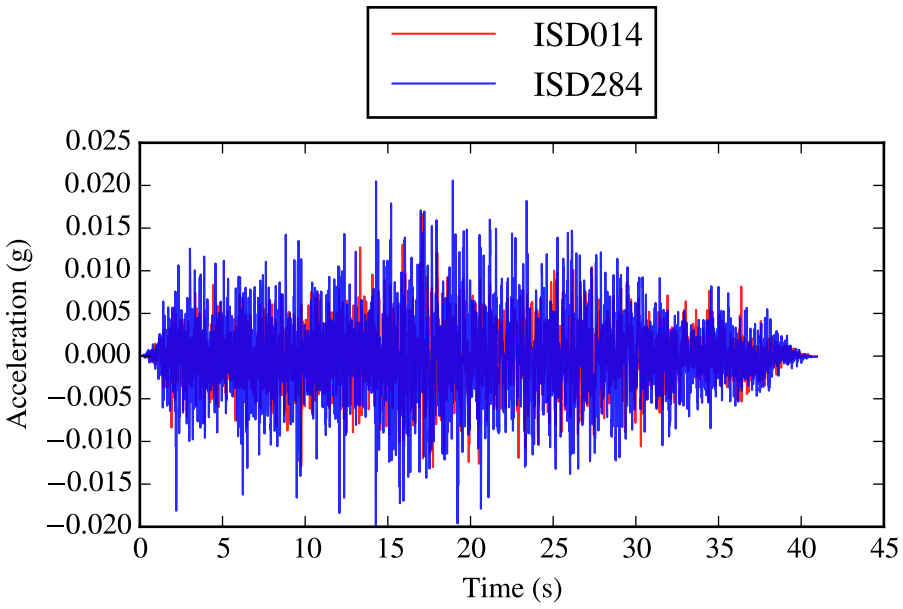


Figure F.269: CEUS 74: *ISD014* and *ISD284*.

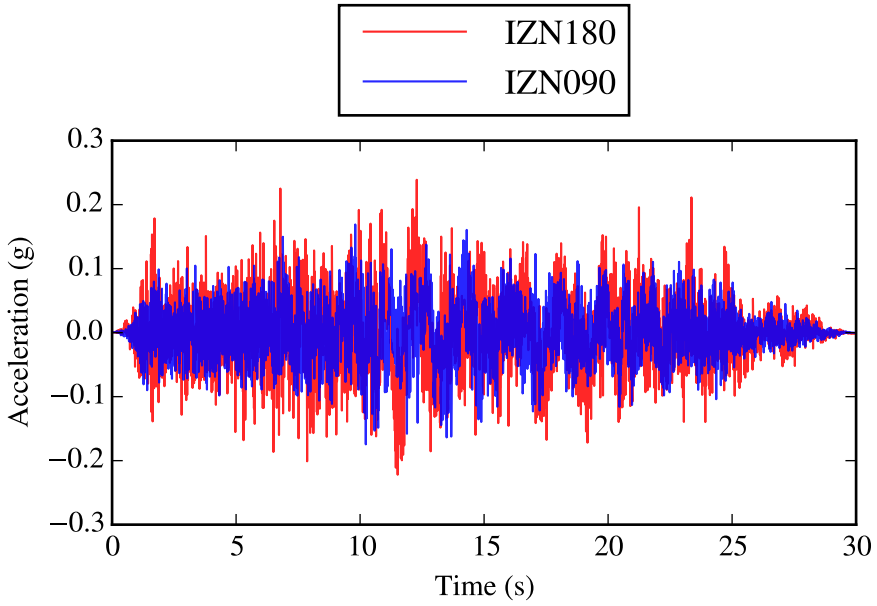


Figure F.270: CEUS 75: *IZN180* and *IZN090*.

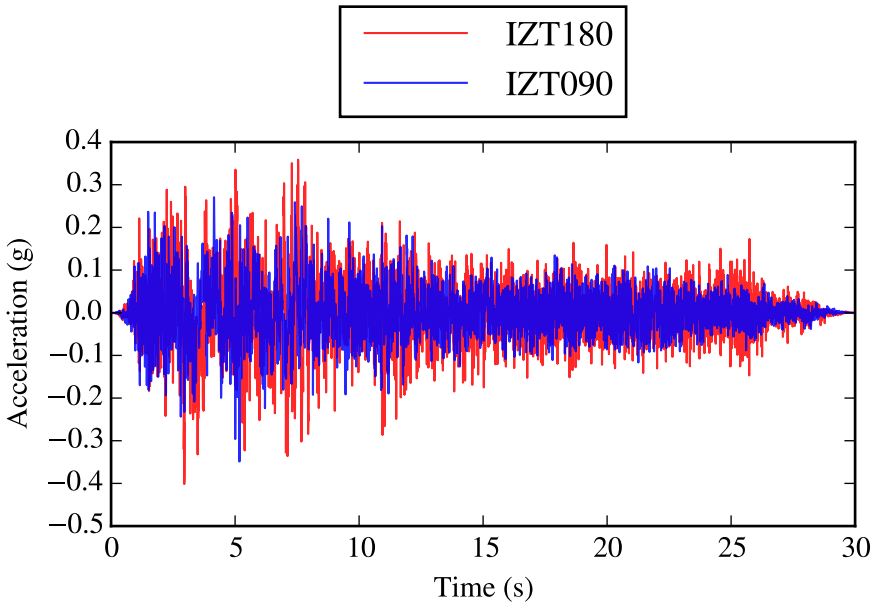


Figure F.271: CEUS 76: *IZT180* and *IZT090*.

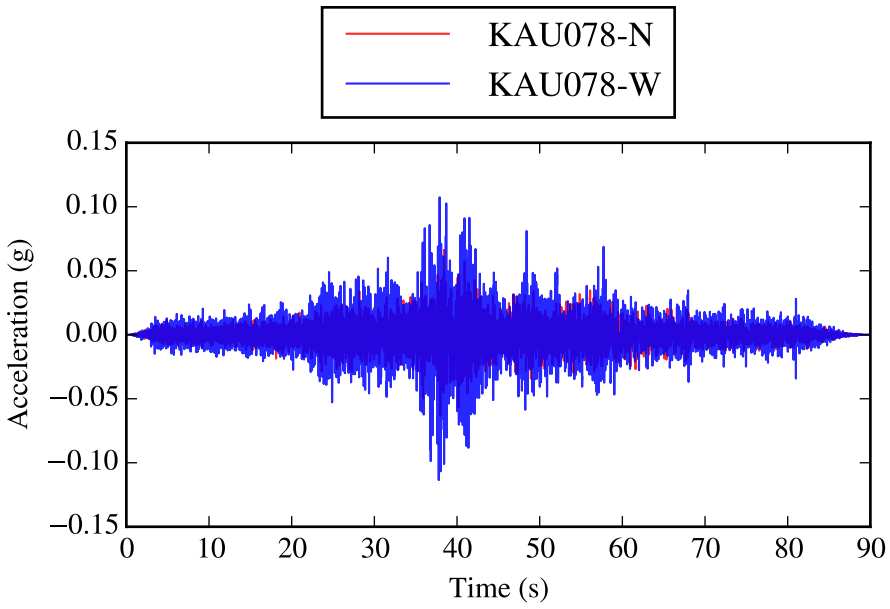


Figure F.272: CEUS 77: *KAU078-N* and *KAU078-W*.

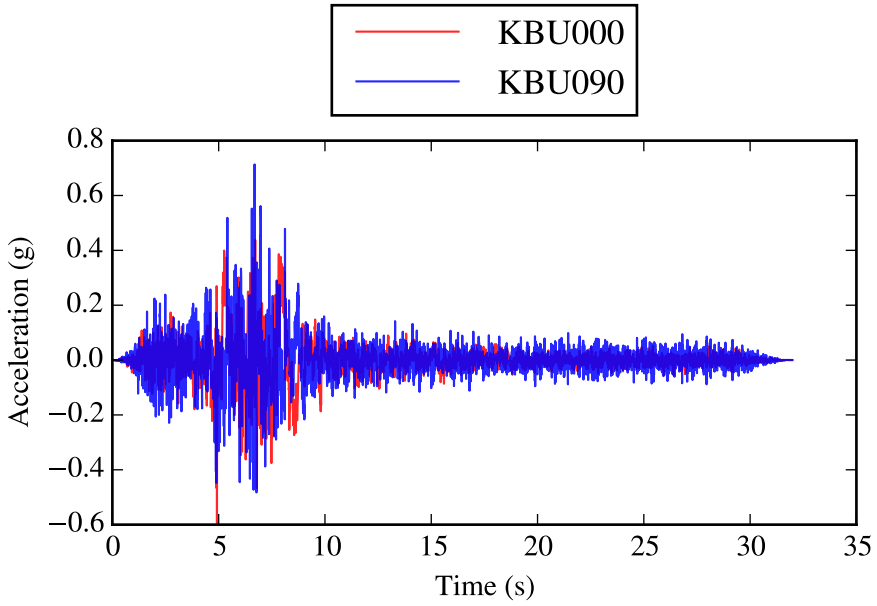


Figure F.273: CEUS 78: *KBU000* and *KBU090*.

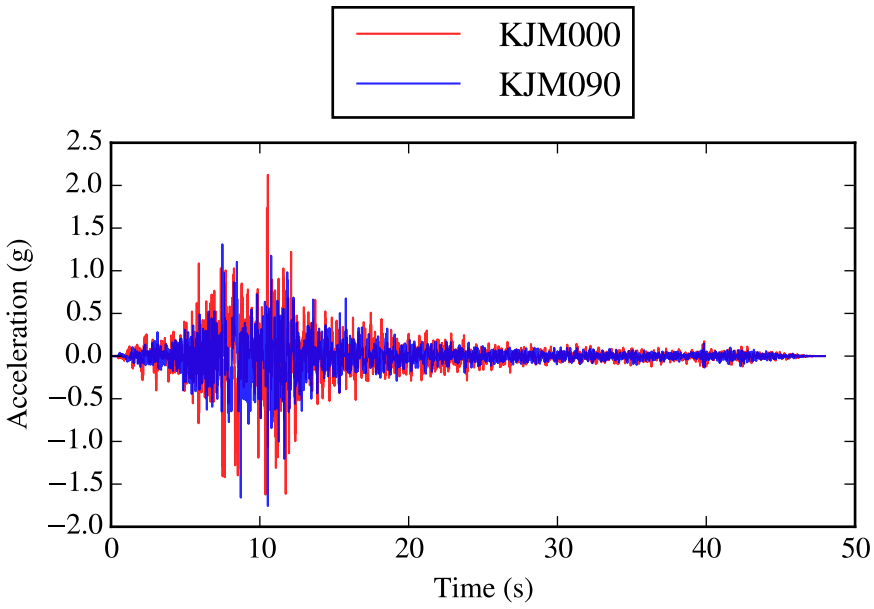


Figure F.274: CEUS 79: *KJM000* and *KJM090*.

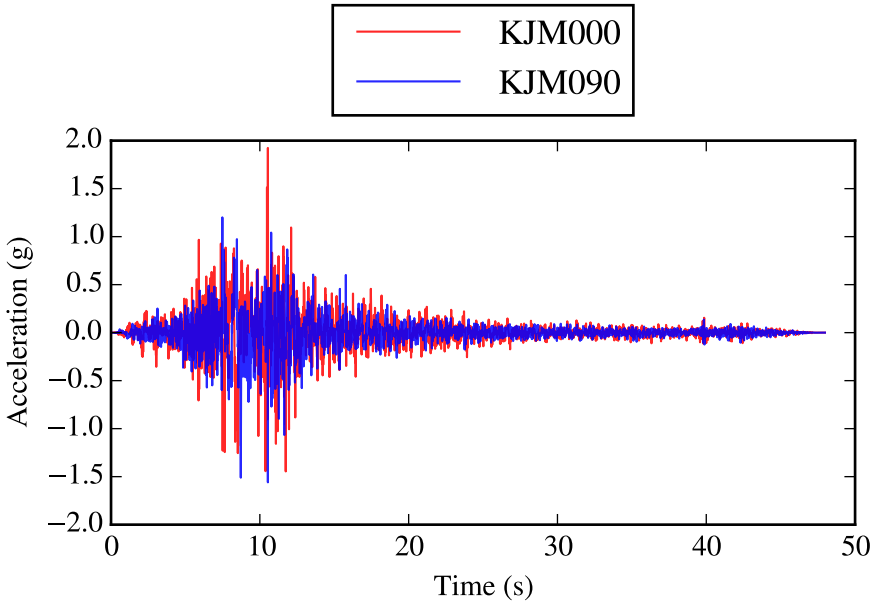


Figure F.275: CEUS 80: *KJM000* and *KJM090*.

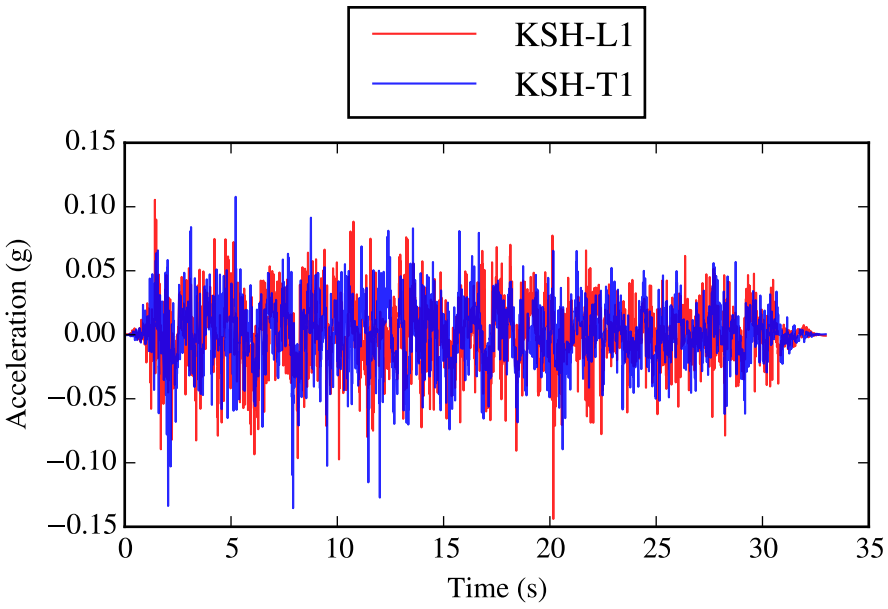


Figure F.276: CEUS 81: *KSH-L1* and *KSH-T1*.

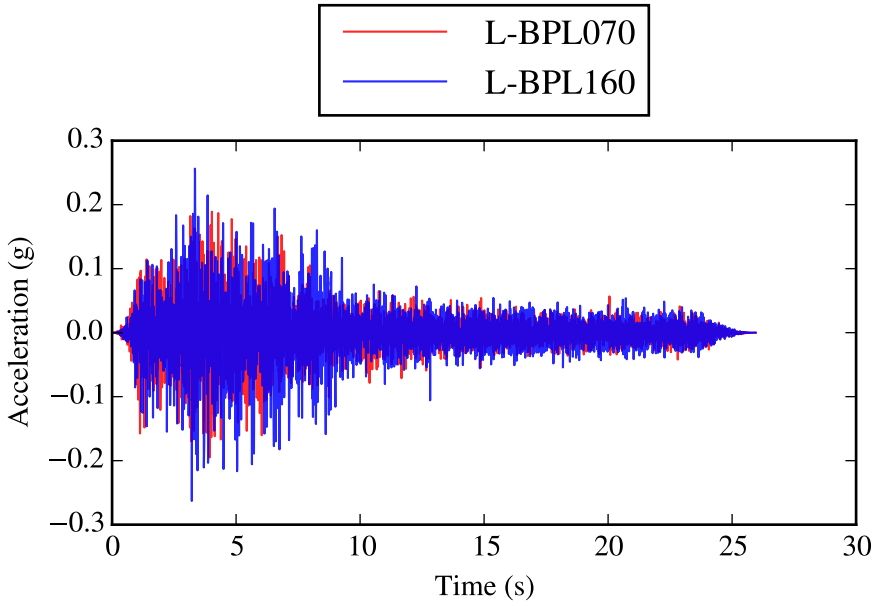


Figure F.277: CEUS 82: *L-BPL070* and *L-BPL160*.

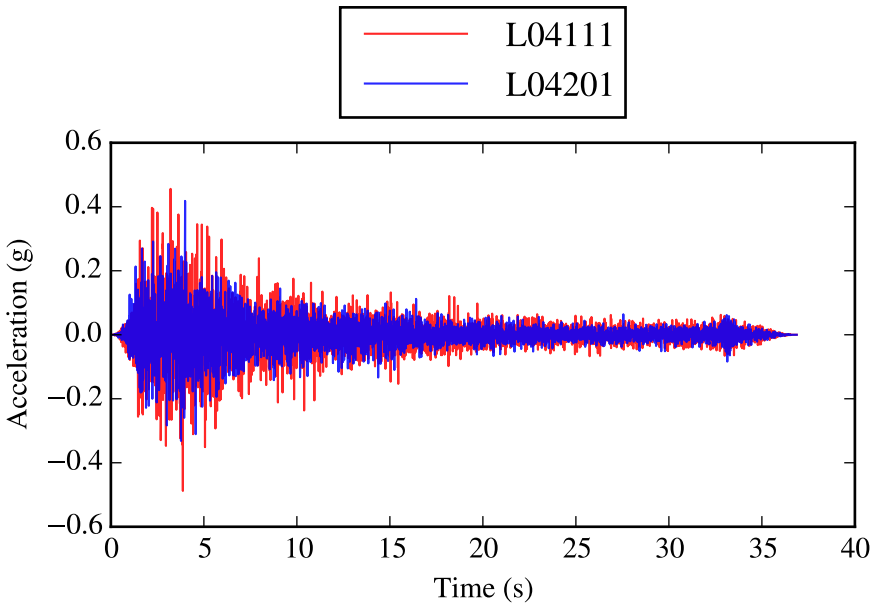


Figure F.278: CEUS 83: *L04111* and *L04201*.

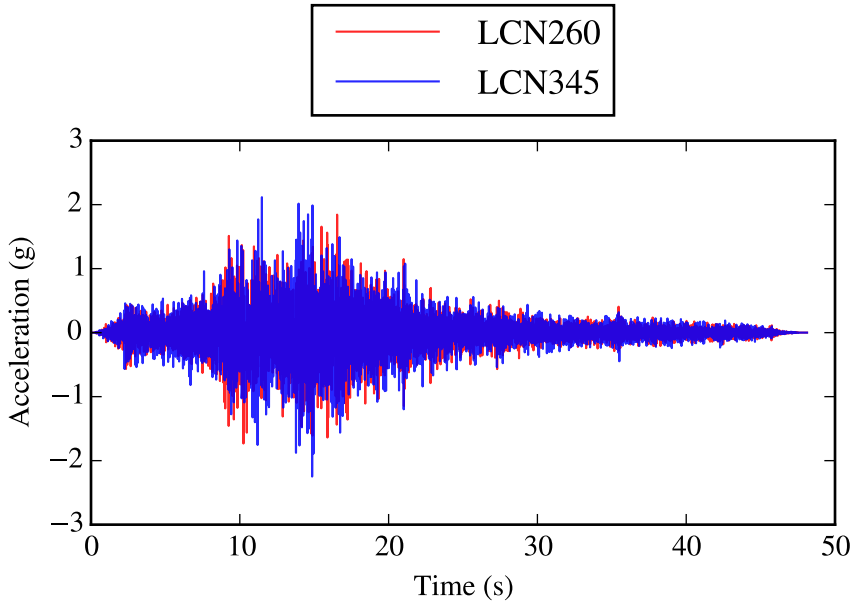


Figure F.279: CEUS 84: *LCN260* and *LCN345*.

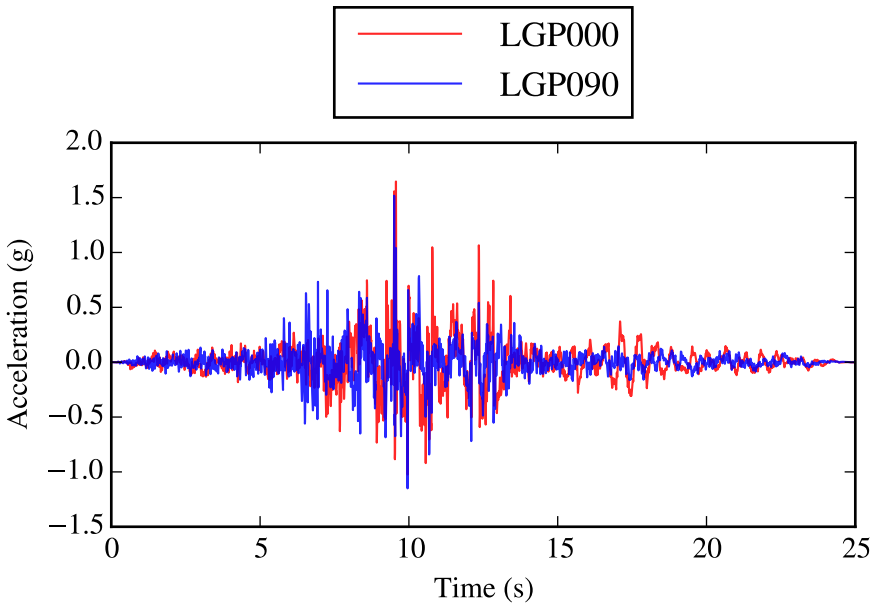


Figure F.280: CEUS 85: *LGP000* and *LGP090*.

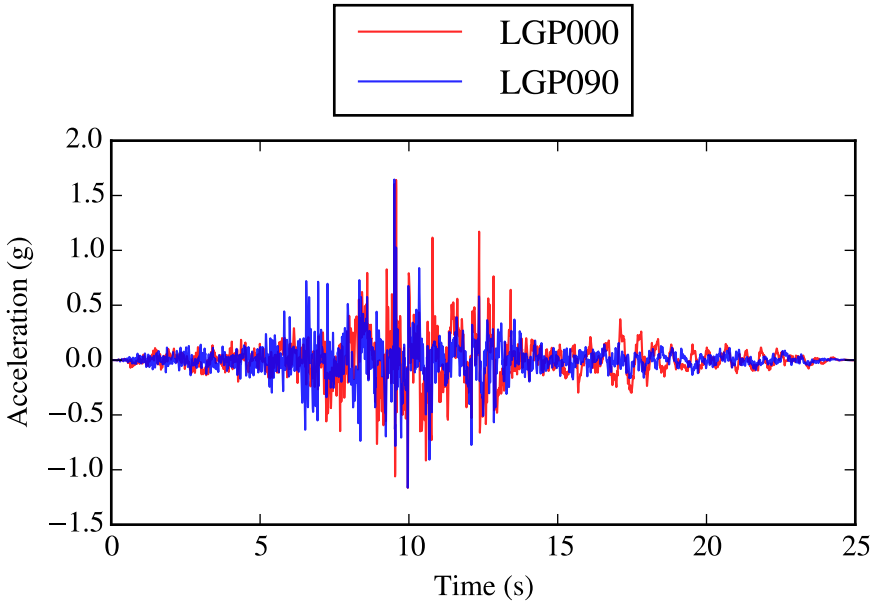


Figure F.281: CEUS 86: *LGP000* and *LGP090*.

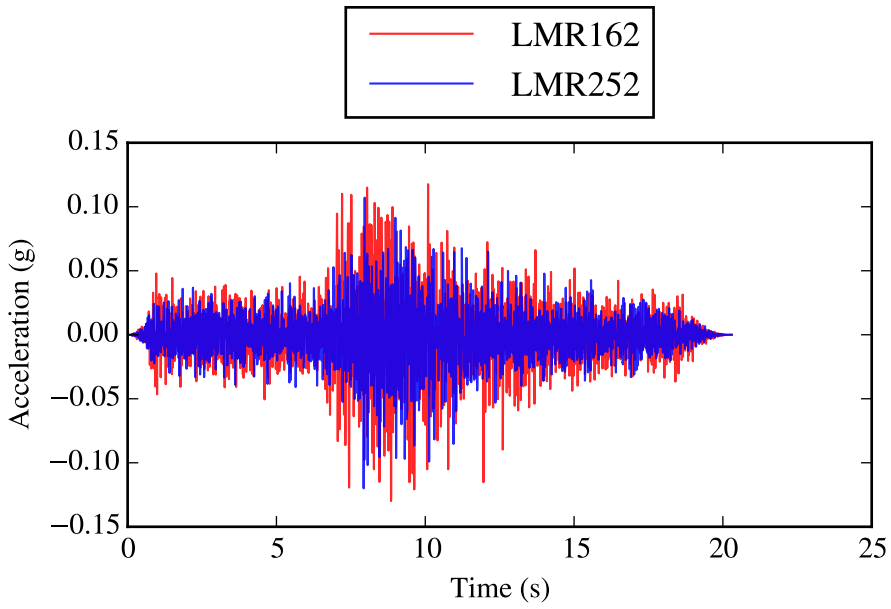


Figure F.282: CEUS 87: *LMR162* and *LMR252*.

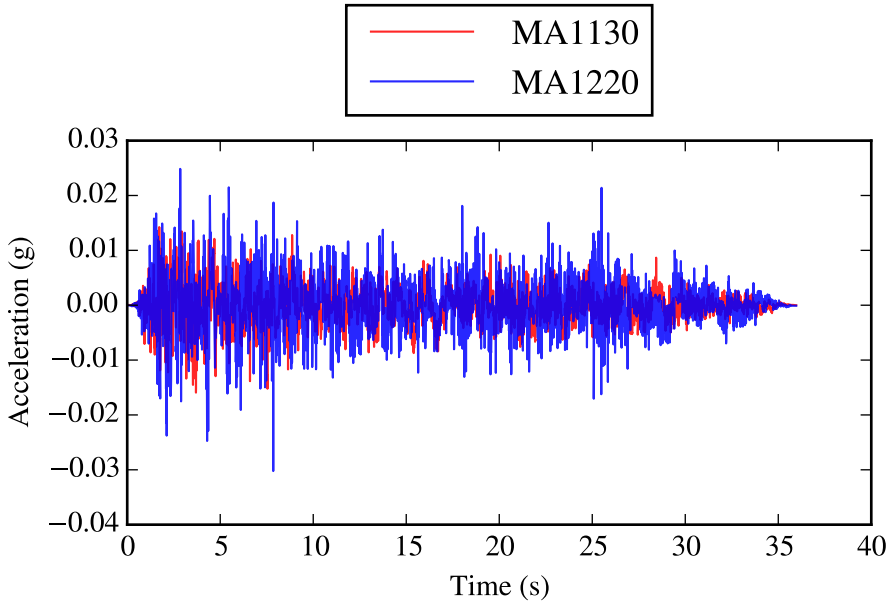


Figure F.283: CEUS 88: *MA1130* and *MA1220*.

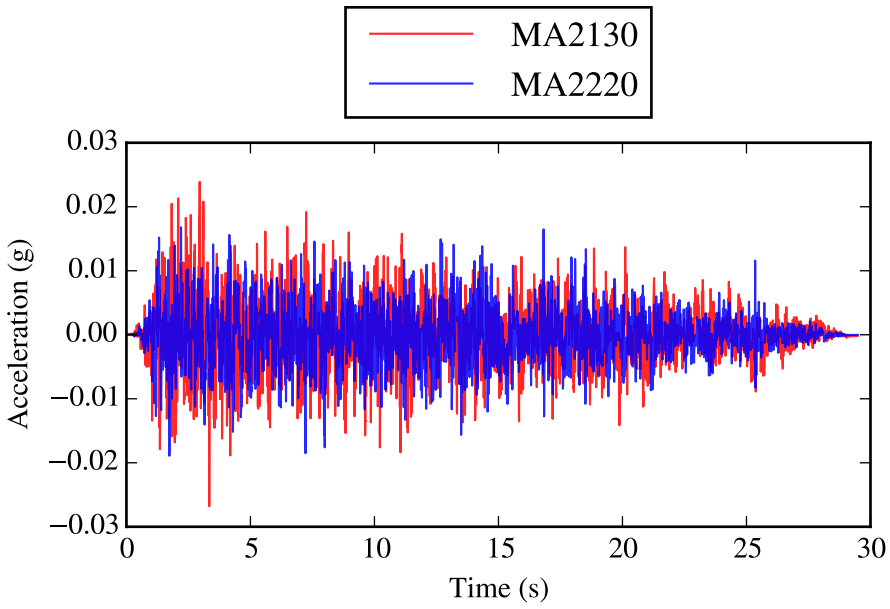


Figure F.284: CEUS 89: *MA2130* and *MA2220*.

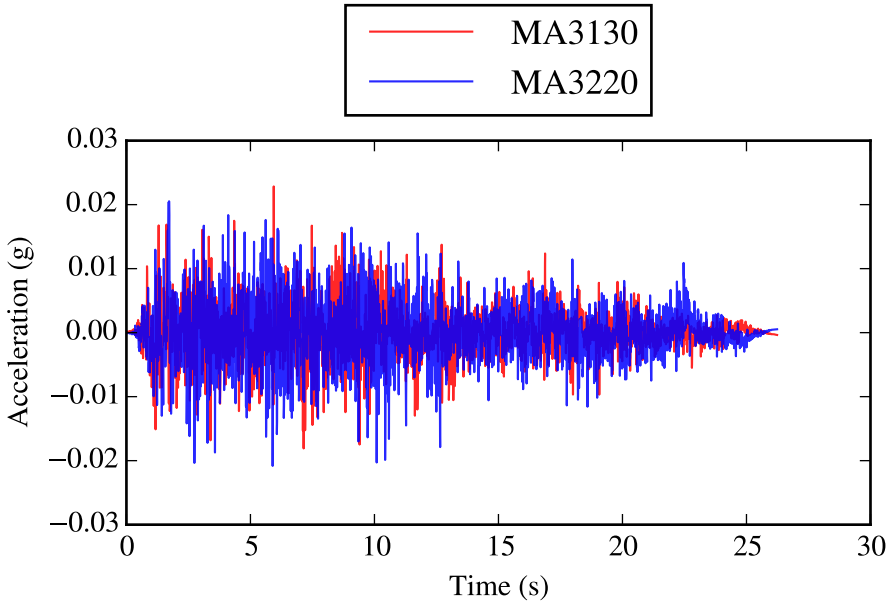


Figure F.285: CEUS 90: *MA3130* and *MA3220*.

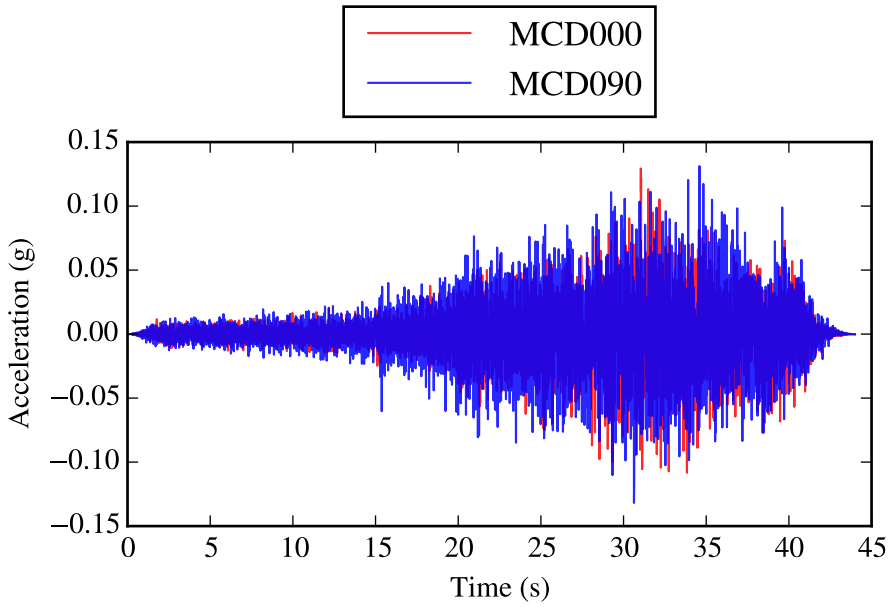


Figure F.286: CEUS 91: *MCD000* and *MCD090*.

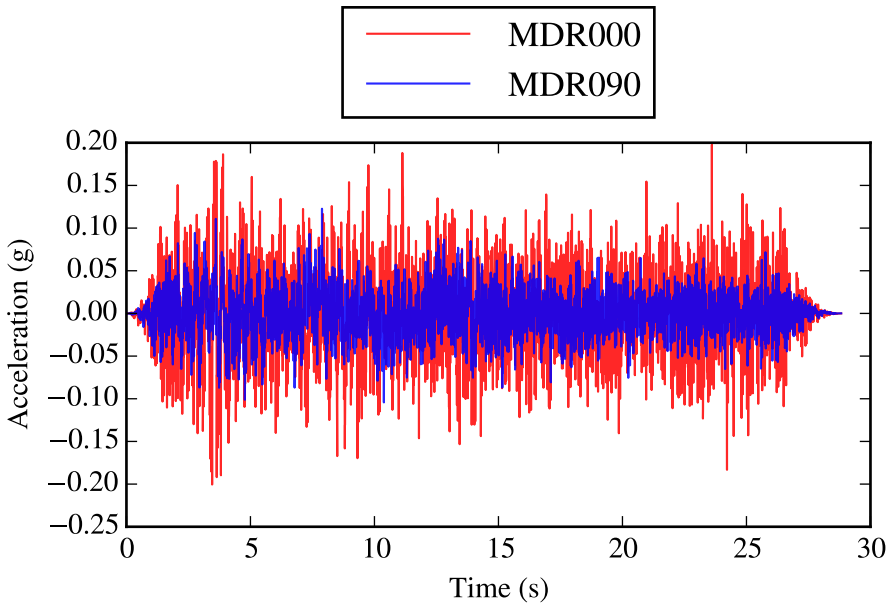


Figure F.287: CEUS 92: *MDR000* and *MDR090*.

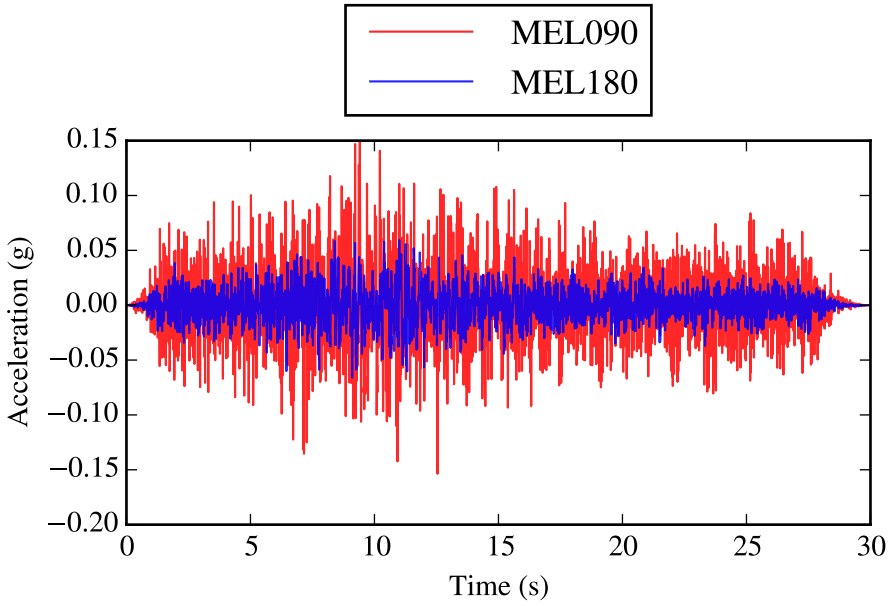


Figure F.288: CEUS 93: *MEL090* and *MEL180*.

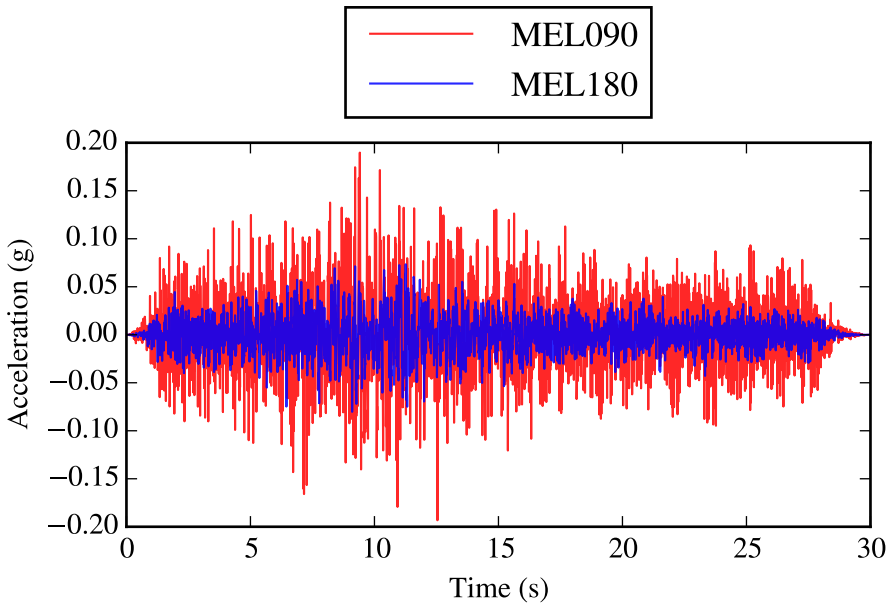


Figure F.289: CEUS 94: *MEL090* and *MEL180*.

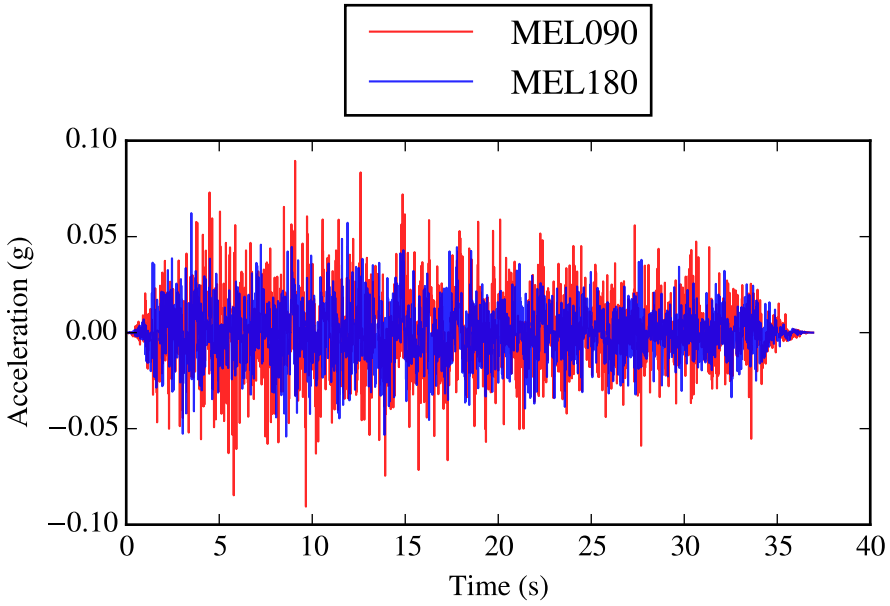


Figure F.290: CEUS 95: *MEL090* and *MEL180*.

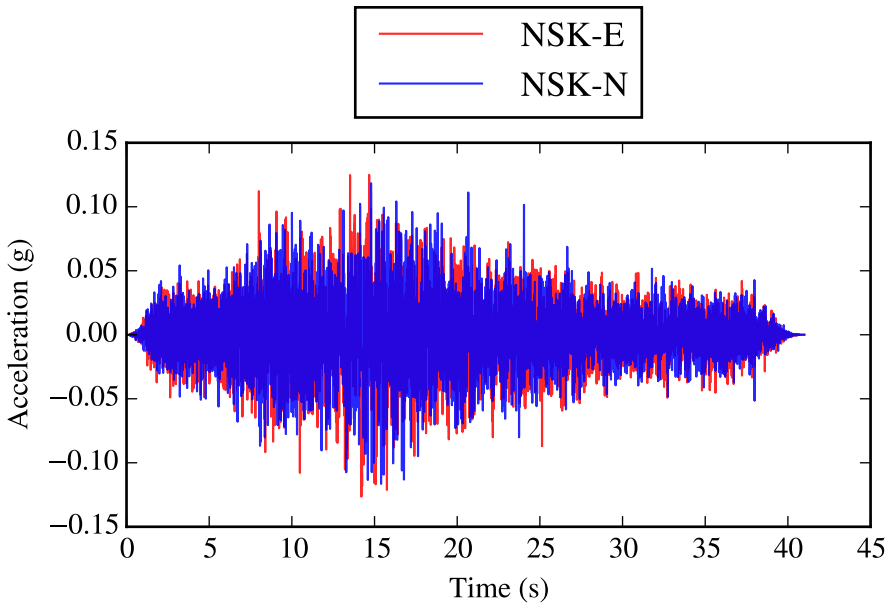


Figure F.291: CEUS 96: *NSK-E* and *NSK-N*.

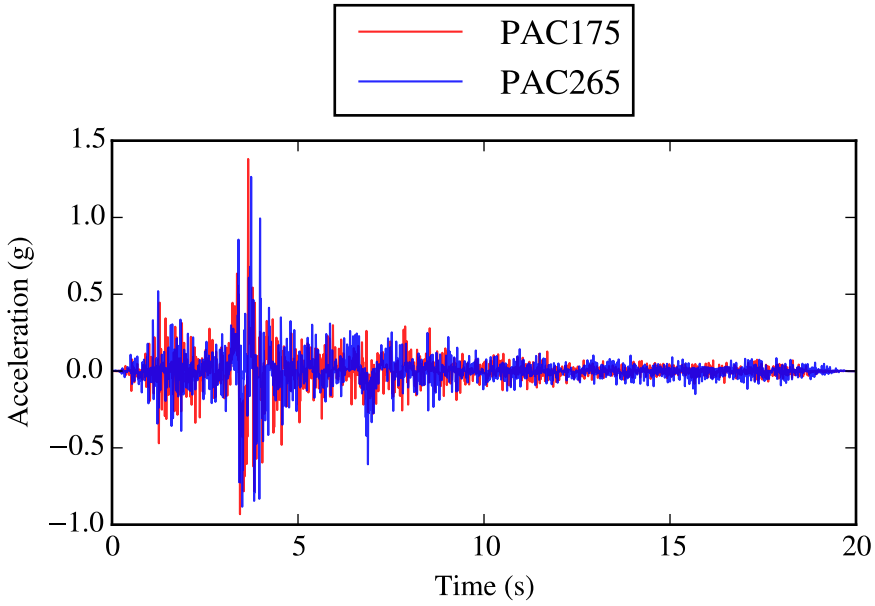


Figure F.292: CEUS 97: *PAC175* and *PAC265*.

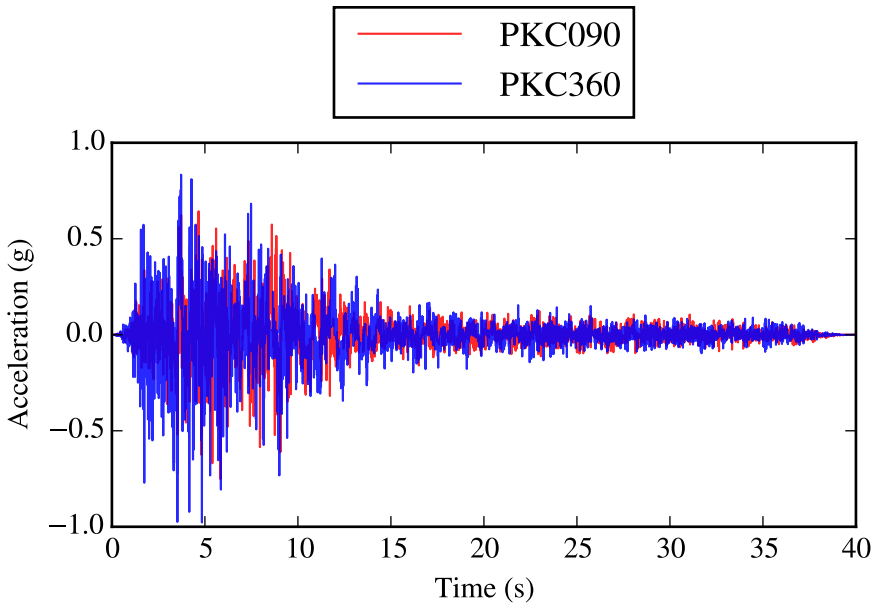


Figure F.293: CEUS 98: *PKC090* and *PKC360*.

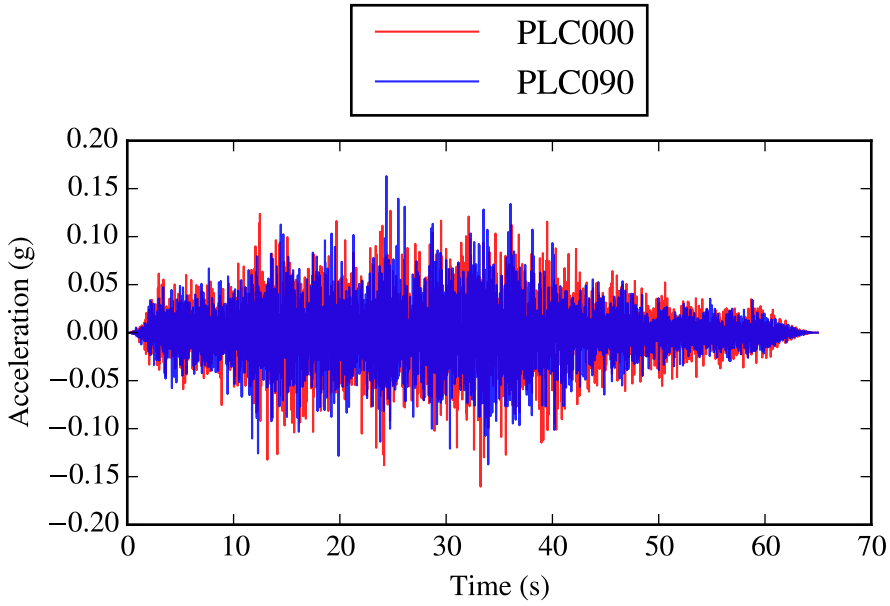


Figure F.294: CEUS 99: *PLC000* and *PLC090*.

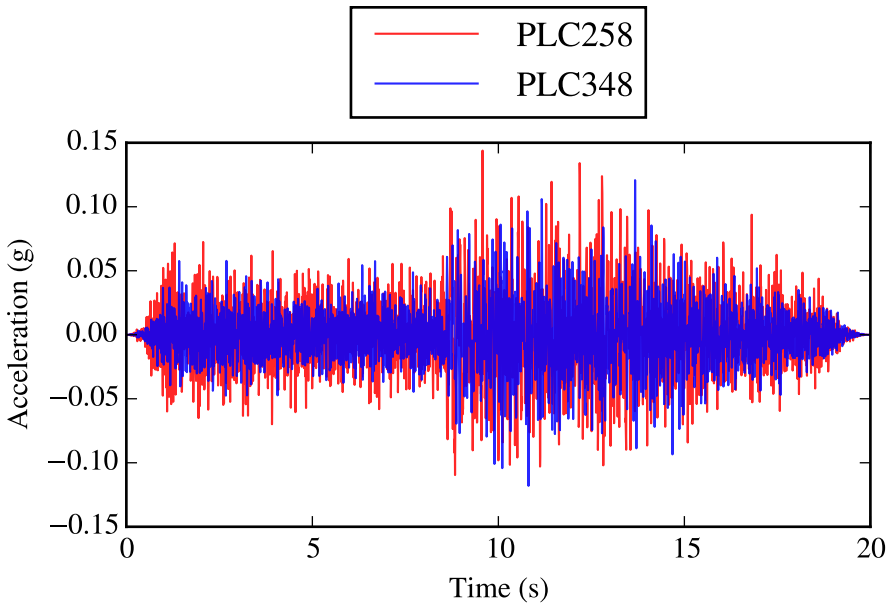


Figure F.295: CEUS 100: *PLC258* and *PLC348*.

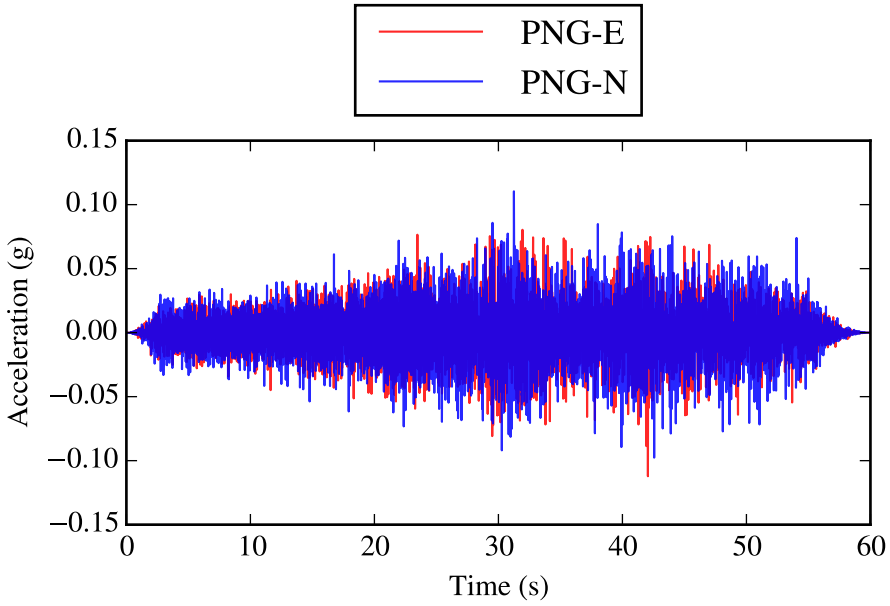


Figure F.296: CEUS 101: *PNG-E* and *PNG-N*.

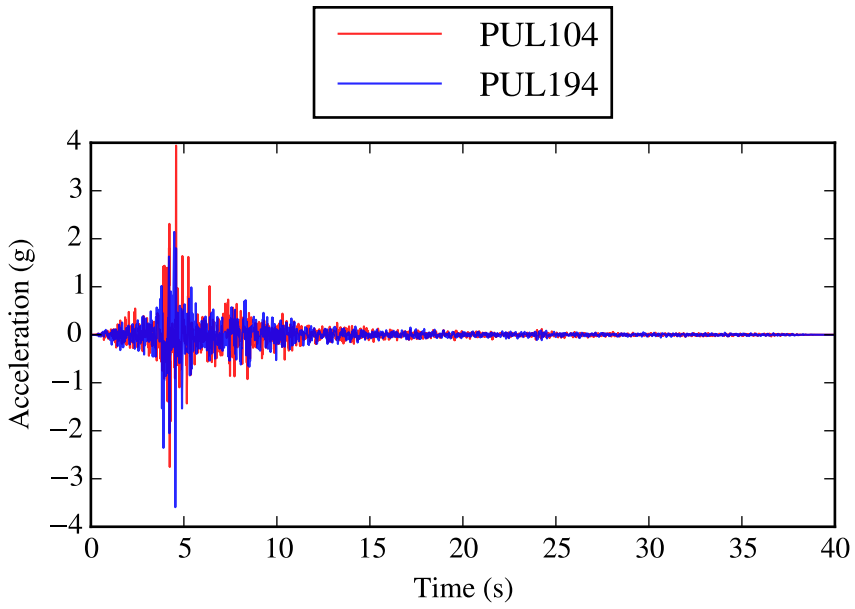


Figure F.297: CEUS 102: *PUL104* and *PUL194*.

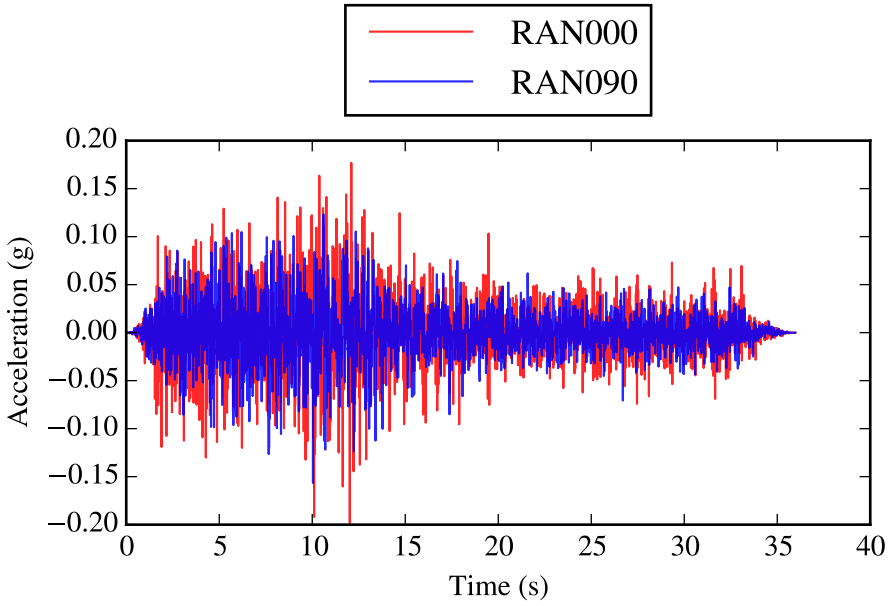


Figure F.298: CEUS 103: *RAN000* and *RAN090*.

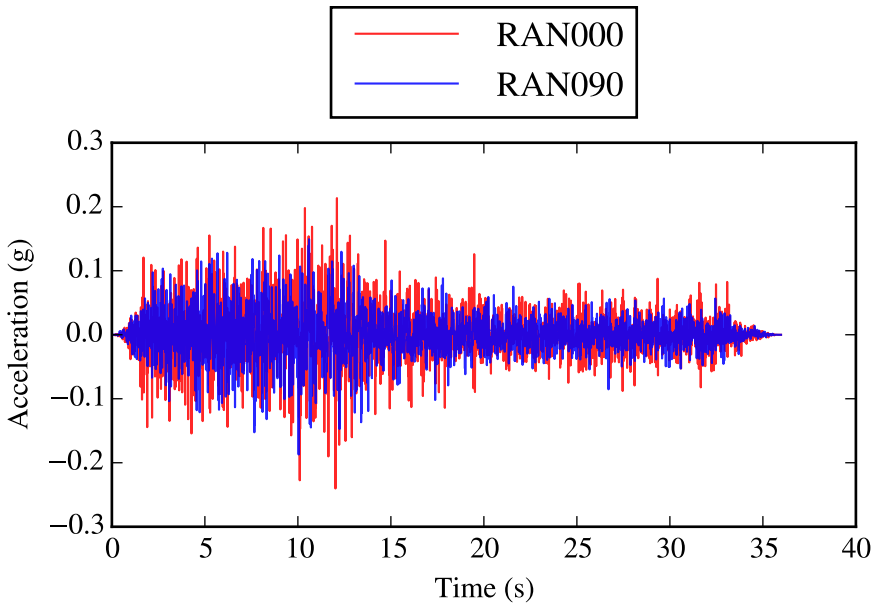


Figure F.299: CEUS 104: *RAN000* and *RAN090*.

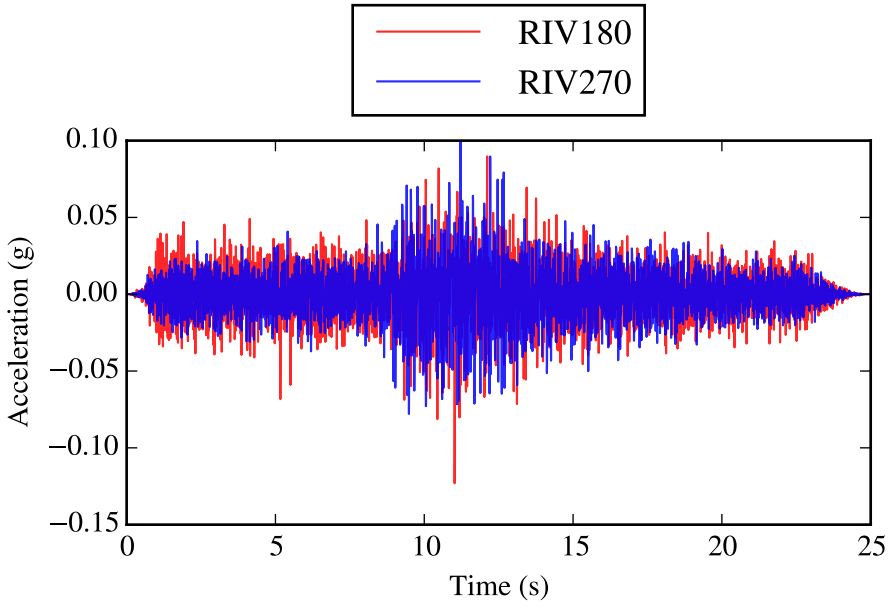


Figure F.300: CEUS 105: *RIV180* and *RIV270*.

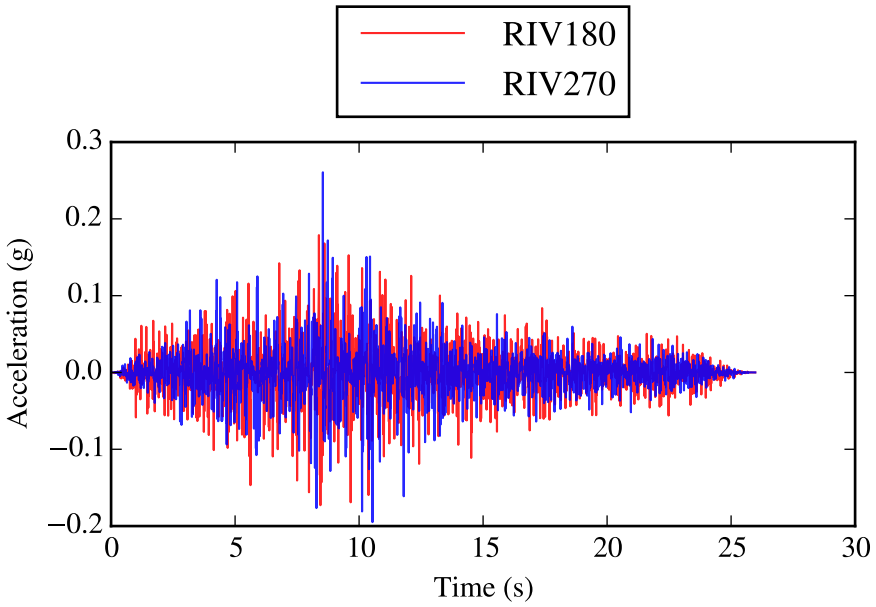


Figure F.301: CEUS 106: *RIV180* and *RIV270*.

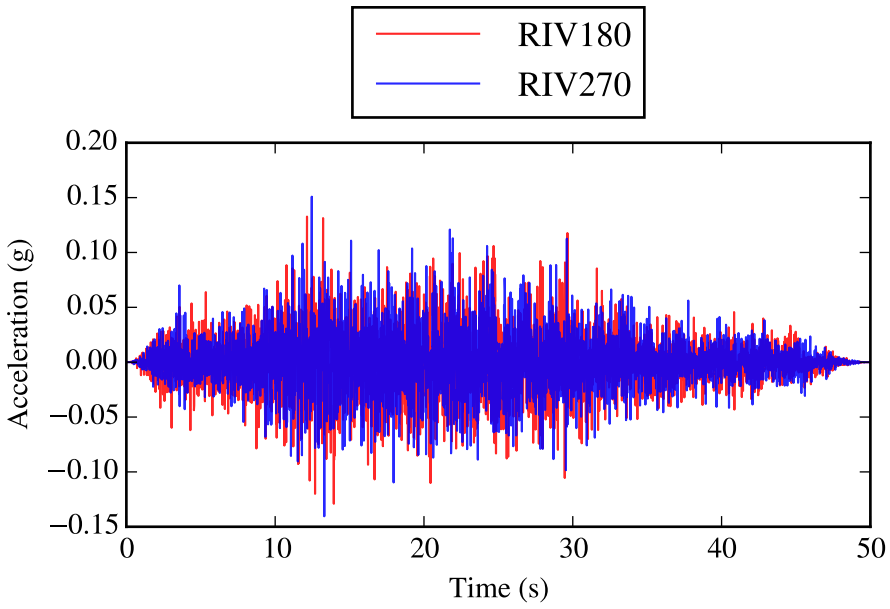


Figure F.302: CEUS 107: *RIV180* and *RIV270*.

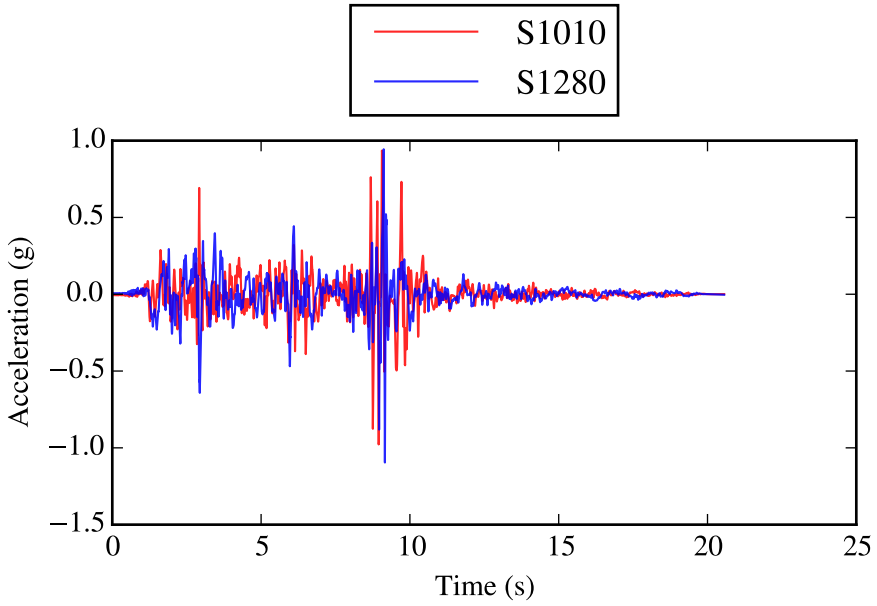


Figure F.303: CEUS 108: *S1010* and *S1280*.

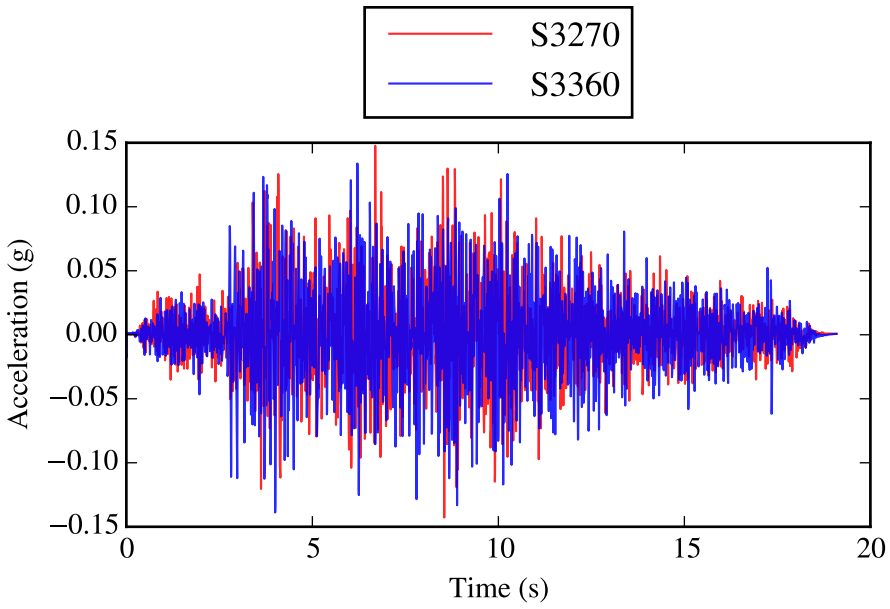


Figure F.304: CEUS 109: *S3270* and *S3360*.

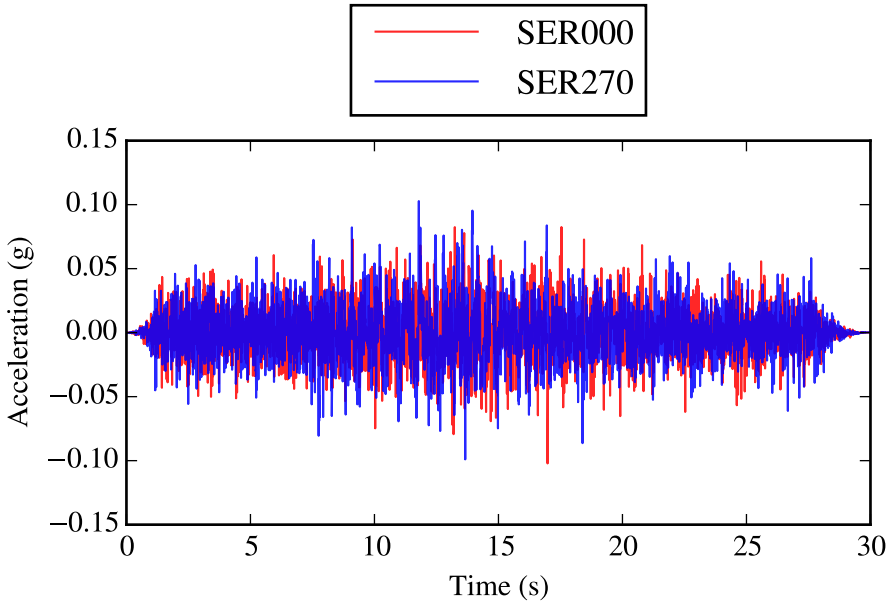


Figure F.305: CEUS 110: *SER000* and *SER270*.

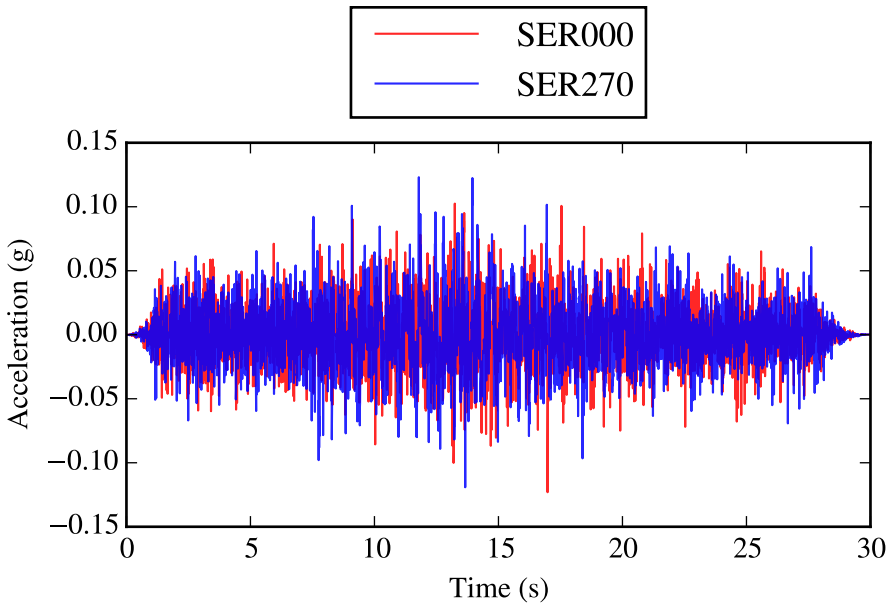


Figure F.306: CEUS 111: *SER000* and *SER270*.

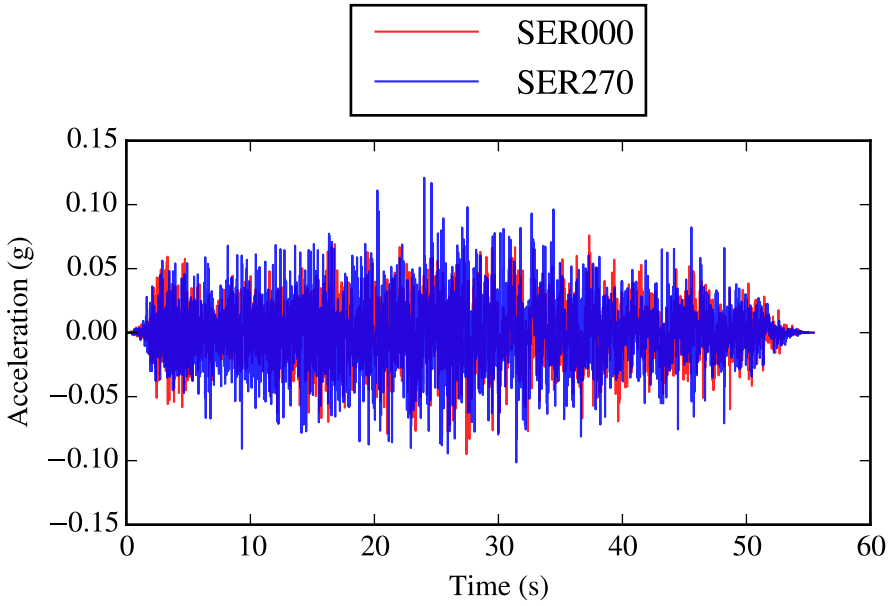


Figure F.307: CEUS 112: *SER000* and *SER270*.

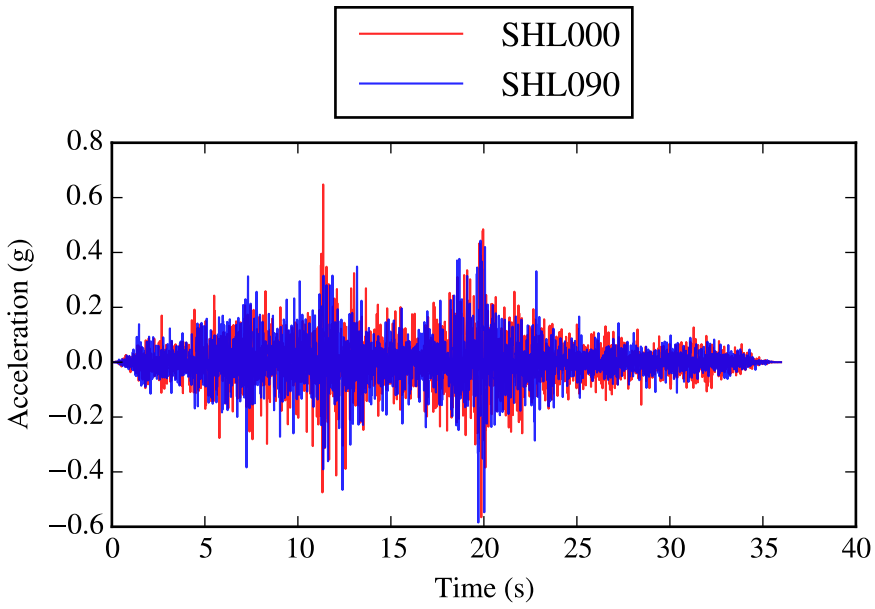


Figure F.308: CEUS 113: *SHL000* and *SHL090*.

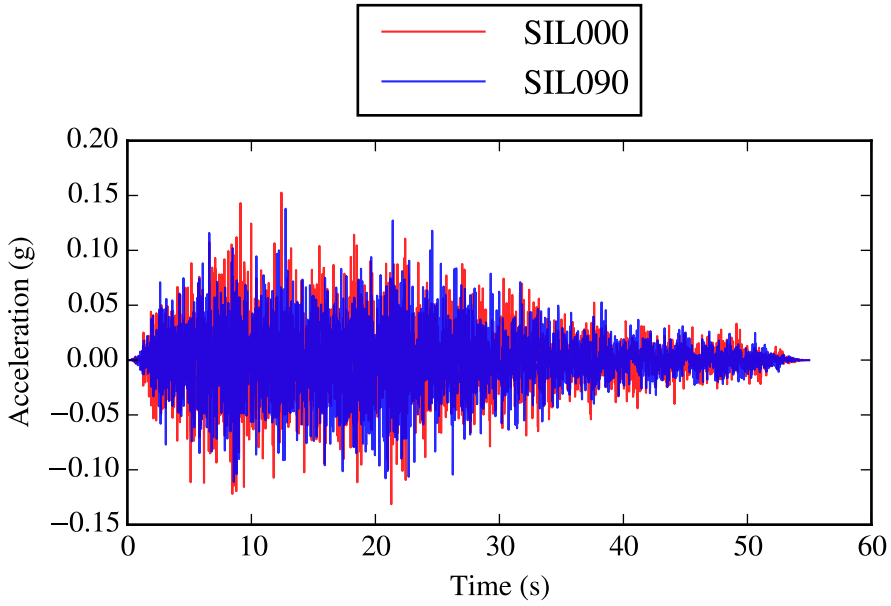


Figure F.309: CEUS 114: *SIL000* and *SIL090*.

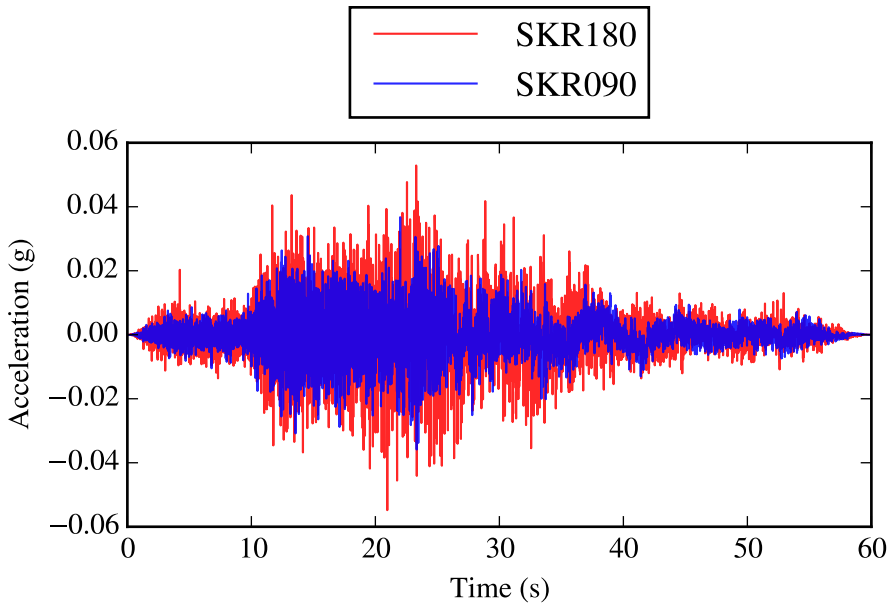


Figure F.310: CEUS 115: *SKR180* and *SKR090*.

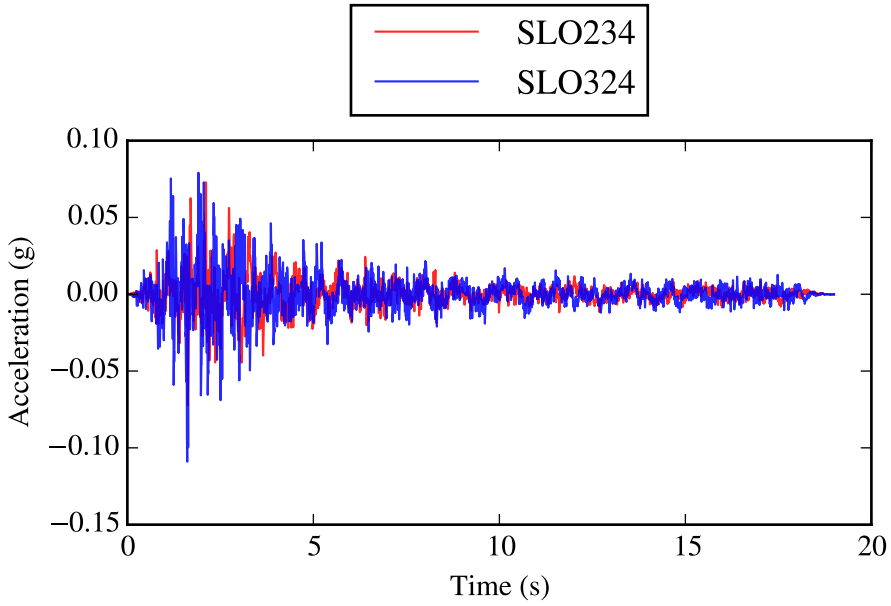


Figure F.311: CEUS 116: *SLO234* and *SLO324*.

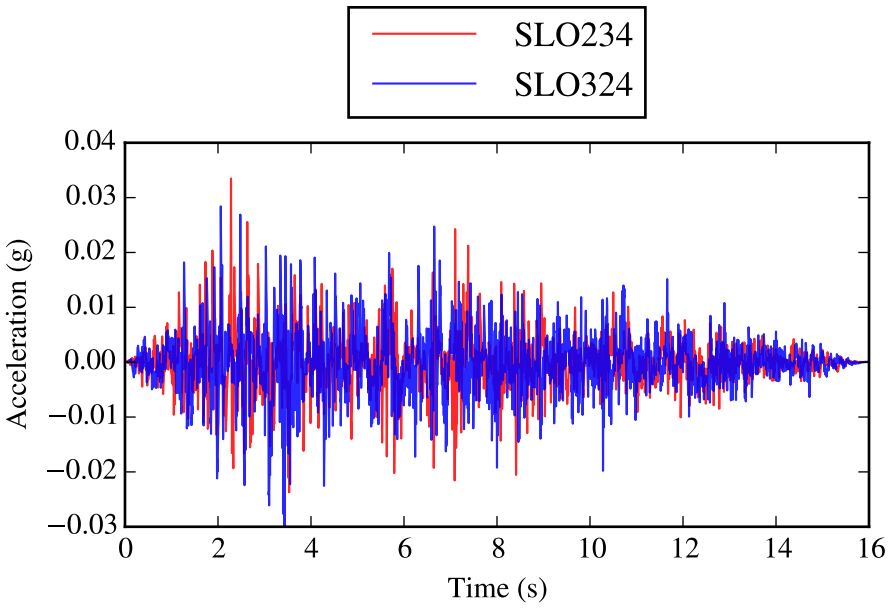


Figure F.312: CEUS 117: *SLO234* and *SLO324*.

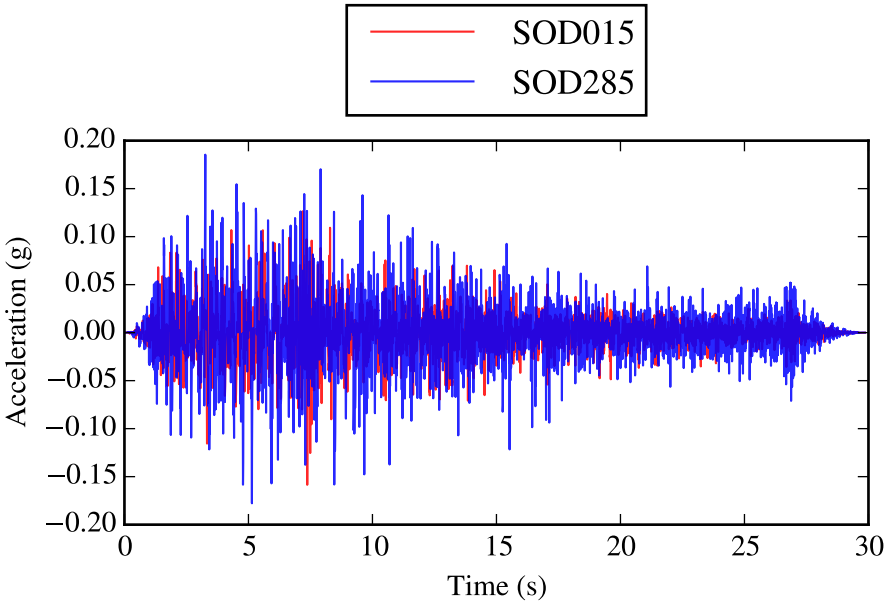


Figure F.313: CEUS 118: *SOD015* and *SOD285*.

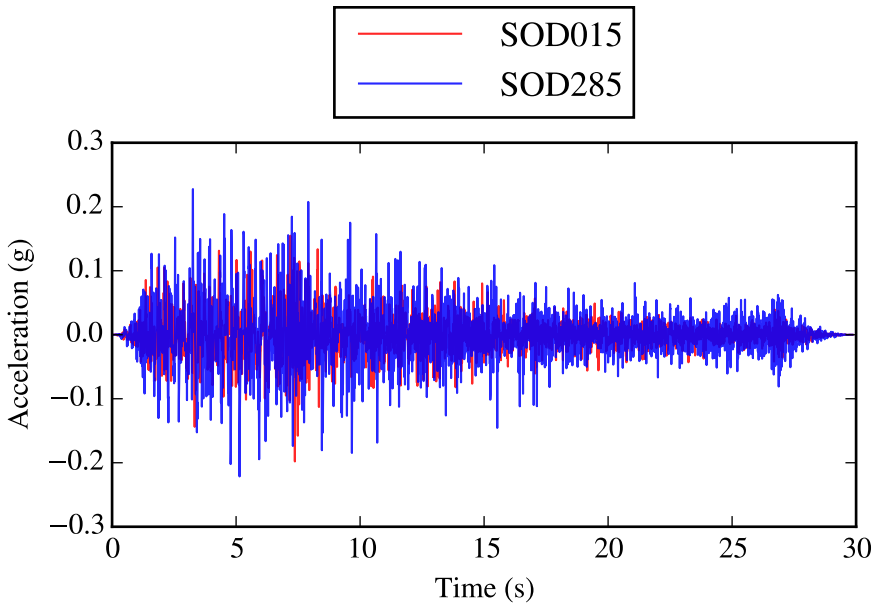


Figure F.314: CEUS 119: *SOD015* and *SOD285*.

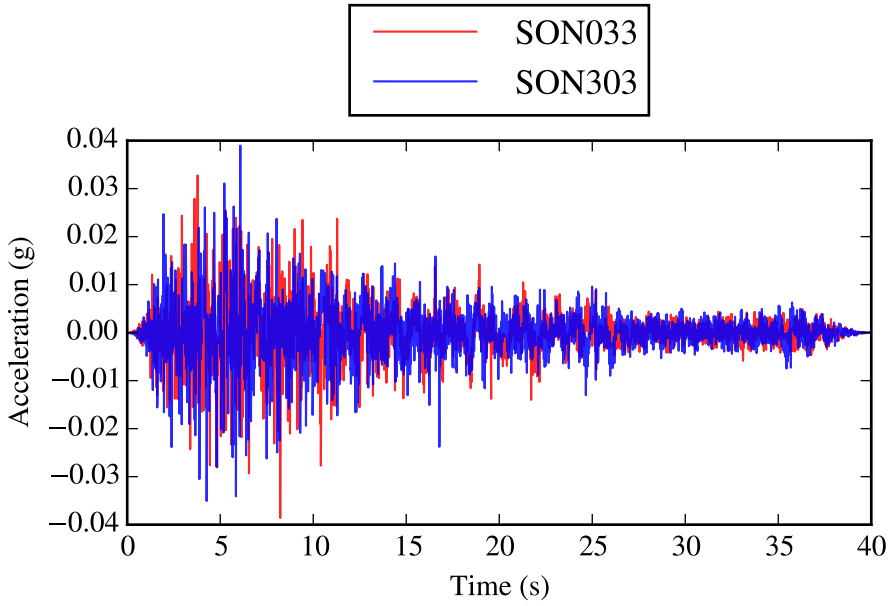


Figure F.315: CEUS 120: *SON033* and *SON303*.

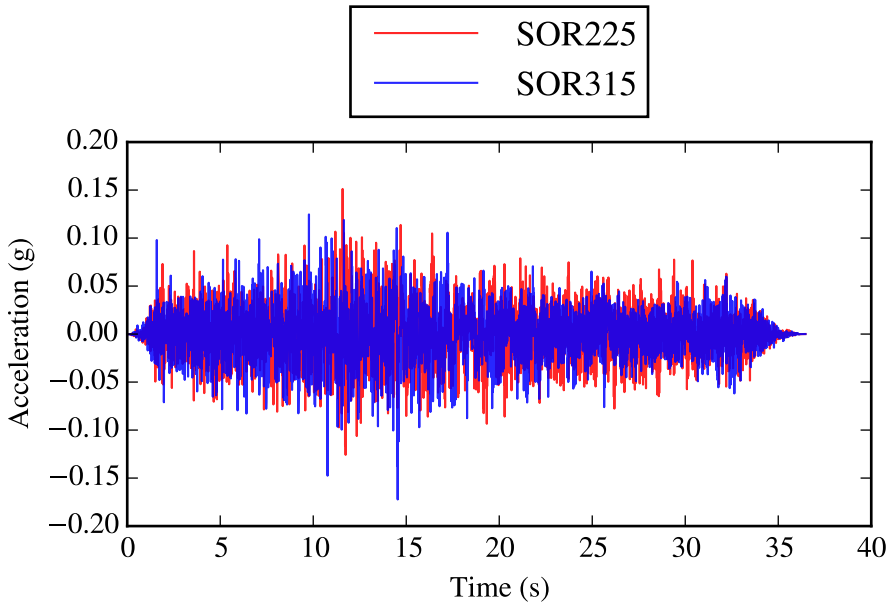


Figure F.316: CEUS 121: *SOR225* and *SOR315*.

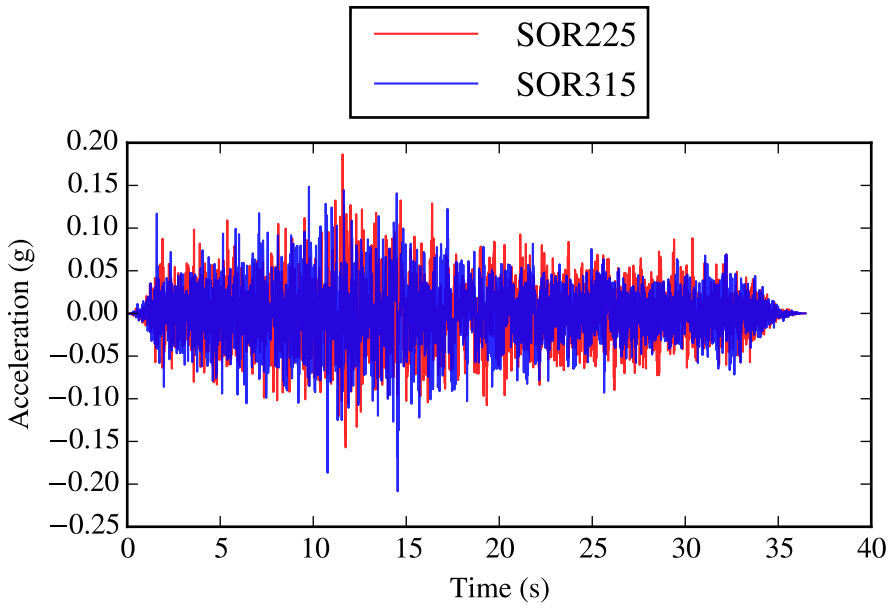


Figure F.317: CEUS 122: *SOR225* and *SOR315*.

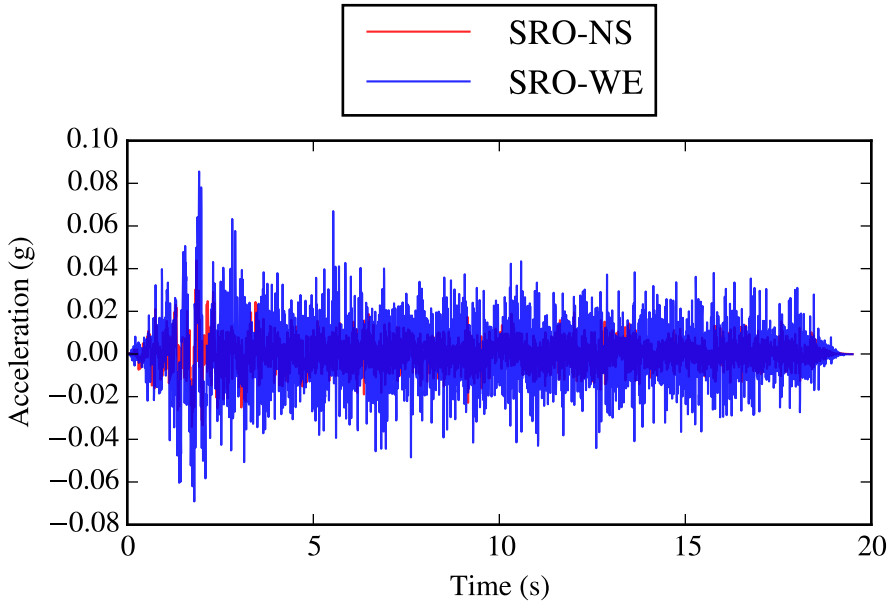


Figure F.318: CEUS 123: *SRO-NS* and *SRO-WE*.

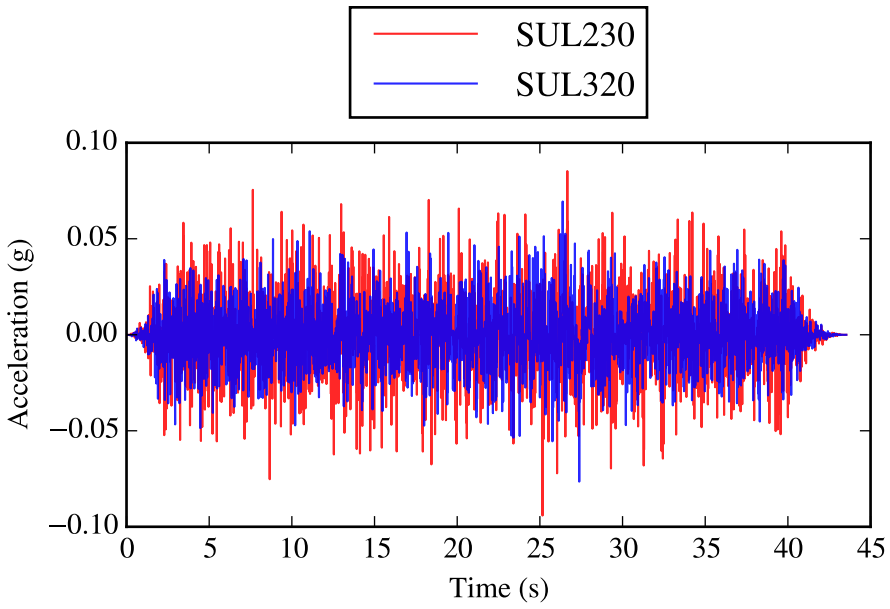


Figure F.319: CEUS 124: *SUL230* and *SUL320*.

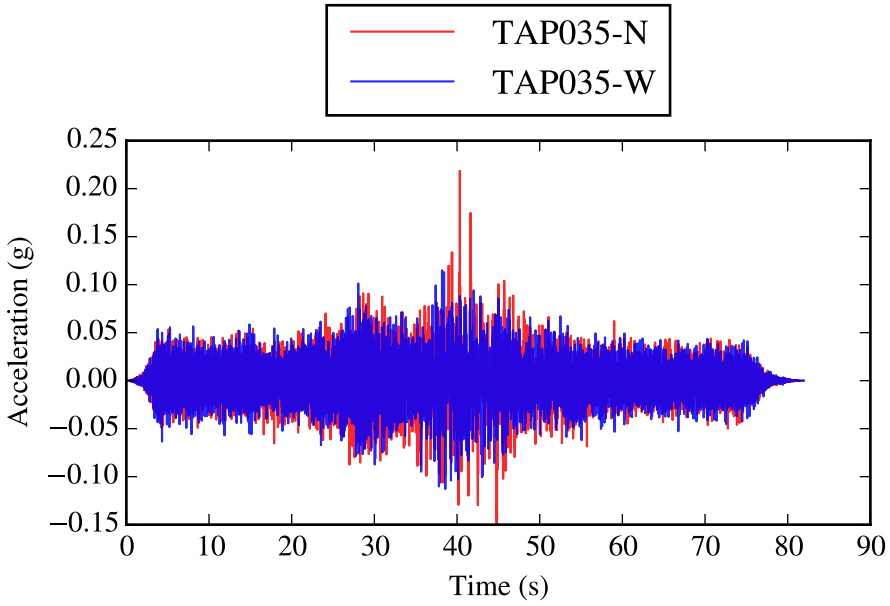


Figure F.320: CEUS 125: *TAP035-N* and *TAP035-W*.

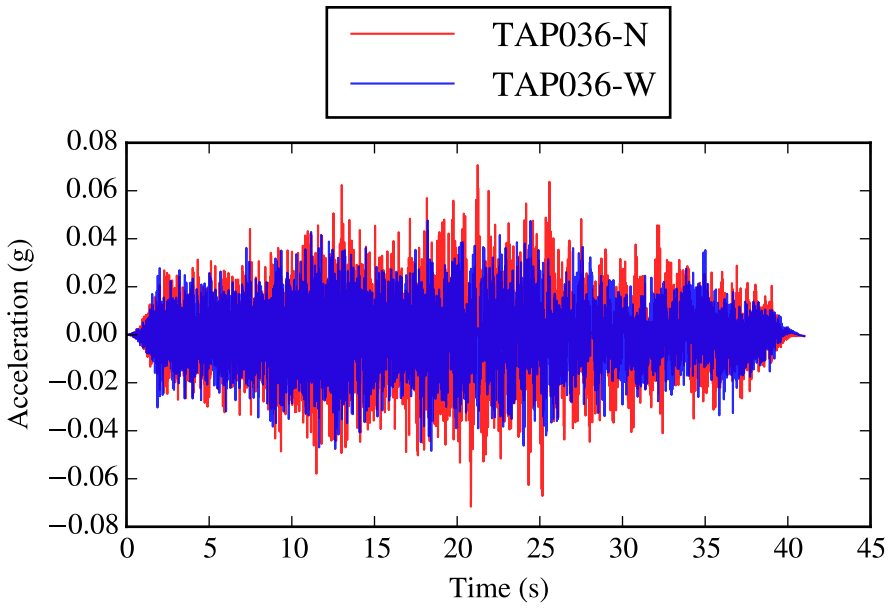


Figure F.321: CEUS 126: *TAP036-N* and *TAP036-W*.

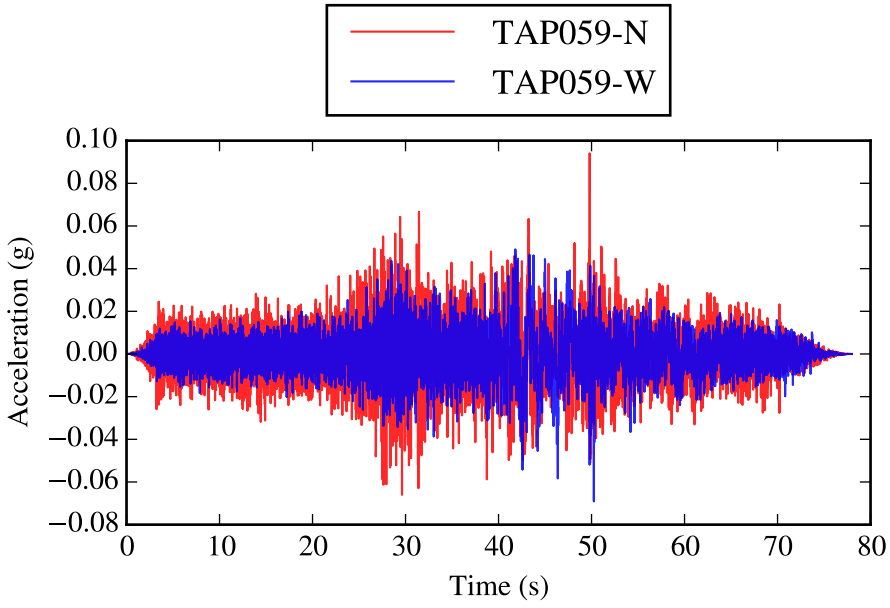


Figure F.322: CEUS 127: *TAP059-N* and *TAP059-W*.

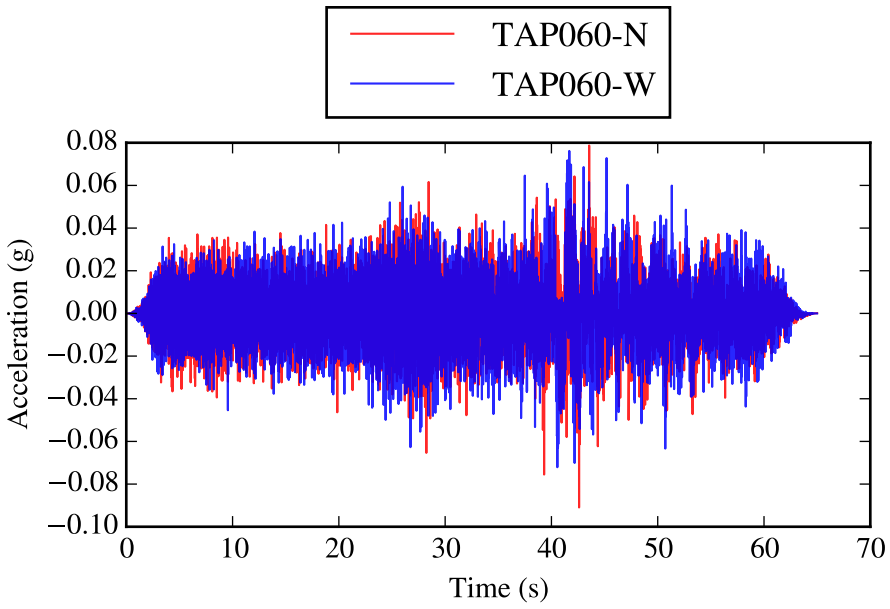


Figure F.323: CEUS 128: *TAP060-N* and *TAP060-W*.

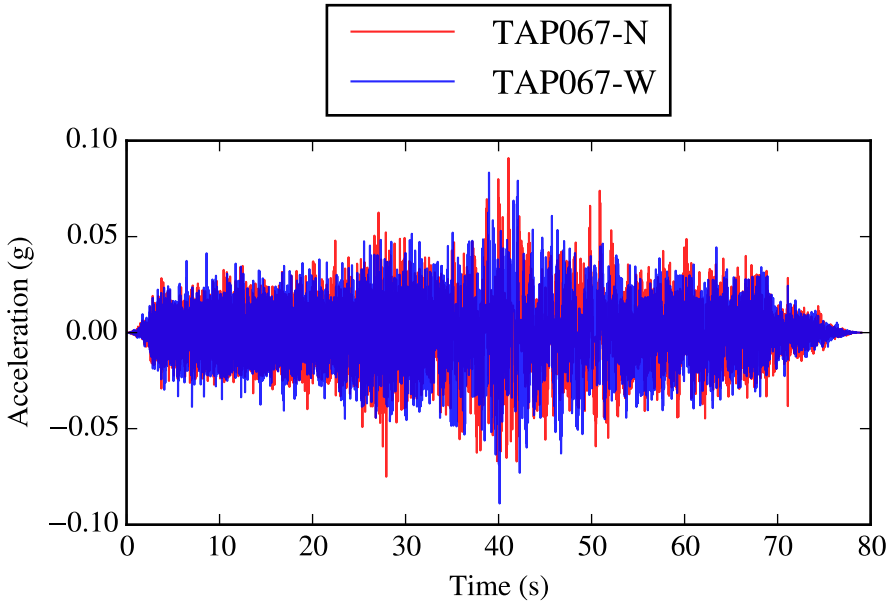


Figure F.324: CEUS 129: *TAP067-N* and *TAP067-W*.

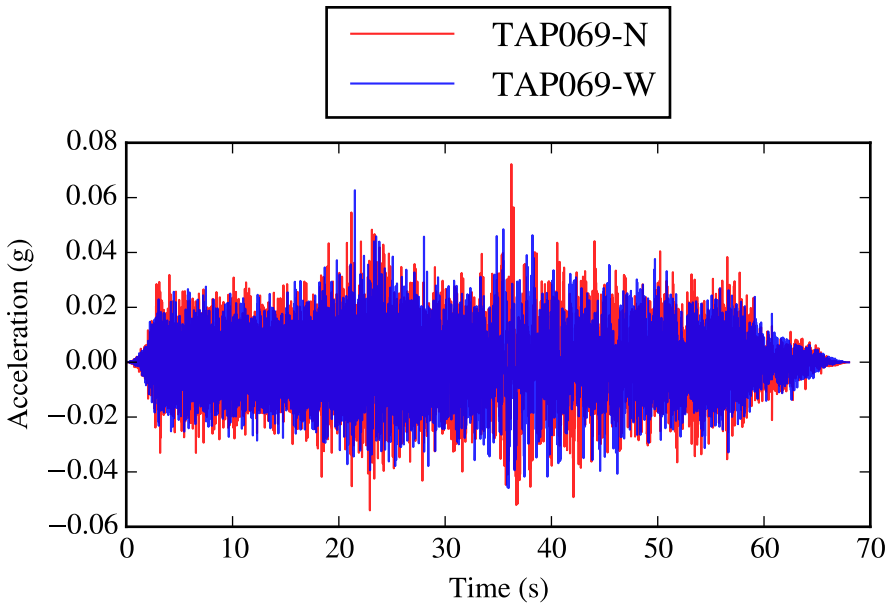


Figure F.325: CEUS 130: *TAP069-N* and *TAP069-W*.

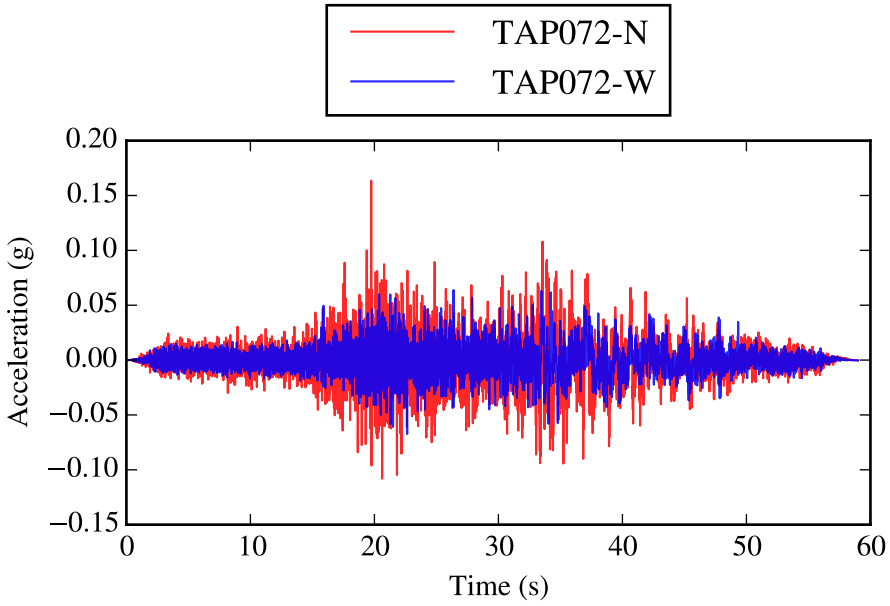


Figure F.326: CEUS 131: *TAP072-N* and *TAP072-W*.

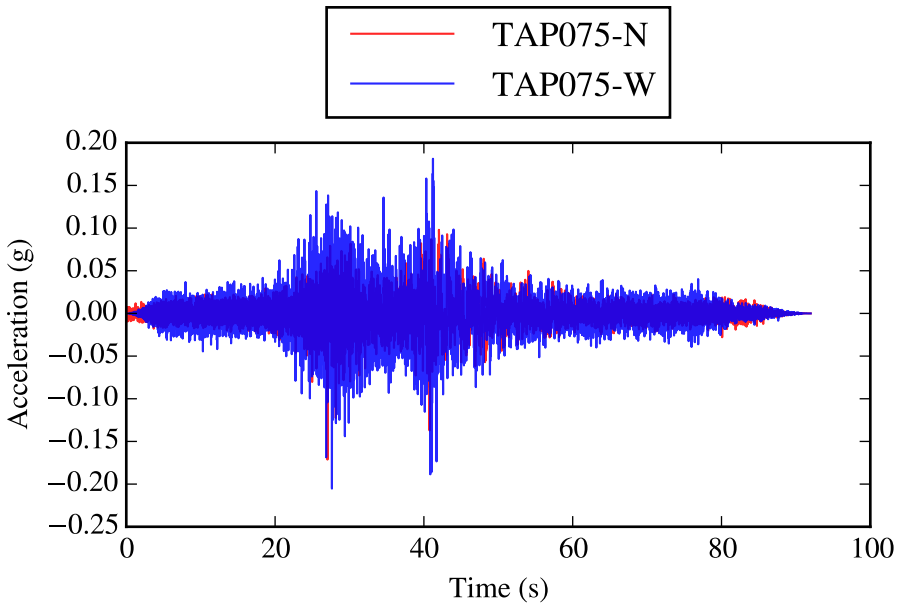


Figure F.327: CEUS 132: *TAP075-N* and *TAP075-W*.

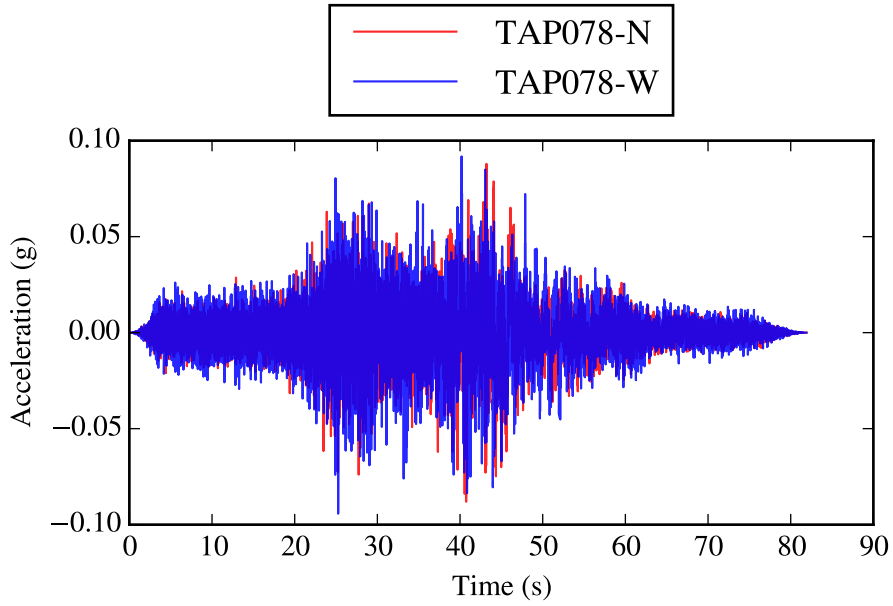


Figure F.328: CEUS 133: *TAP078-N* and *TAP078-W*.

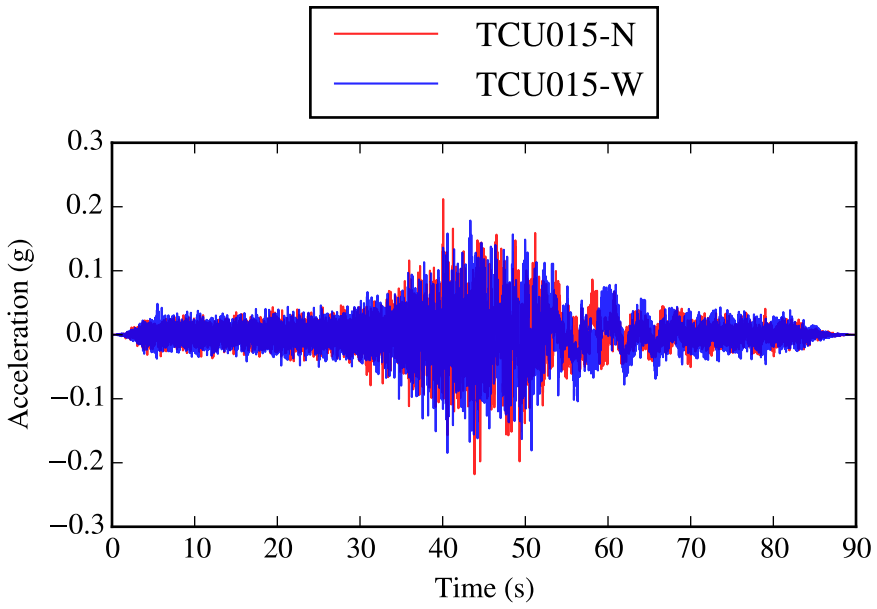


Figure F.329: CEUS 134: *TCU015-N* and *TCU015-W*.

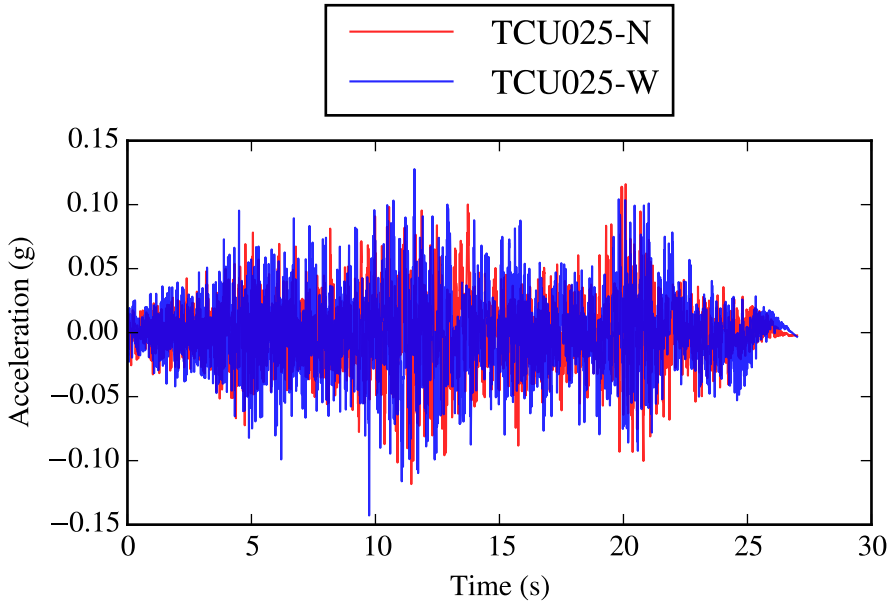


Figure F.330: CEUS 135: *TCU025-N* and *TCU025-W*.

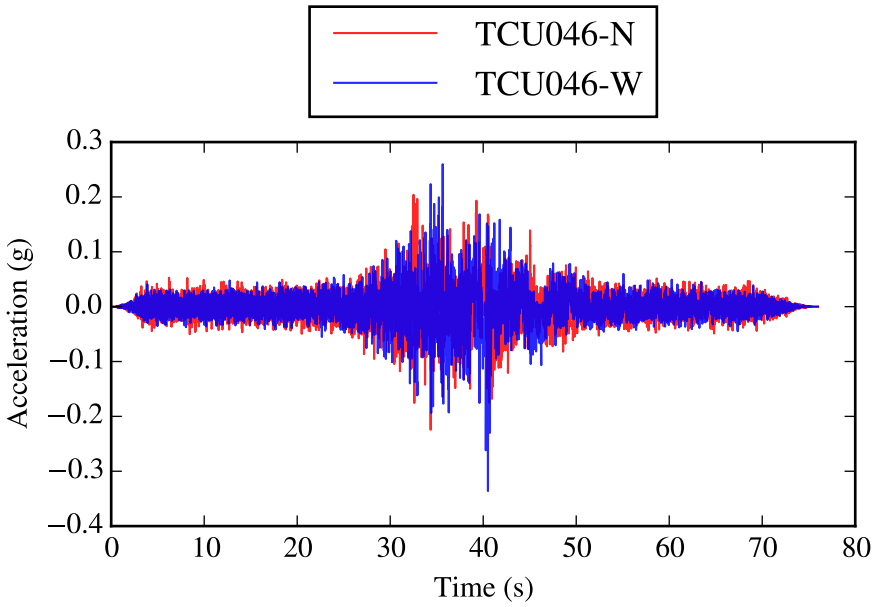


Figure F.331: CEUS 136: *TCU046-N* and *TCU046-W*.

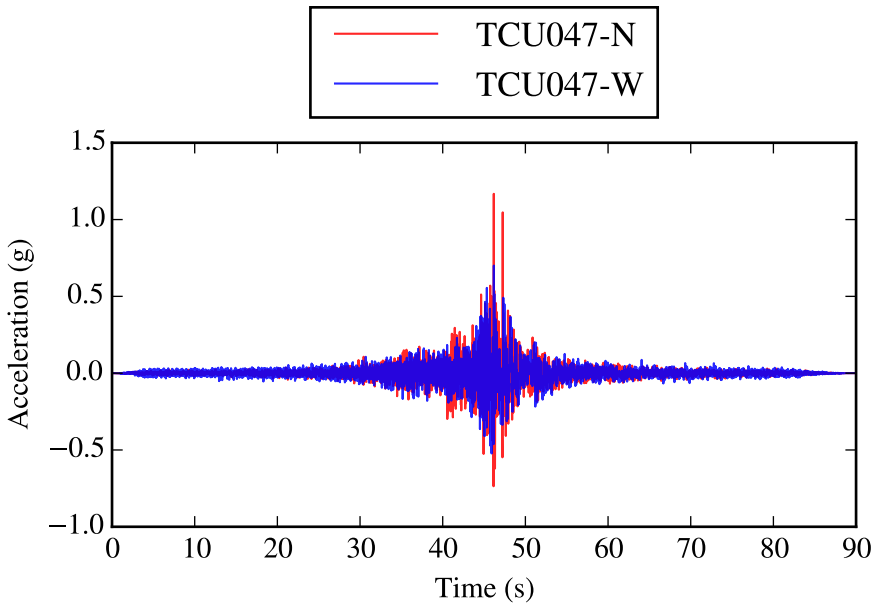


Figure F.332: CEUS 137: *TCU047-N* and *TCU047-W*.

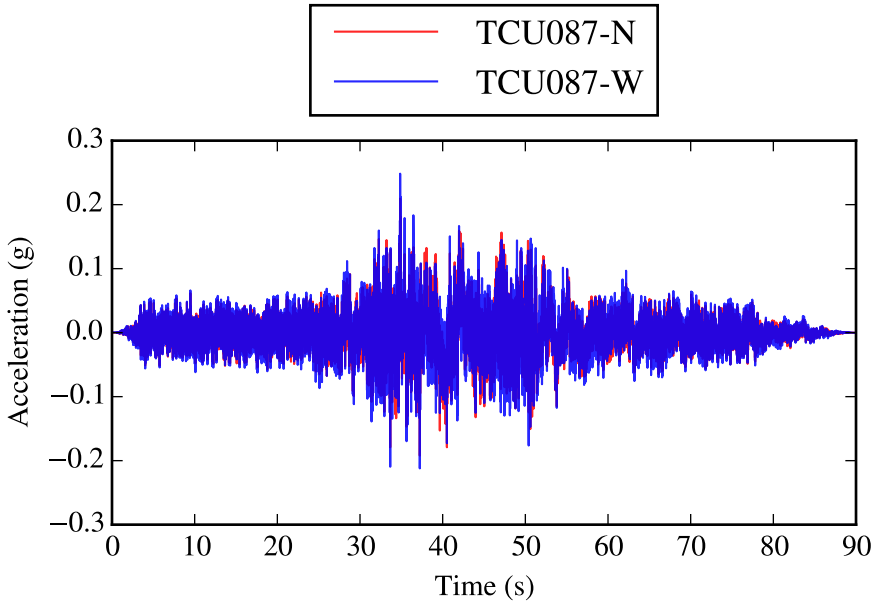


Figure F.333: CEUS 138: *TCU087-N* and *TCU087-W*.

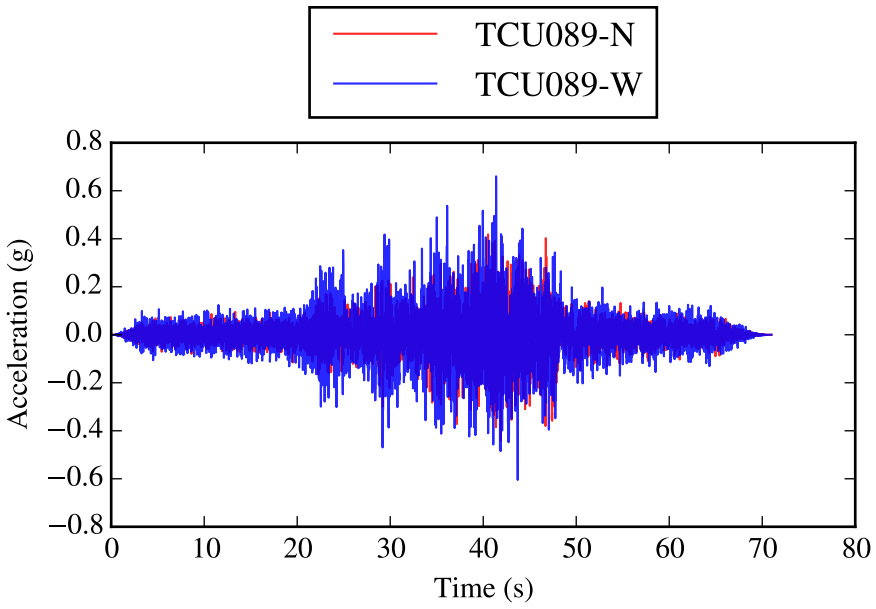


Figure F.334: CEUS 139: *TCU089-N* and *TCU089-W*.

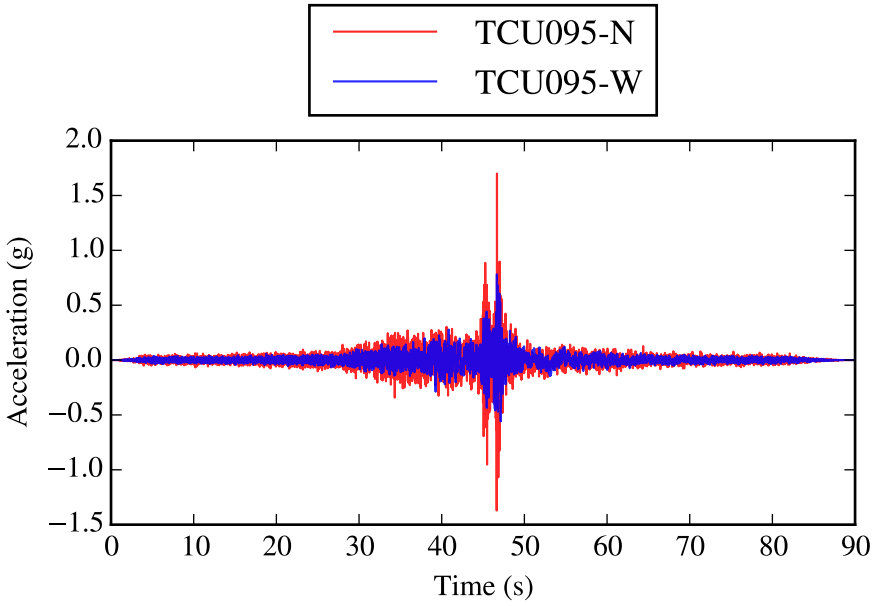


Figure F.335: CEUS 140: *TCU095-N* and *TCU095-W*.

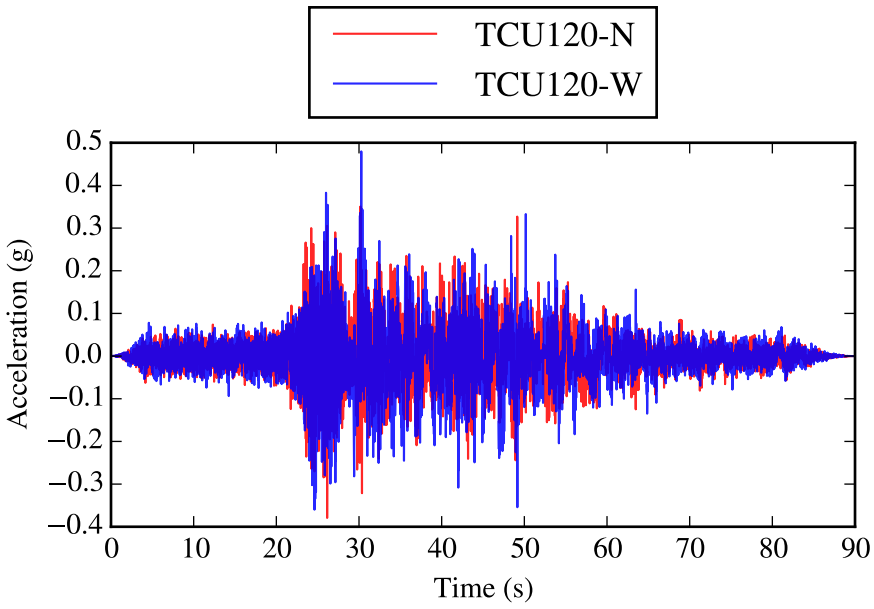


Figure F.336: CEUS 141: *TCU120-N* and *TCU120-W*.

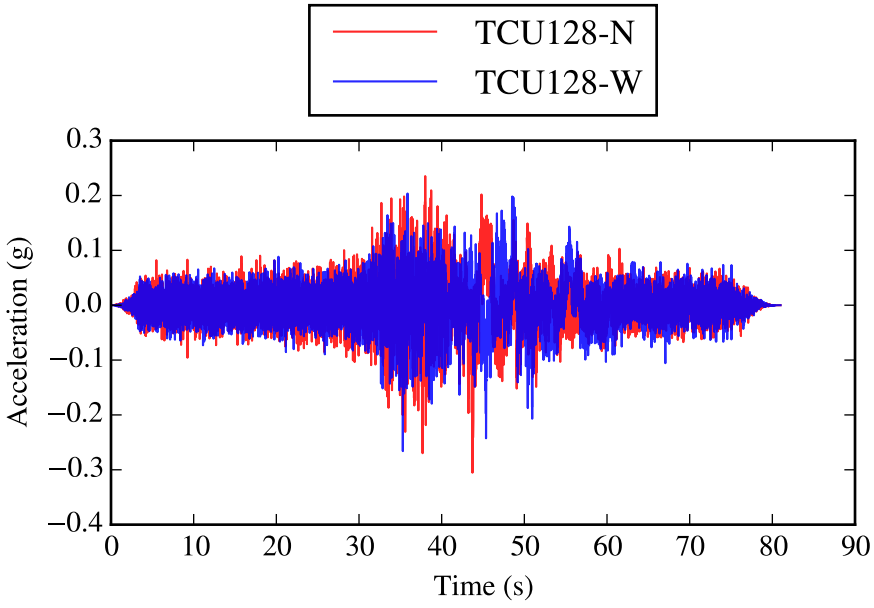


Figure F.337: CEUS 142: *TCU128-N* and *TCU128-W*.

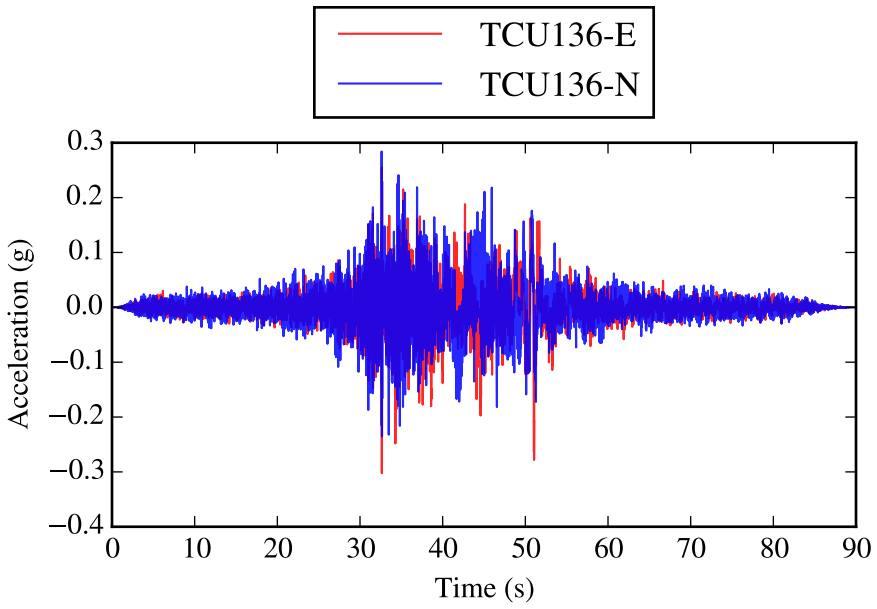


Figure F.338: CEUS 143: *TCU136-E* and *TCU136-N*.

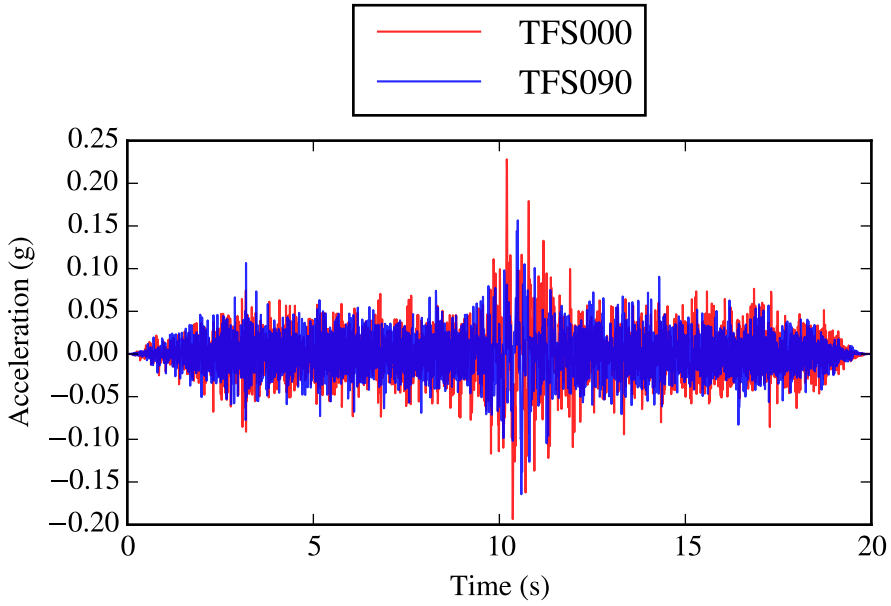


Figure F.339: CEUS 144: *TFS000* and *TFS090*.

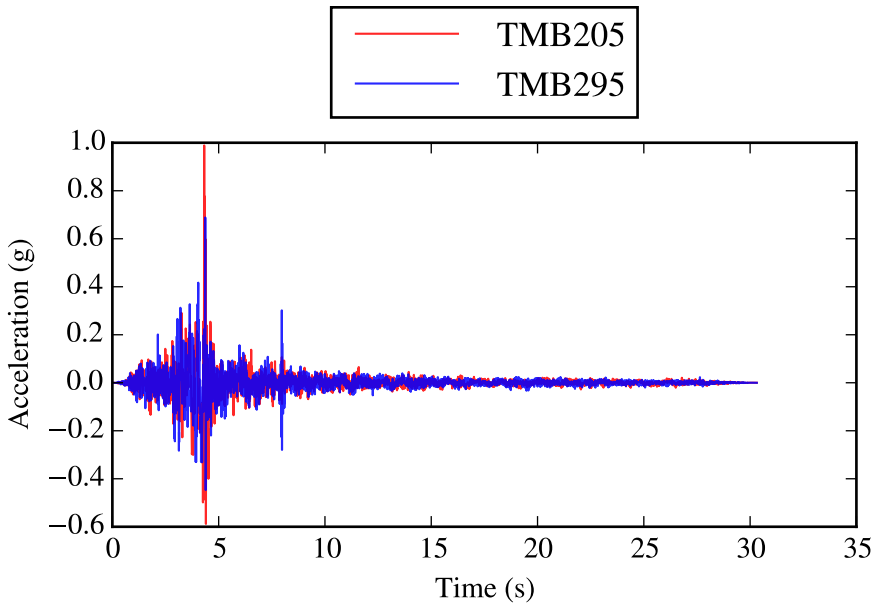


Figure F.340: CEUS 145: *TMB205* and *TMB295*.

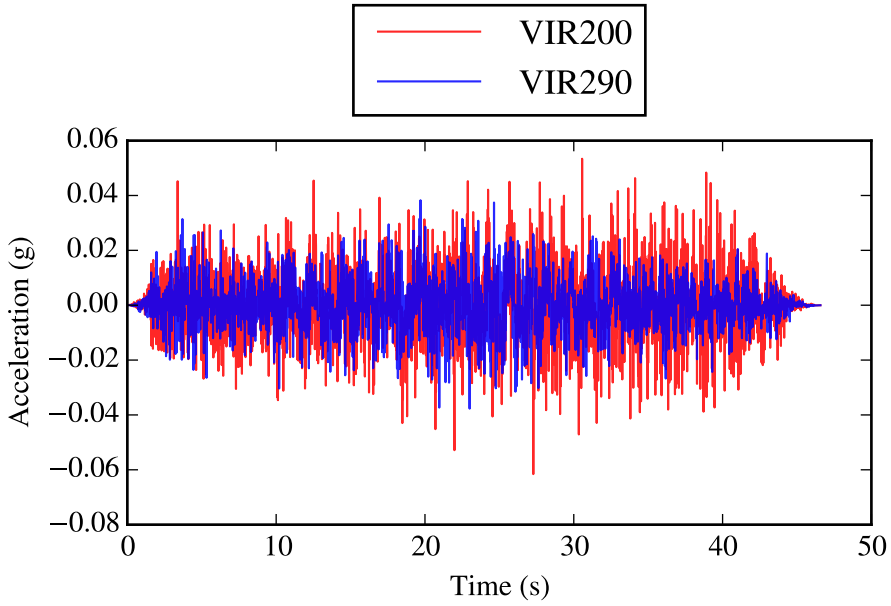


Figure F.341: CEUS 146: *VIR200* and *VIR290*.

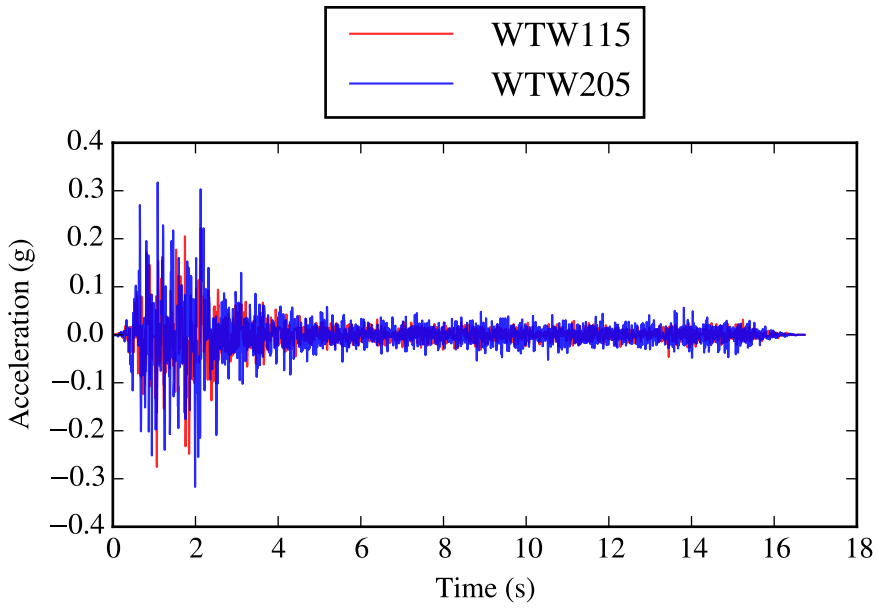


Figure F.342: CEUS 147: *WTW115* and *WTW205*.

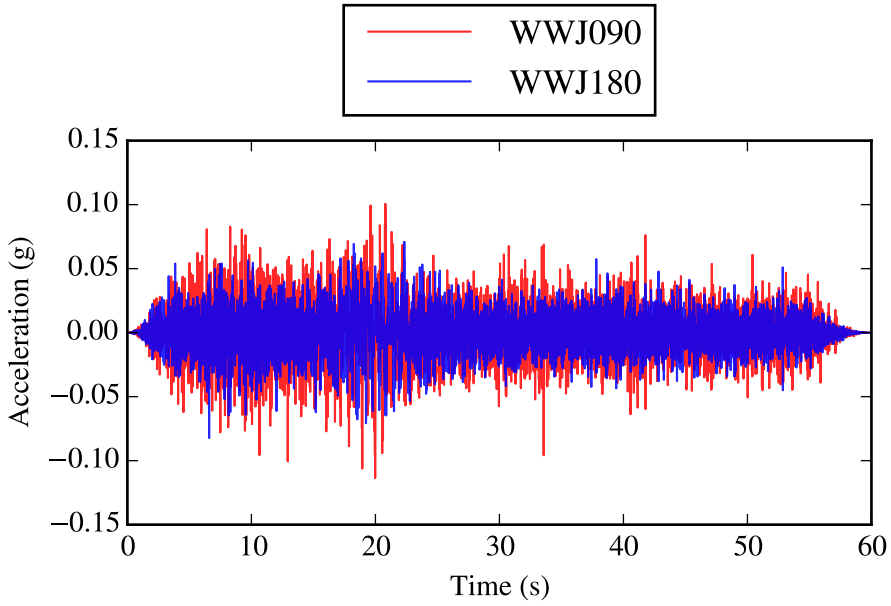


Figure F.343: CEUS 148: *WWJ090* and *WWJ180*.

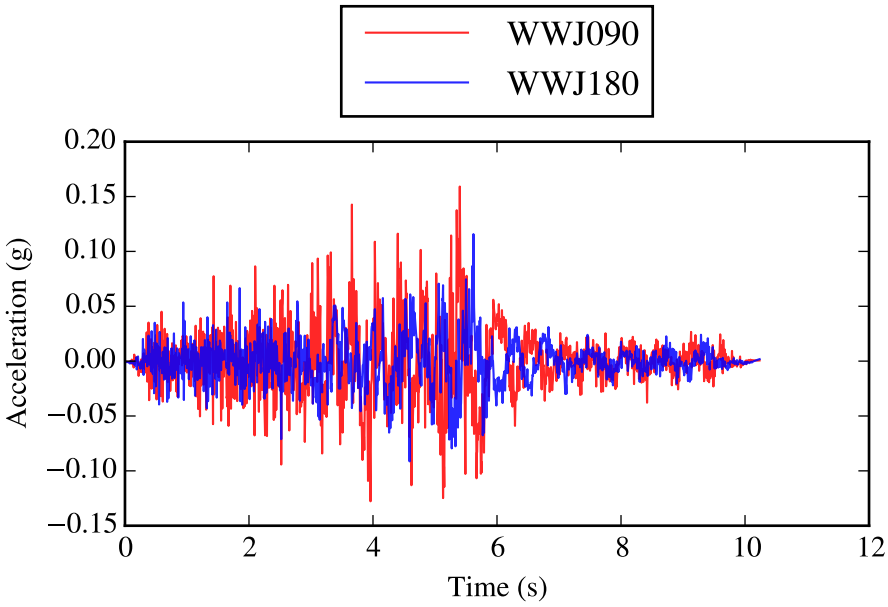


Figure F.344: CEUS 149: *WWJ090* and *WWJ180*.

Appendix G

Cyclic Simple Shear Testing Manual

G.1 Introduction

This manual is meant as a guide to aid the new student/researcher in the use of the GCTS equipment for cyclic direct simple shear testing at Virginia Tech. A separate manual should be available for cyclic triaxial testing. This treatment of the subject is by no means exhaustive; pertinent external literature should be reviewed as well as the manuals provided by GCTS.

A general overview of the equipment and setup are given herein. Additionally, best practices for sample preparation and equipment operation are discussed. In this manual, dry pluviation is presented as the method of sample preparation. However, with slight changes, other preparation methods can be easily implemented. Finally, discussion of data reduction is presented. The GCTS equipment has very high potential for academic research, however, it has its share of quirks. The over-arching purpose of this manual is to help the reader avoid the issues that prevent a successful test in the hope that proficiency can be gained in less time than that of the author.

G.1.1 Physical Setup

The GCTS equipment is located in Lab 15 of the W.C. English Geotechnical Lab at 260 Inventive Lane (previously 104 Plantation Road). Facing the equipment, the components of the equipment are, from left to right:

- **Hydraulic Pump**- This gray-colored tank with valves, pipes, and gauges provides the force to the hydraulic-operated normal and shear actuators. The oil level,

temperature, and condition should be periodically checked.

- **Load Frame and Pressure Cell Components**- This is where samples are prepared and tested.
- **PCP-200 Pressure Control Panel** or *Pressure Panel*- This panel contains a vacuum and a pressure gauge, vacuum and pressure controls, burettes, valves, and switches for the Top Back Pressure lines, Botttom Back Pressure lines, and the Cell Pressure lines.
- **APC UPS**- This black box provides power in case of a power outage. During a test it should give the user about 10 minutes to shutdown the controller and PC, preventing data loss. Note: the hydraulic pump is *not* connected to the UPS, so if the power goes out, your test it over anyway. The UPS should also prevent power surges, etc. Documentation (registration info, manual, etc) for this device should be found on one of the shelves with other documents.
- **SCON-200 Digital System Controller** or *SCON* or *controller*- This blue cube controls the entire system and collects all data. The firmware of the controller was updated to version 1.96 in the summer of 2013.
- **ULT-100 Ultrasonic Interface**- This gray box on top of the the SCON controls and receives information from the bender-element platens.
- **Lab PC**- This PC and associated software provide an interface to the SCON controller, and, thus, the rest of the equipment.
- **Welch Duoseal Vacuum Pump 1402**- Located on the counter behind the SCON, this vacuum pump, with the Nold DeAerator, deaerates water for use in saturated samples. The oil in the vacuum pump should be changed about once a year (which never really happens) or anytime contaminants can be seen in the oil. A manual for this particular pump can be found online without too much effort. Effort should be made to prevent the introduction of any contaminants, even water. A valve on the back of the pump allows for 'gas ballast.' Operating the pump with the gas ballast (with the valve open) prevents the condensation of water vapor within the pump at the slight sacrifice of vacuum pressure (see the manual for more information).
- **Geokon Nold DeAerator Model 2100**- On the wall above the vacuum pump, the deaerator removes the dissolved air from water to aid in saturation of samples. Only distilled water should be used in the deaerator. To prevent water from being pulled into the vacuum pump, the deaerator tank should be filled only to three-fourths of the tank capacity. Water is deaerated by running the impeller while a vacuum is applied. Ten or 15 minutes should remove most of the air; see the manual for more information.

Figure G.1 shows some of these components and their approximate layout in the lab.

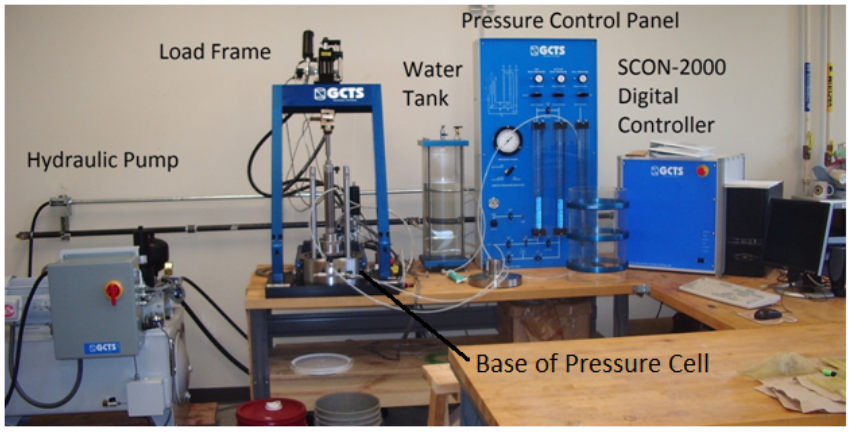


Figure G.1: The physical layout of some of the GCTS equipment.

G.1.2 The Cell

The cell is composed of many parts that fit together to allow the specimen to be pressurized (i.e. cell pressures greater than ambient air pressure). We will not be using several parts of the cell for cyclic simple shear testing, but I will introduce them all, anyways. These parts include:

Base The base of the cell is short, stainless steel cylinder mounted below the normal load frame (Figure G.2). On the front are valves and poke-throughs for the top back pressure lines, bottom back pressure lines, and the cell drain line. To the sides are poke-through ports for internal sensors. Attached at the rear is the ball bushing guide for the shear actuator piston.

Columns The columns support the cell top cap. See Figure G.3.

Cell Wall This acrylic plastic shell fits over the columns and top cap and onto the base of the cell.

Top Cap The top cap mounts onto the top of the columns and contains a ball bushing guide for the normal piston and poke-through ports for internal sensors.

Top Ring The top ring seals the cell assembly.

If filled with water, the maximum cell pressure is 1000 kPa. If the cell is air-filled, the maximum pressure is 500 kPa.

G.1.3 The Pressure Panel

The pressure panel contains valves and regulators to used to apply pressures (including vacuums) to the sample specimens. See Figure G.4. These pressures can be piped via

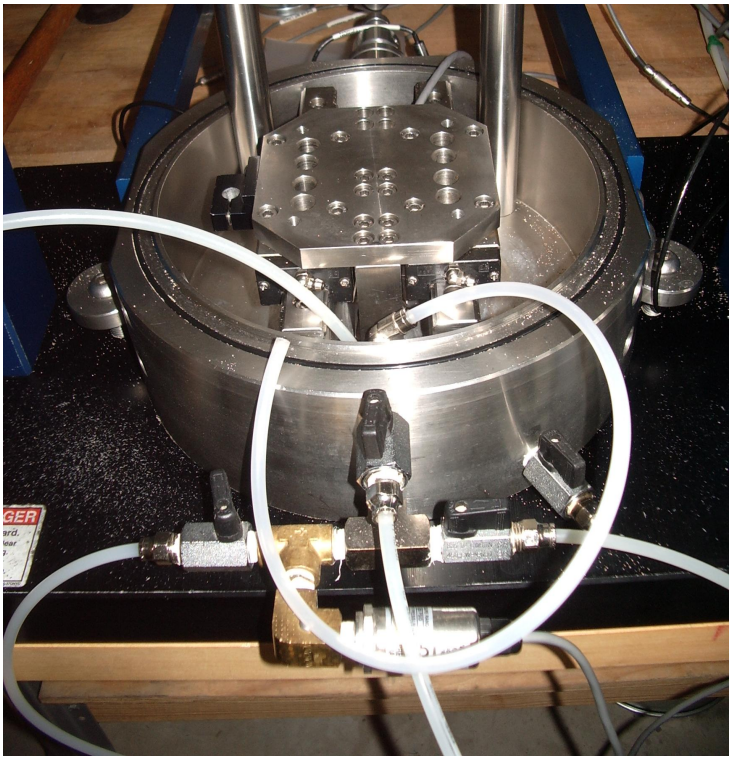


Figure G.2: The base of the cell with the shear carriage installed and two posts removed. Notice the back pressure sensor mounted at the front.

Figure G.3: Other parts of the cell. Clockwise from bottom left: the top cap, two of the four columns, the cell wall, and the top ring.

the cell lines, the bottom back pressure lines, and the top back pressure lines. At the top of the panel are the manual pressure regulators. Below these are the selector switches so that each line can be individually set to manual pressure control, vacuum control, vent, or servo pressure control. Below the selector switches are the burettes that show the level of water in each column, and below the burettes are valves for each line. To the left side of the panel is the vacuum pressure regulator, the vacuum port, the dial gauge, and the dial gauge selector switch. Note that the back pore pressure sensor has been moved from the back of the pressure panel to the front of the base of the cell.

G.1.4 Other Parts

A few other items are needed to perform a successful cyclic simple shear test. Figures G.5-G.7 show some of the items. They include:

- Top and bottom platens (plain or with bender elements)
- O-rings
- Brass confining rings (or a reinforced membrane)
- Flask with stopper and spout
- The internal LVDT assembly, brackets, and reaction plates (somewhat optional)
- Normal piston
- Shaft-locking collars
- The normal actuator contact plate
- Two normal-movement guides
- Cell-stiffening plate and reaction arms
- Top platen-to-normal-piston block

G.2 The Testing Cycle

This section covers the steps that need to be taken to perform a test in its entirety. It includes setup, sample preparation, equipment assembly, and cyclic loading.

G.2.1 Sample Preparation

Placement of Bottom Platen, Membrane, and Mold

1. Mount the bottom platen onto the shear carriage using the four appropriate bolts.

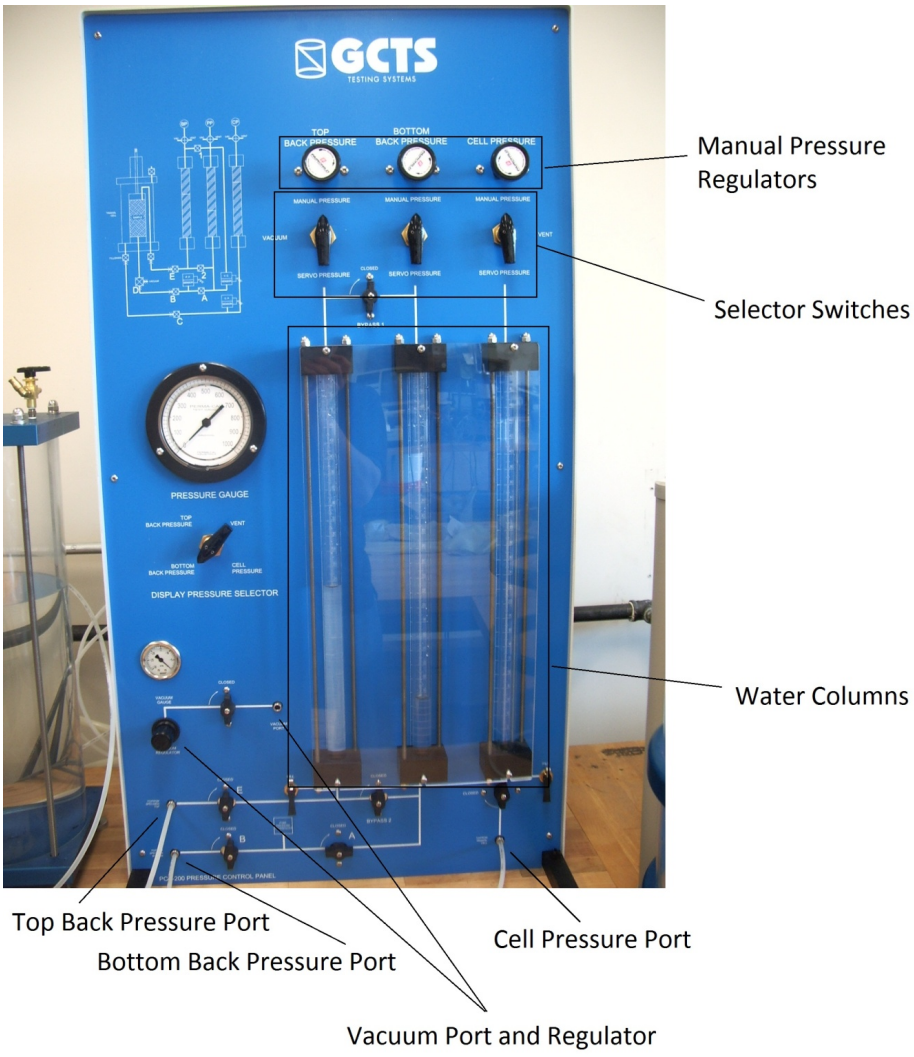


Figure G.4: The pressure panel.

Figure G.5: Clockwise from left: top platen, flask with stopper and spout, brass confining rings, membrane, o-rings, and, at center, bottom platen.

Figure G.6: Clockwise from the top left: the internal LVDT assembly, the internal shear LVDT reaction plate and bracket, the normal piston, large and small shaft-locking collars, and the normal actuator contact plate. In the center is the top-platen-to-normal-piston block.

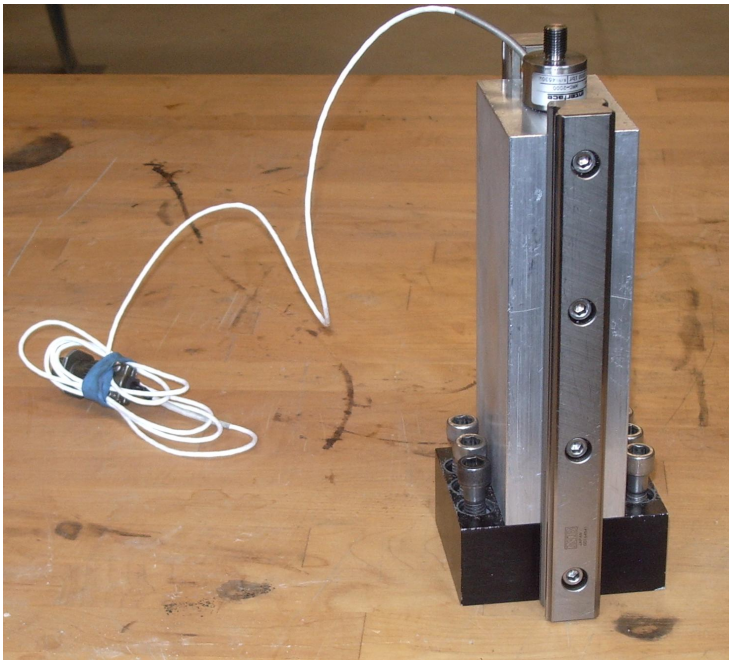


Figure G.7: The normal track assembly with the mini-load cell on top.

Either the bender-element or non-bender element platens may be used. I like to put an old piece of membrane under the platen to keep sand off of the carriage tracks (see Figure G.8). I also find it useful to remove the front two cell columns before I start, but depending on your preparation method, it may not be necessary.

2. Place the membrane on the bottom platen and secure it with two o-rings (Figure G.9).

I find it useful to use the mold as an o-ring stretcher (Figure G.10); I place the o-rings on the top of the mold and thread the unattached end of the membrane through the mold until the top of the mold fits around the bottom platen. The o-rings can then be rolled off onto the bottom platen.

3. Place the confining rings with their base on the bottom platen, taking care not to put a fold in the membrane.
4. Attach the vacuum hose to the side of the confining ring base and apply a small vacuum (probably no more than 2 or 3 inches of mercury).
5. Gently pull the free end of membrane up and around the outside of the confining rings. Make sure there are no wrinkles in the membrane inside the confining rings

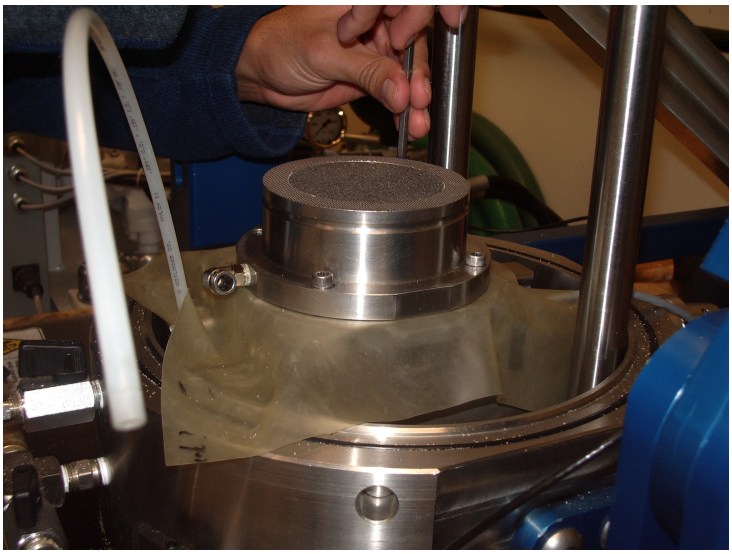


Figure G.8: Mounting the bottom platen onto the shear carriage.

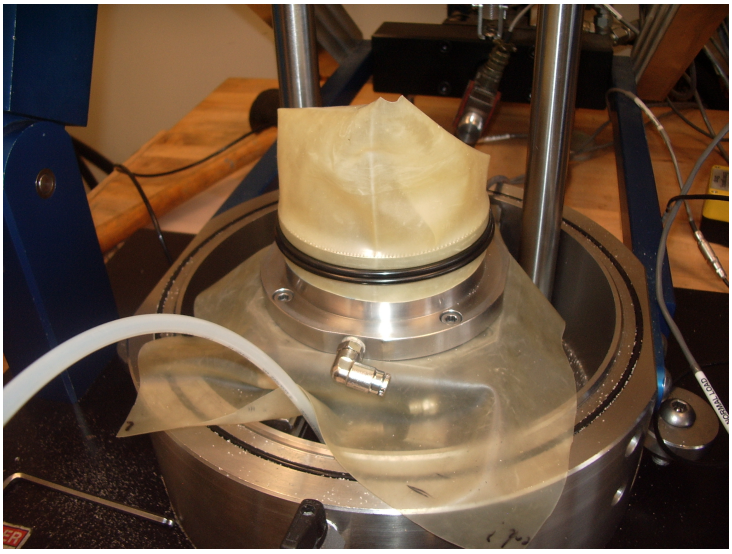


Figure G.9: Membrane attached to the bottom platen.

Figure G.10: Using the mold as an o-ring stretcher.

and that the membrane is pulled tight to the inside of the rings by the vacuum. See Figure G.11.

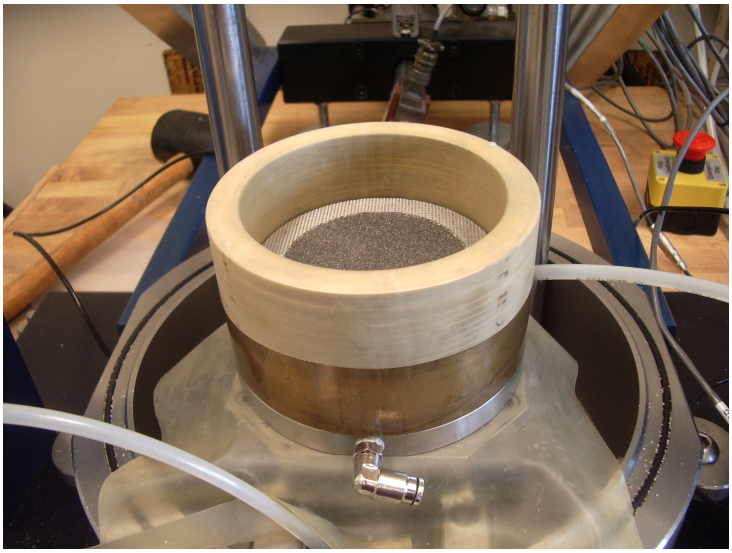


Figure G.11: Bottom platen with membrane and confining rings.

At this point I like to clamp a plastic 'fence' around the outside of the rings, as shown in Figure G.12. It keeps me from spilling a lot of sand into the bottom of the cell. This 'fence' is cut from a dollar-store cutting board.

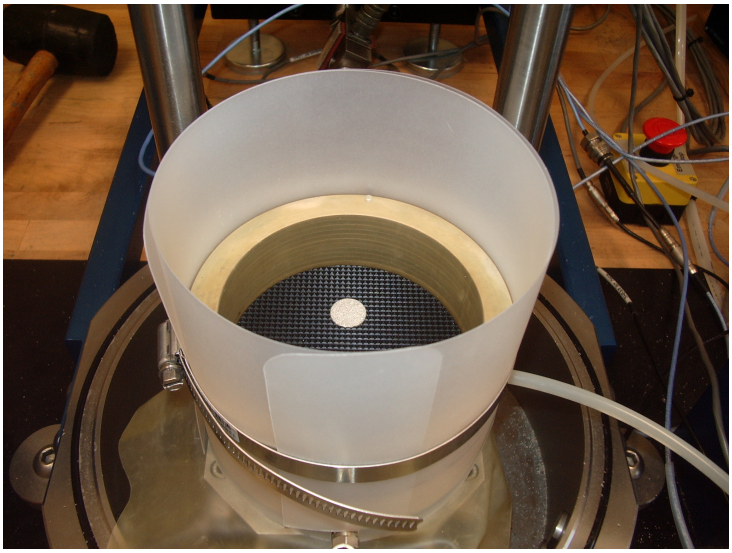


Figure G.12: Bottom bender-element platen with rings and 'fence.'

Dry Pluviation

Wet pluviation mimics the natural fluvian and lacustrine deposition of soils (Vaid and Negussey, 1984, 1988), and works best for poorly-graded materials that are less likely to experience particle segregation. However, dry pluviation is, in many ways, easier to perform and doesn't allow the segregation of particles by size as much as wet pluviation. Because we won't need to saturate our speciments, dry pluviation is the method presented here.

The dry (or *air*) pluviation procedure outlined here closely follows that of Vaid and Negussey (1988).

> If your material is fine-grained, well-graded, or cannot be dried, another method of sample preparation is warranted. Moist tamping and slurry deposition are popular alternatives (see Bradshaw and Baxter (2007); Wang et al. (2011)).

1. Place a generous amount of dry sand into a flask. The flask should have an accompanying stopper and funnel (see Figure G.13).
2. With a gentle swirling motion, pour sand from the flask, through the stopper and funnel, into the confining rings. Slightly overfill the rings so that the sand is heaped above the top of the rings (Figure G.14).

> The density of the sand can be controlled by the pour height and the size of the funnel. Some experimentation may be needed to get the correct densities.

Figure G.13: Flask with sand, stopper, and funnel.

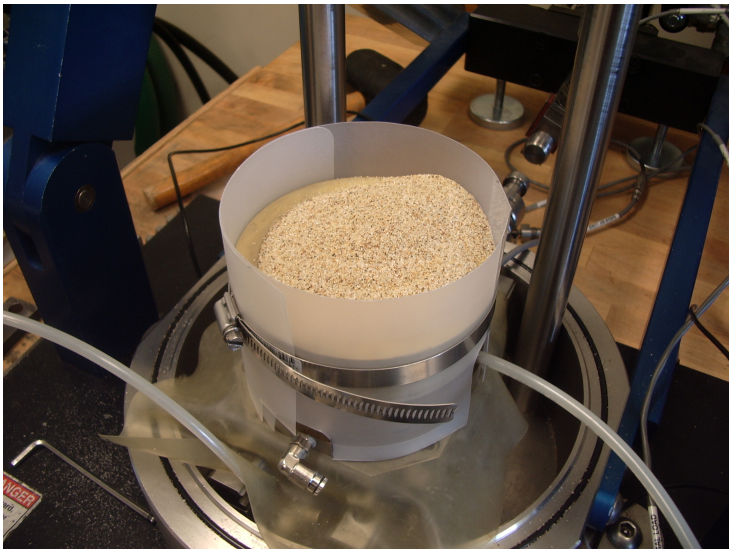


Figure G.14: Sample after dry pluviation sand placement. Notice the heaped sand.

3. Using a straight edge and as few strokes as possible, scrape the extra sand from the top of the rings, leaving the sand level flush with the top of the rings (see Figure G.15.

> I use a mini-vacuum to clean up the excess sand from both the top of the rings and around the base of the cell (Figure G.16. I created this particular vacuum by attaching a small, thin cloth bag to one end of a T. The opposite side of the T has air pressure coming in. The remaining part of the T has the vacuum nozzle attached to it. It works as a crude venturi vacuum.

4. Replace the cell columns, if they were removed previously. Place the top of the cell onto the columns (no need to bolt it on).

5. Attach the top-platen-to-normal-piston block onto the top platen. Place this assembly on a beam resting on the confining rings. Figure G.17 shows how it should look.

> When attaching the top-platen-to-normal-piston block to the top platen, it needs to be tightened very little. Otherwise you will disturb the top surface of the specimen when you try to remove it.

6. Lower the normal actuator piston through the cell top cap and screw it onto the top-platen-to-normal-piston block. Lift the entire assembly, remove the beam, and gently lower the top platen onto the top of the specimen. Make sure the top platen is straight before you place it. You may need move the shear carriage forward or

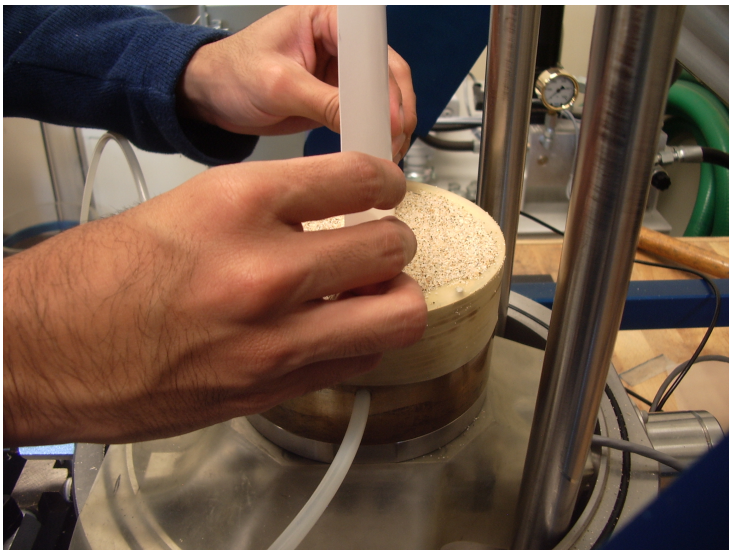


Figure G.15: Scraping the excess sand off of the top of the confining rings.

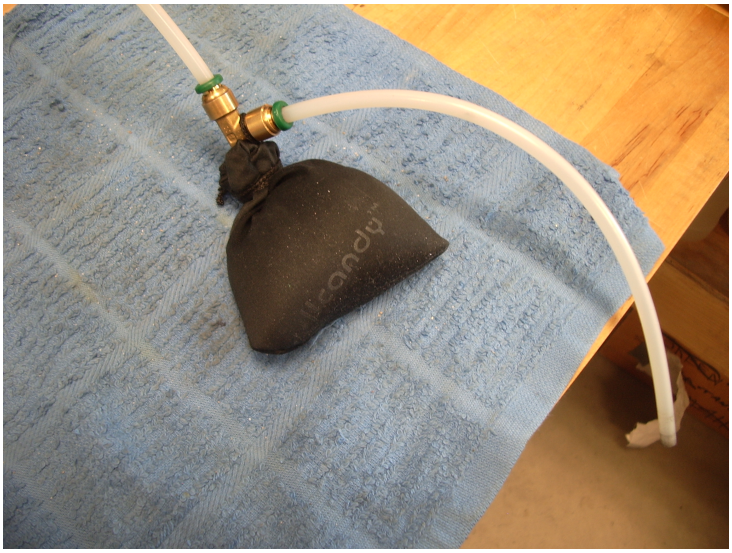


Figure G.16: The mini-vacuum that I use to clean up sand.

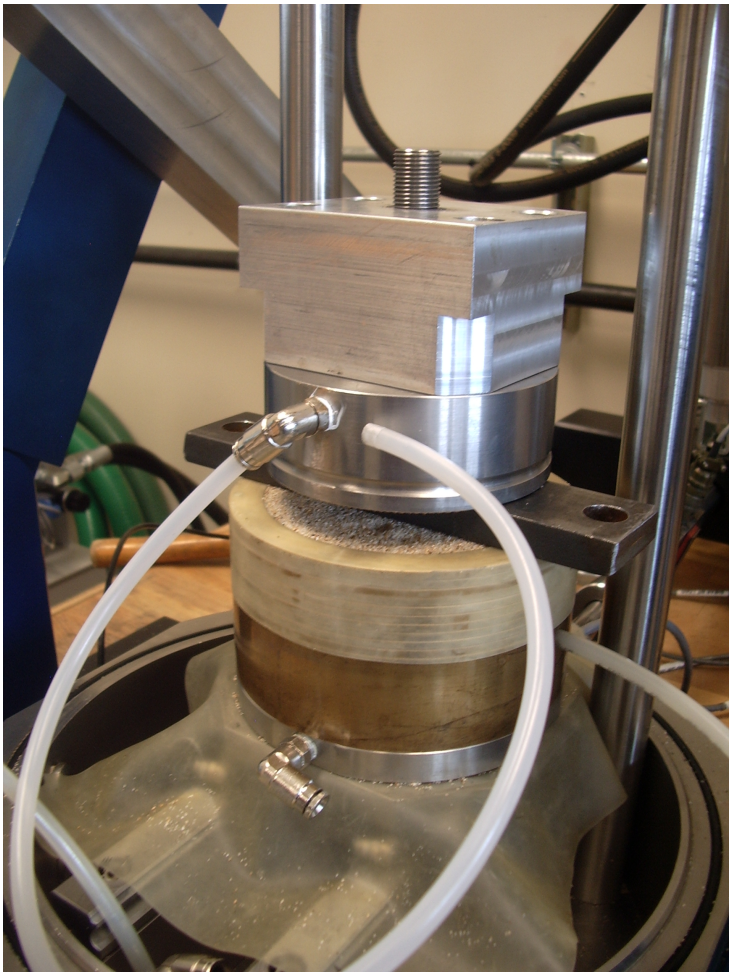


Figure G.17: The top platen with the top-platen-to-normal-piston block on top of the specimen-in-preparation

backward so that the top platen is centered inside the confining rings. Tighten the shear piston locking collar.

> As I lower the top platen, I place the beam in between the columns and the top platen, as shown in Figure G.18. This helps me place the top platen straight.

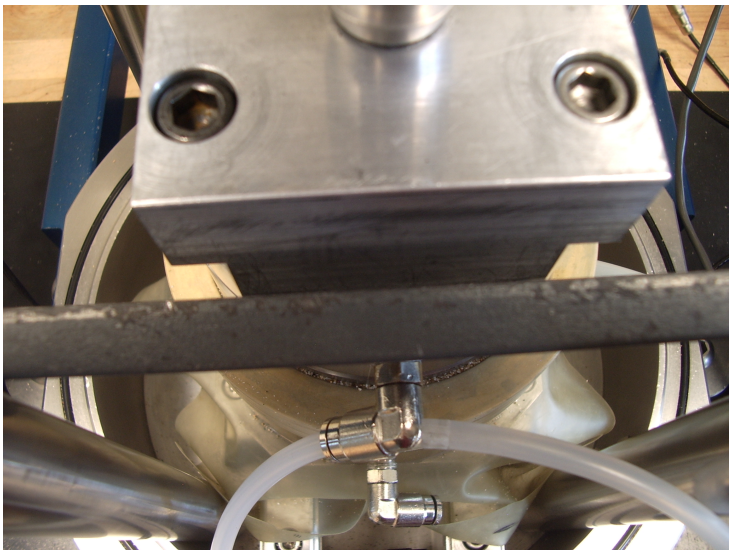


Figure G.18: Using the beam to straighten the orientation of the top platen.

7. Turn off the vacuum and remove the vacuum hose from the side of the confining rings. Roll the membrane up and onto the top platen. Place the o-ring(s) onto the top platen to hold the membrane in place.
8. At this point, vibrate or otherwise densify the specimen, if needed or wanted.

> I like to lightly tap the top of the normal actuator with a rubber mallet. It slightly densifies my sample and makes me feel confident that the top platen interfaces well with the specimen.

9. Apply a vacuum to the specimen (20-30 kPa) and remove the normal piston, the top-platen-to-normal-piston block, and the cell top cap.
10. Measure and record the height of the specimen.

G.2.2 Assembly of Equipment

In this section, I list the order in which I assemble the GCTS equipment for a cyclic simple shear test. Some of the orderings are important; others are not. However, I will not take the time to tell you which is which. Good luck!

1. Attach the internal shear LVDT reaction plate and bracket to the right side of the shear carriage.
2. Place the internal LVDT assembly on top of the top platen (includes the internal shear LVDT bracket, the internal normal LVDT reaction plate, and the horizontal plate that holds both).
3. Place both of the black normal-movement guides on the cell columns. (They can be seen in Figure G.19.)

   I try to place these as far apart as possible. The lower one I put just above the internal LVDT assembly, the upper one just below the aluminum cell stiffener plate. It works best to only tighten one corner until all three pieces (i.e. the two black normal-movement guides and the stiffener plate) are in place.

4. Place the aluminum-colored cell stiffener plate onto the cell columns. Insert and tighten all bolts (the bolts that connect the stiffener plate to the arms, those on the black guides, and the rest on the cell stiffener plate) . Figure G.19 shows how it should look up to this point.
5. Gently lower the normal track assembly (Figure G.7) onto the LVDT assembly and top platen. Bolt the normal track assembly to the top platen (Figure G.20).
6. Place the small locking collar onto the mini normal load cell.
7. Set the top of the cell onto the support columns. Start screwing in the corresponding bolts into the columns, but do not completely tighten them down.
8. Insert the normal piston into the top cap; screw it onto the mini normal load cell. Do not screw it tight; leave a small gap between the normal piston and the LVDT. If it is tightened, it will cause the measured load values to go awry.
9. Cautiously tighten the top cap bolts and the set screws center the normal track assembly. (There are four set screws, two in each of the black normal-movement guides.) Make sure that no moments are applied to the mini normal load cell.
10. Place the large locking collar around the normal piston and screw the normal actuator contact plate into the end of the piston. Zero out the mini normal load cell to the appropriate value (17.13 lbs for the non-bender-element platens).
11. Install and zero the internal LVDTs.
12. With the vacuum still applied to the specimen, place some weights onto the normal piston to double-check the mini normal load cell calibration. Calibrate the load cell, if needed.
13. Bring the normal loading frame into its vertical position.
14. Turn on the pump.
15. Open the Outputs Function Window. Switch the output to Shear Actuator and the Feedback to 'T-Shear Stress' or 'Shear Load.' In the PID window, make sure the corresponding 'P' value is fairly low. Make sure the value is set at zero, and turn on the shear actuator.

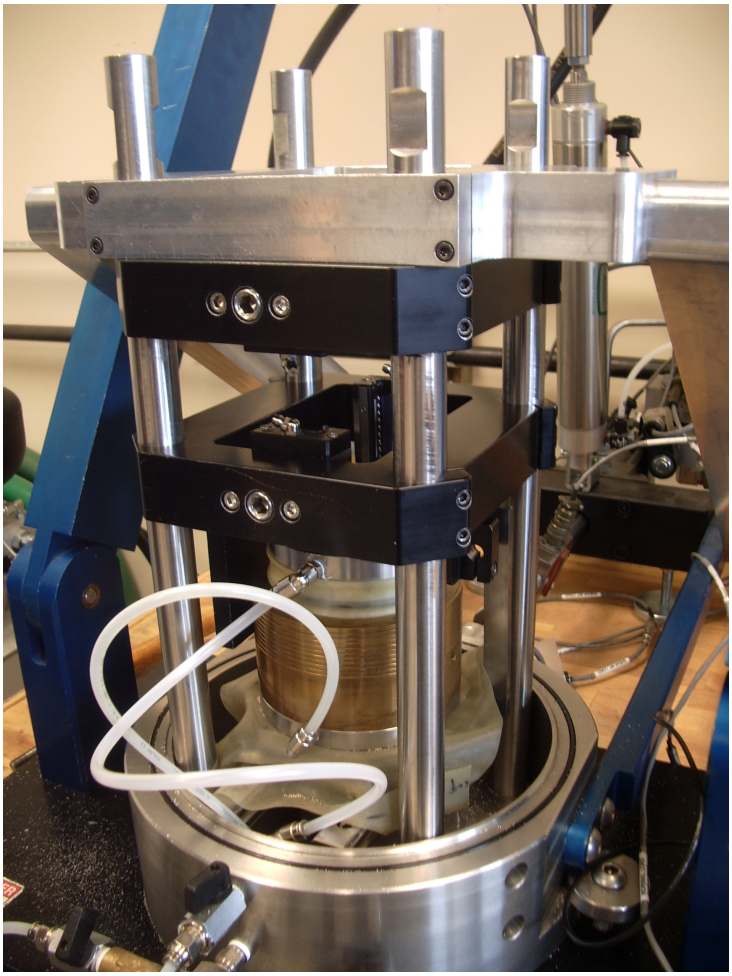


Figure G.19: Partial assembly of the internal cell components including the black normal-movement guides, the silver-colored stiffener plate, and the internal LVDT reaction plates and other parts.

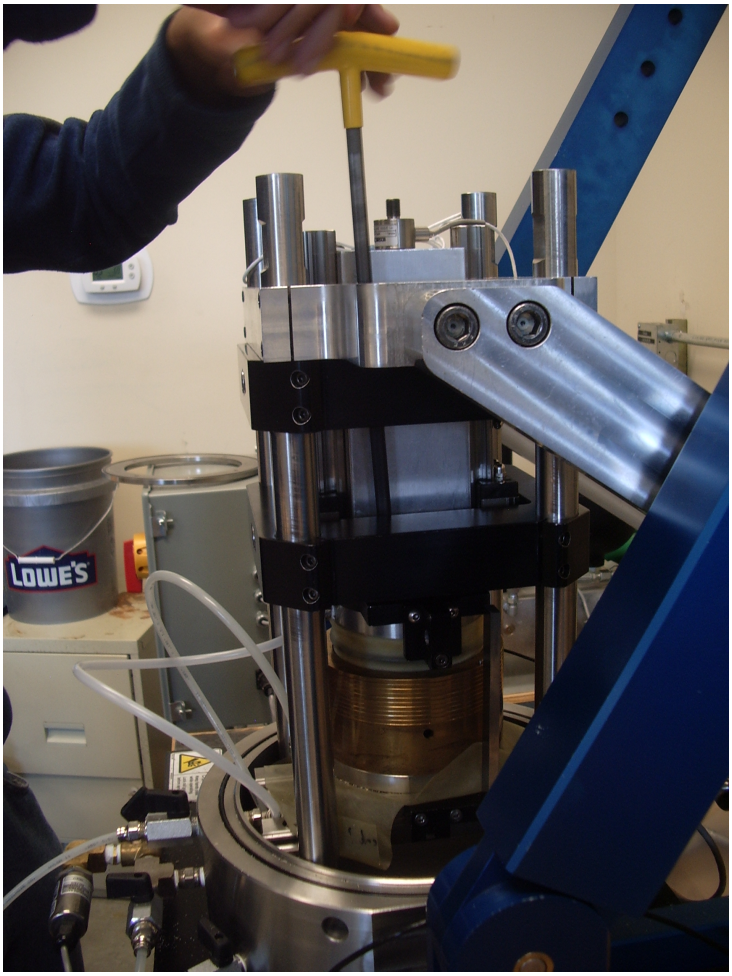


Figure G.20: Bolting the normal track assembly onto the top platen.

16. Cautiously pull on the end of the external shear load cell to cause the shear actuator to move into position. Allow it to make contact with the piston going into the cell, and firmly attach the two parts of the shear piston by screwing on the mating collar.

17. Set the Feedback to 'Shear Displacement'. Unlock the piston at the collar, remove the shear pins in the confining ring set, and, using the Inputs Offset tool, set the shear displacement equal to that of the Internal Shear LVDT.

18. In the Outputs Function window, set the Output to Normal Actuator. Since we are using the internal mini load cell, I manually lower the normal actuator. To do so, set the Feedback to 'Normal Displacement.' Turn on the actuator and move it until contact is made.

    If you know the displacement at which the two piston parts meet, type in a distance about 5 mm from contact. For example, a previous test set the distance to contact at 0 mm, so I type in -5 mm and turn on the normal actuator in the software. Once the actuator has arrived within about 5 mm, I click the down arrow of the Set-Point scrollbar until the distance is down below about 0.5 mm. Then I click the 'Fine Adj' checkbox and continue to lower the actuator until I see the normal load increase.

19. At this point, I change the Feedback to 'Sn - Normal Stress.' Next, use the Inputs Offset tool to set the normal displacement value to that of the internal normal LVDT value.

20. Remove the applied vacuum, and open the back pressure lines to the sample.

    You are now ready to initiate a test.

G.2.3 GCTS Control of Testing

This section will guide you through the actual running of the test in which the GCTS system controls the specimen and collects data.

It is assumed that a suitable test file already exists. If not, you need to create one. See the GCTS manuals and section G.4.1 in this manual. *This* section will walk you through starting and running a test with a consolidation and cyclic loading phase.

1. Click on the 'P/S/S' (or 'Projects') button to open the 'Projects/Samples/Specimen' window. Select the appropriate project and sample, or create new ones.

2. Click 'New' to create a new specimen. Give an appropriate 'ID' (and description, if desired). Fill in the boxes for the measurements of height and diameter. On the next tab, change the specific gravity (G_s) as needed. When ready, click 'Ok.'

3. Select the correct test. Click on the 'Edit' button if you need to make changes. When finished, click 'Ok.'

4. A screen will ask you if you want to apply a 5 kPa axial stress difference. If your normal actuator is already in contact, click 'Skip.' (I've had problems with this step in the past, so I always skip it. Your mileage may vary.)

5. A screen may come up asking you if you want to correct the specimen dimensions. Choose how you deem prudent and continue.

> Now the test window should have come up. The test window shows the status of the test including the stage and phase of the test. It also has buttons at the top to stop and pause the test. If you have chosen the correct test file to run, the normal stress should increase until it reaches the predefined effective consolidation pressure. This is the consolidation phase. You should allow your specimen to consolidate for a minute or two, at least. I generally let it sit for 5 minutes during the consolidation phase, for consistency.

6. When normal displacements have steadied, tighten the shaft-locking collars around the normal actuator.

> I start with the largest collar, at the top. Then I continue to the attached collar (on the top of the normal piston bushing), and, finally, I finish with the smallest collar under the top cap. If you are not careful, you may disturb the sample while tightening the collars, so tighten slowly! If the normal actuator begins to have a high frequency response, your P value is probably too high.

7. In the test window, click 'Next' to go to the cyclic loading phase. If you are sure, click 'Ok.'

8. Behold the cyclic loading phase! Depending on your settings in the test setup, you may need to end the test manually by clicking on the 'Stop' button. Otherwise, let the test run its course.

> From the PID window, I like to open the 'Tune Chart.' If the feedback doesn't match the command of the normal actuator, you may need to adjust the P-I-D values. Usually, I only have to adjust the P values.

9. Wait for the GCTS system to transfer the test file from the SCON to the PC.

G.2.4 Cleanup/Tear Down

After the test file has transferred from the SCON to the PC, you can regain control of the GCTS system. A few final steps are needed before we are finished.

1. Unscrew the mating collar from the shear actuator.
2. Close the test window and the 'Projects/Samples/Specimens' window. Find your 'Outputs Function' window.
3. Change the 'Output' to the Shear Actuator.

4. Change the shear stress to zero. Push the shear actuator until it moves all the way back.
5. In the 'Outputs Function' window, turn off the shear actuator.

   At this point, I like to change the feedback to 'Shear Load' so that it is ready for the next test.

6. Again in the 'Outputs Function' window, switch the 'Output' to Normal Actuator.
7. With the feedback set to 'Normal Displacement,' change the set point to -25 mm in order to lift the normal actuator out of the way.
8. Once the normal actuator has finished moving, turn off the normal actuator in the Outputs Function window. Turn off the pump in the 'Hydraulic Tool' window.
9. Tilt the normal load frame back.
10. Disassemble the equipment. You can mix up the order however you want, but here is a recommended order:
    (a) Loosen the internal shear LVDT.
    (b) Remove the internal shear LVDT reaction plate and bracket.
    (c) Remove the internal normal LVDT.
    (d) Loosen the set screws in the normal movement guide (that center the normal track assembly).
    (e) Remove the normal actuator contact plate from the end of the normal piston.
    (f) Loosen and remove the large shaft-locking collar.
    (g) Loosen the shaft-locking collar attached to the normal piston bushing.
    (h) Loosen the small shaft-locking collar.
    (i) Unscrew and remove the normal piston.
    (j) Remove the top cap by first removing the bolts holding it down.
    (k) Remove the small shaft-locking collar.
    (l) Remove the normal track assembly.
    (m) Remove the cell-stiffener plate and the two normal movement guides. I find it works best to first loosen all of the small bolts on the sides of both the cell-stiffener plate and the normal guides before trying to lift off any of the pieces.
    (n) Remove the internal shear LVDT from the internal LVDT assembly and remove this assembly.
    (o) Unscrew and remove the two front columns.
11. Remove the top o-ring(s).

12. Remove the top platen and brush the sand off of it into a bowl.
13. Remove the confining rings and put the pins back in.
14. Remove the screws holding down the bottom plate.
15. Dump the specimen into the same bowl as before. Brush off the the membrane and bottom platen.
16. Place the bowl into the oven and allow the specimen to dry for approximately 24 hours.
17. When the specimen has dried, mass the specimen and record. This is needed to calculate the relative density of the specimen.

G.3 Data Reduction

The data reduction can be performed using most data processing software. Before beginning data reduction, it is first necessary to obtain the output file from the GCTS software.

1. Open the 'P/S/S' window in the GCTS CATS software. Navigate through the projects and samples to the specimen you would like to examine.
2. Select the specimen of interest.
3. In the toolbar at the top of the 'P/S/S' window, click on the 'Edit' button. Click 'Ok' to dismiss the popup.
4. Find the appropriate space and enter the dry mass of the specimen. Click 'Ok' when done.
5. In the toolbar at the top of the 'P/S/S' window, click on the 'Export' button.
6. In the new dialog window, select an output format and the data that you would like to export. I prefer the .csv format and I select all data to be exported.
7. When finished, click 'Ok' and pick a folder to which you would like to export the data file.

For cyclic tests, several values must be calculated from the output file. These include the void ratio and relative density of the sample, the cyclic stress ratio, the number of cycles to liquefaction, and the dissipated energy to liquefaction.

G.3.1 Void Ratio and Relative Density

The void ratio (e) is calculated using the total specimen volume and the volume of solids (V_s) in the specimen. The volume of solids is calculated as

$$V_s = \frac{m_{specimen}}{G_s \rho_{water}} \tag{G.1}$$

where $m_{specimen}$ is the mass of the specimen, G_s is the specific gravity of the sample, and ρ_{water} is the density of water. The void ratio is

$$e = \frac{V_{specimen} - V_s}{V_s} = \frac{V_v}{V_s} \tag{G.2}$$

where $V_{specimen}$ is the volume of the entire specimen and V_v is the volume of voids in the specimen.

The relative density (D_r) is calculated as

$$D_r = \frac{e_{max} - e}{e_{max} - e_{min}} * 100\% \tag{G.3}$$

where e_{max} and e_{min} are the maximum and minimum void ratios of the sample, respectively.

G.3.2 Number of Cycles to Liquefaction

There are several methods used to determine the point of liquefaction during cyclic loading. Wu et al. (2004) discusses many of these methods. The most common methods define the onset of initial liquefaction as either the point at which the pore pressure ratio (r_u) is equal to unity, or point at which a certain strain is reached. You will have to decide the correct criteria for your laboratory testing needs.

G.3.3 Calculation of Dissipated Energy

Dissipated energy is defined as the area bound by the hysteresis loops of a stress-strain plot (see Figure G.21). For a simple shear test, it can be calculated by using the trapezoidal rule:

$$\Delta W = \frac{1}{2}\sum_{i=1}^{n-1}(\tau_{i+1} + \tau_i)(\gamma_{i+1} - \gamma_i) \tag{G.4}$$

where ΔW is dissipated energy in units of stress, n is the number of data points in the stress and strain time histories, τ is the shear stress, and γ is the shear strain.

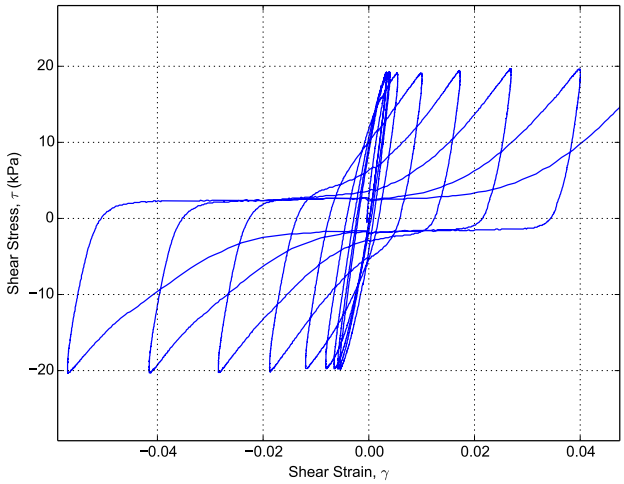


Figure G.21: The hysteresis loops of a stress versus strain plot.

G.3.4 Plotting Results

Plotting results is an important part of data reduction. Plots can help you diagnose poor tests and discover new results. You should plot both results from a single test, and the

results for multiple tests. Plots for the results of a single test may be a plot showing stress versus strain, a stress time history, a strain time history, a pore pressure time history, etc. Plots of multiple test results might include CSR versus number of cycles to liquefaction (on log-log scales, the trend should be linear), and dissipated energy versus effective consolidation pressure or versus relative density. Figure G.22 shows an example of a diagnostic output plot.

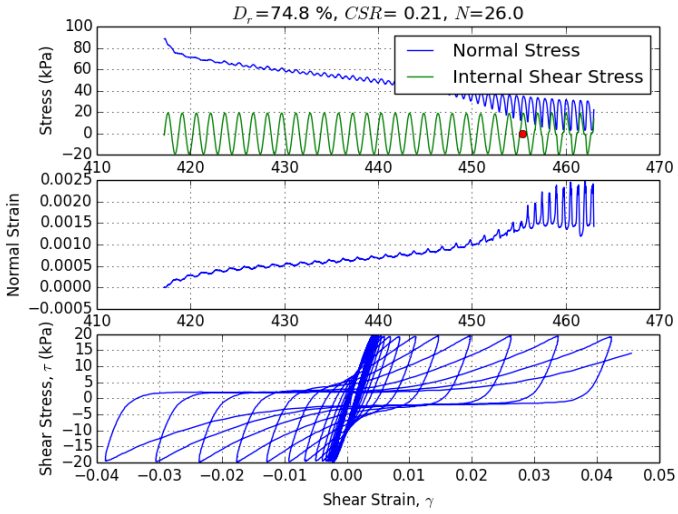


Figure G.22: An example of an output plot.

G.4 Maintenance/Preparation

G.4.1 Creating a Cyclic Simple Shear Test File

A cyclic simple shear test consists of three stages: a ramp up to confining pressure, consolidation, and cyclic loading. Additionally, you may wish to add a final phase to move the normal actuator out of the way at the end of the test. This would allow you to disassemble the equipment while the data file transfers. In this section, only the ramp, consolidation, and cyclic loading stages will be created. You are left to your own devices to add any other stages.

> I refer to *stages* here, but, strictly speaking and according to the GCTS system, we will be creating three separate *phases* of a 'Universal' *stage*.

1. Open the 'Test Setup' window (Figure G.23), and select 'New.'

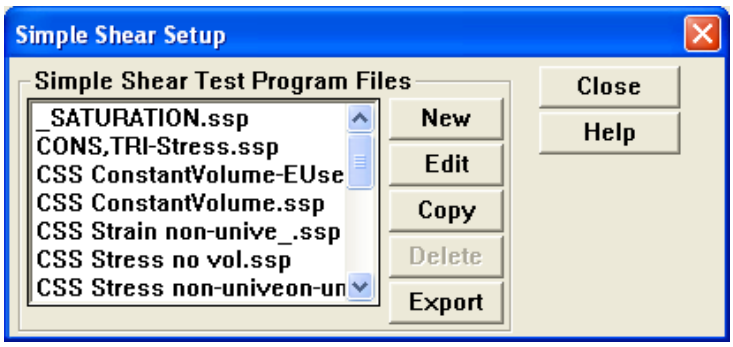


Figure G.23: The 'Test Setup' dialog box.

2. In the 'Simple Shear Program Definition' dialog box (Figure G.24) , type in the appropriate information into the 'ID' and 'Description' boxes. Click on the 'Save Objects' button.
3. In the 'Test Data Save Options' dialog box (Figure G.25), select the inputs and analog outputs that you are interested in saving. I generally select them all. Close this dialog when you are finished.
4. Once again in the 'Simple Shear Program Definition' dialog box, click on the 'New' button to create a new stage. In this dialog box (Figure G.26), select 'Universal' and 'Ok.'
5. In the 'Simple Shear Program Stage [1]: Universal' dialog box (Figure G.27), click the desired strain-handling boxes and then choose 'New' to create a new universal program.

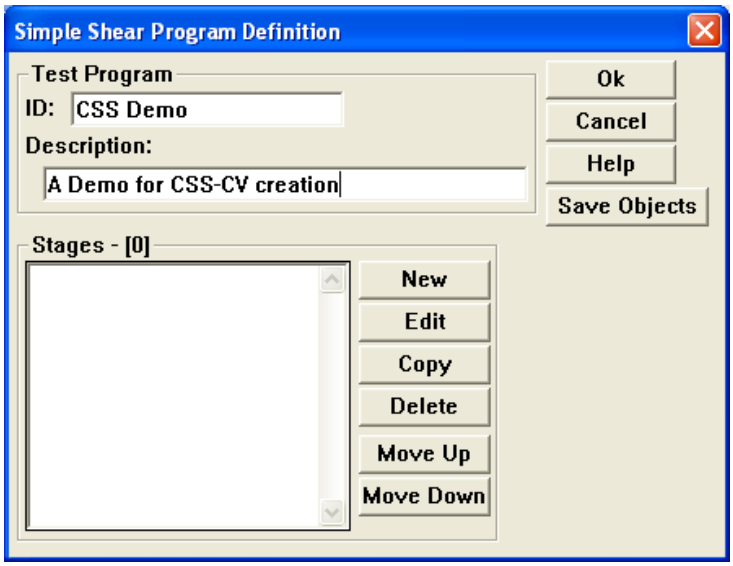


Figure G.24: The 'Simple Shear Program Definition' dialog box.

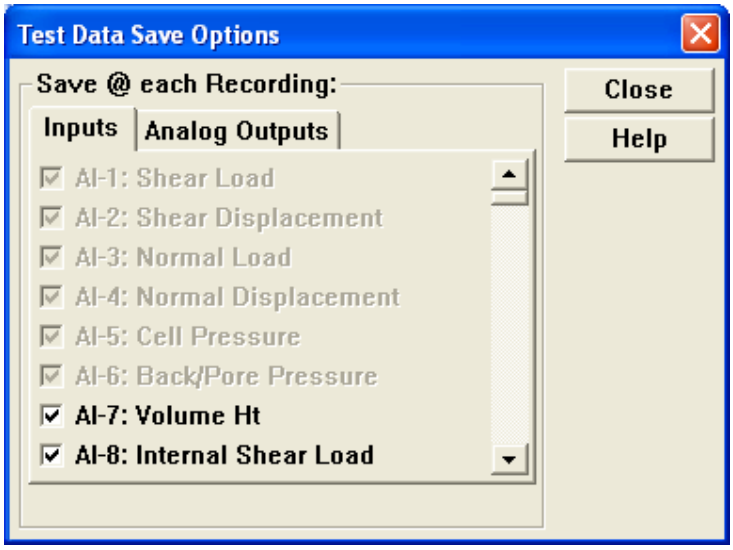


Figure G.25: The 'Test Data Save Options' dialog box.

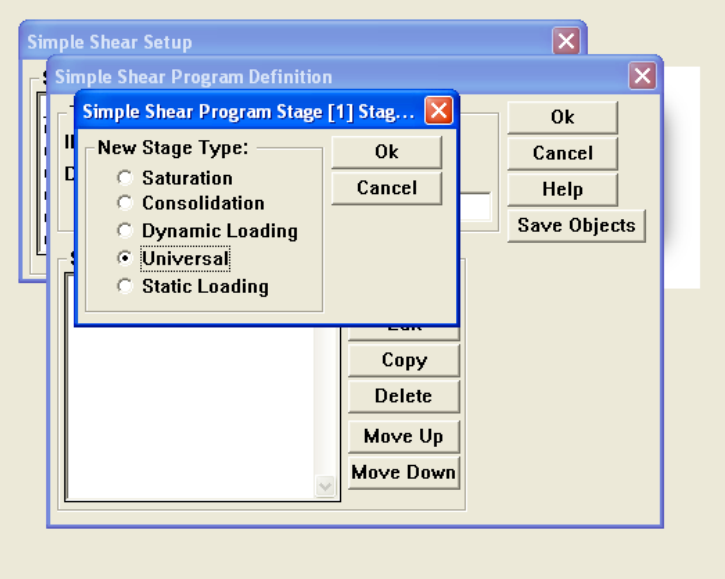


Figure G.26: The new stage dialog box.

> I like to 'Update [the] Specimen Dimensions' at the beginning of this stage, but you should make sure that it is taking care of your strains as you wish. Click 'Ok' when you are done.

6. Give the new universal program an 'ID' and 'Description' (Figure G.28), and create a new phase.
7. **Ramp Phase** In this phase dialog box, we will create the ramp phase. In the Duration tab (Figure G.29), set an appropriate duration. In this case, it depends on how long we take to ramp to the consolidation pressure.
8. In the 'Data Acquisition' tab (Figure G.30), select the type of data acquisition that you would like. I like to have timed increments. Pick an increment that will give you a good idea of the loading function without creating too big of a data file. You shouldn't need many data points in this phase.
9. Still in the Ramp phase dialog box, double-click on the 'A0-2: Normal Actuator - Not Defined' line at the bottom of the dialog. This opens another dialog box (Figure G.31); make the following changes:
   - Select 'Sn - Normal Stress' from the feedback drop-down list.
   - Select the 'Ramp' waveform.
   - Change the 'Ramp Control Parameters' so that the normal stress ramps up to some value at a rate (or in the time) that you want.

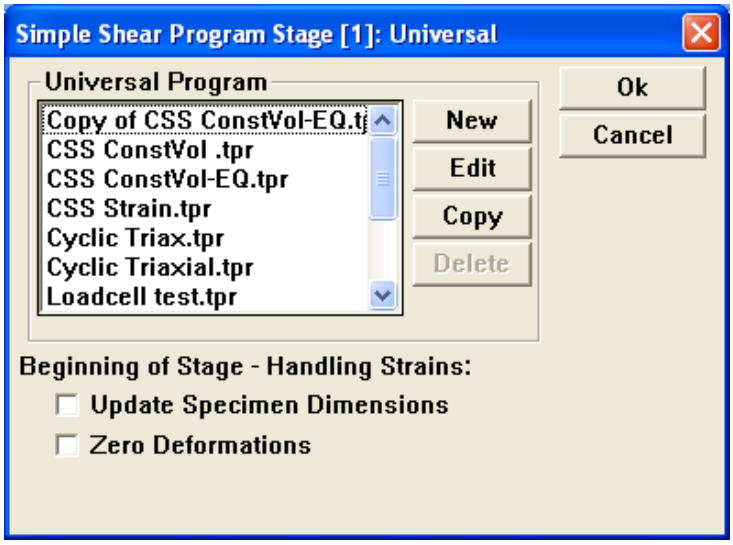


Figure G.27: The universal program selector dialog box.

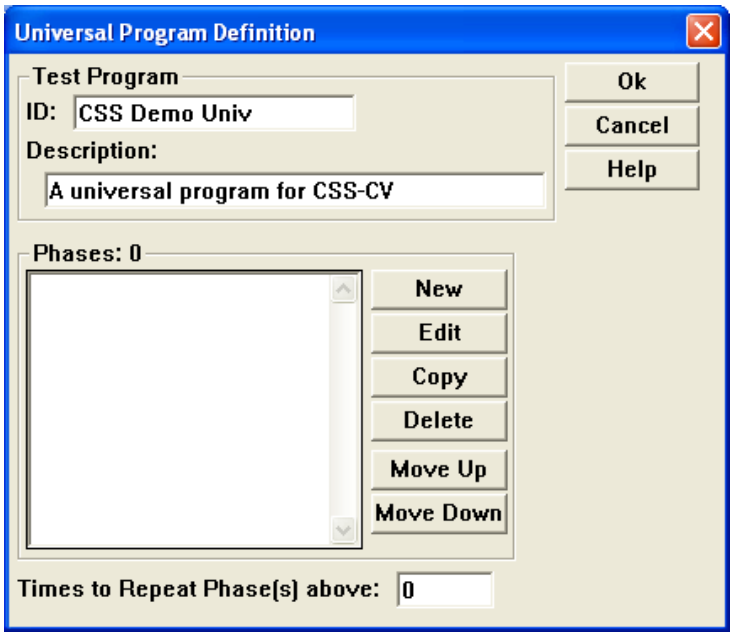


Figure G.28: The universal programs dialog box.

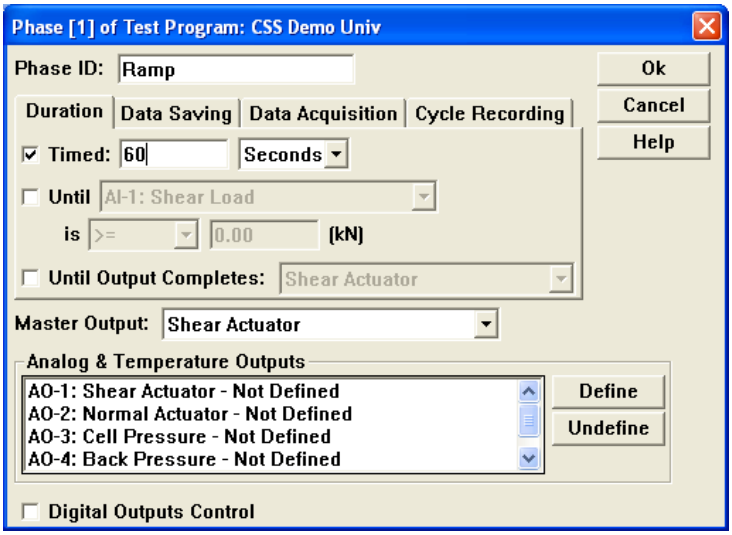


Figure G.29: The duration tab of the Ramp phase dialog box.

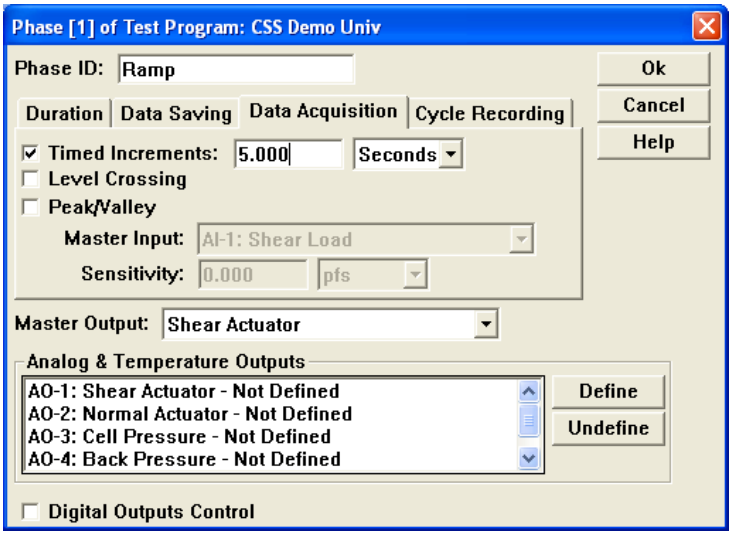


Figure G.30: The data acquisition tab of the Ramp phase dialog box.

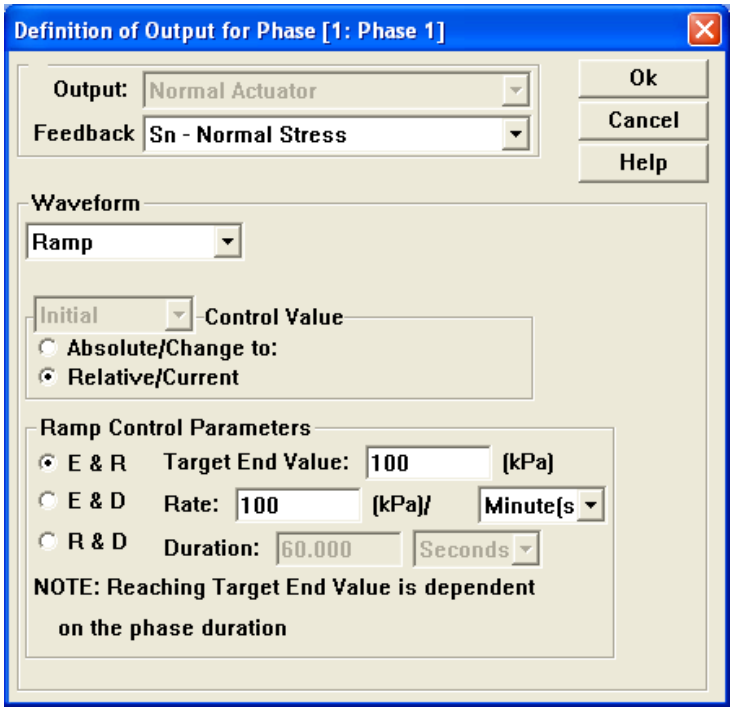


Figure G.31: The normal actuator definition dialog box for the Ramp phase.

Generally, I use the 'E & R' (*e*nd value and *r*ate), but you could make it work with any of those options. 'D' is for duration.

- Click 'Ok' when you are done.

10. Back in the Ramp phase dialog box, double click on the 'A0-1: Shear Actuator - Not Defined' line at the bottom. This opens another dialog box (Figure G.32); make the following changes:

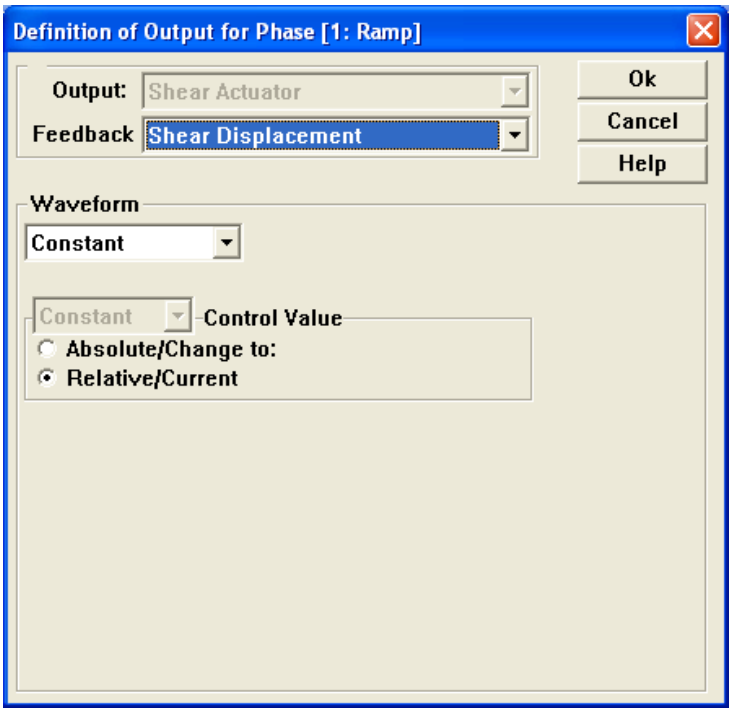


Figure G.32: The shear actuator definition dialog box for the Ramp phase.

- Select 'Shear Displacement' from the feedback drop-down list. This will hold the shear displacement to be the same as when the phase started.

  Alternately, you could select 'Shear Stress' from the feedback and have it set at a constant value of zero. The benefit of this is that this input is ready to go for the cyclic phase, and, thus, is less likely to do something weird when the cyclic phase starts. On the other hand, your shear actuator would be more likely to misbehave during the ramp and consolidation phases. But your mileage may vary.

- When you are finished here, click 'Ok.'

11. Now we are back again in the Ramp phase dialog box. We don't need to define the Back or Cell Pressure inputs, so if everything looks ok, click 'Ok.'

12. Returning to the Universal Program Definition dialog, click 'New' to create the consolidation phase.

13. **Consolidation Phase** In the 'Duration' tab of this dialog box (Figure G.33), set the phase duration to a time that is well above that needed for consolidation. We manually proceed from this phase, so we don't have to worry about getting the duration perfectly.

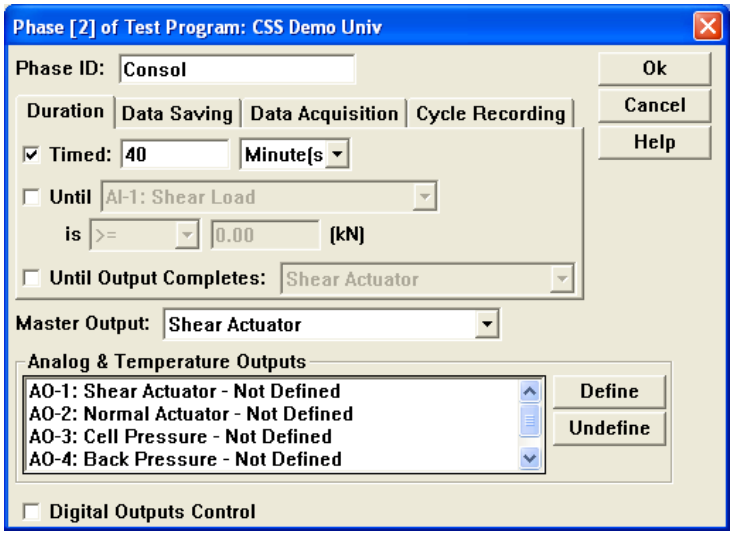


Figure G.33: The dialog box for the Consolidation phase.

14. In the 'Data Acquisition' tab, choose a timed scheme that won't collect too much data. Every 5 seconds might be enough.

15. Double click on the 'A0-1 Shear Actuator - Not Defined' line to set the controls for the shear actuator (Figure G.34).

    - Choose the feedback to be the same as the previous phase. In our case, it should be 'Shear Displacement.'
    - Set the waveform to 'Constant' with 'Relative/Current' as control value.
    - Click 'Ok' when done.

16. Double click on the 'A0-2 Normal Actuator - Not Defined' line to set the output for the normal actuator (Figure G.35).

    - Set the feedback to 'Sn - Normal Stress.'

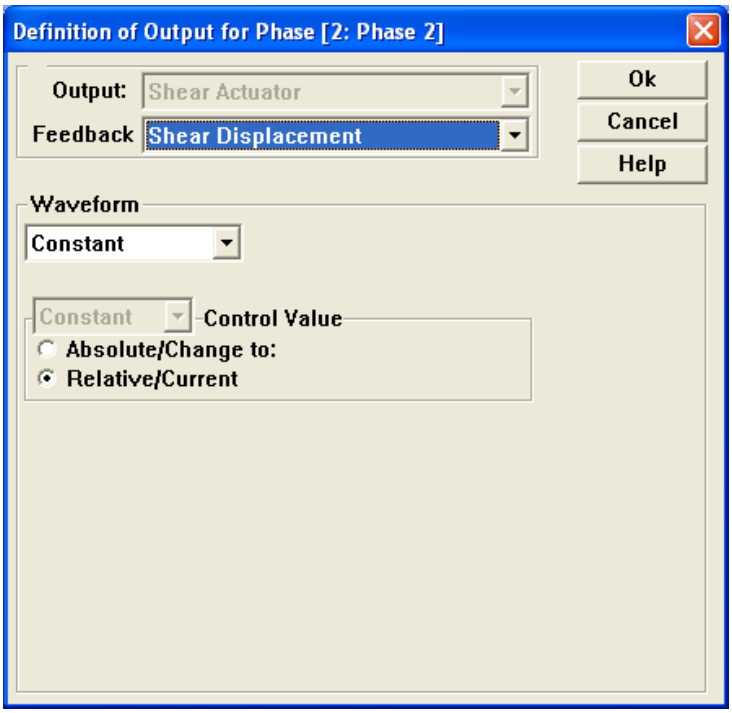


Figure G.34: The shear actuator definition dialog box for the Consolidation phase.

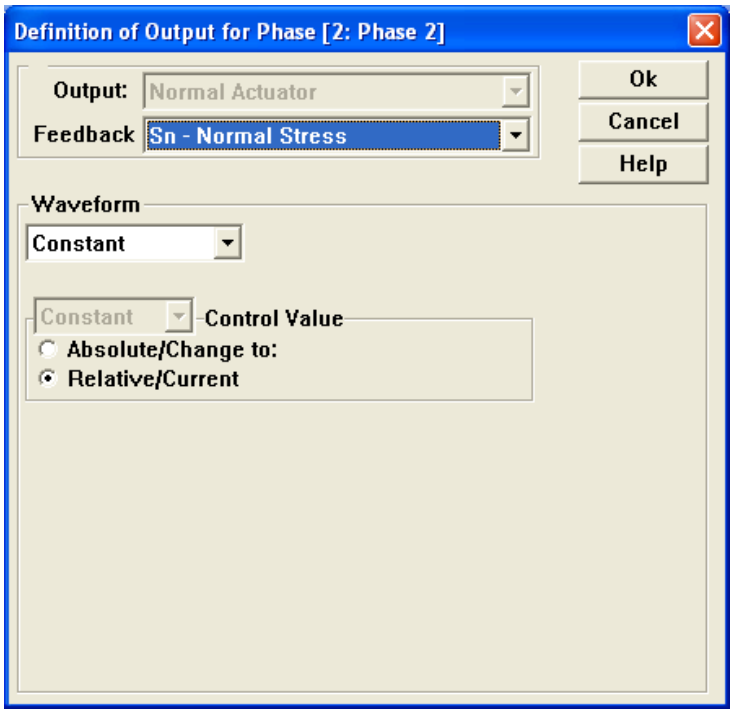


Figure G.35: The normal actuator definition dialog box for the Consolidation phase.

- Set the waveform to 'Constant' and the control value to 'Relative/Current.'
- Click 'Ok' when finished.

17. Back in the Consolidation phase definition dialog, double-check everything and click 'Ok' when finished.
18. Now back to the Universal Program Definition dialog (Figure G.26), you should see the ramp and consolidation phases. Click 'New' to create the third and final phase, the cyclic phase.
19. **Cyclic Phase** In the Duration tab of the Cyclic phase dialog (Figure G.36), set an appropriate duration. Since we want to run the test until failure, I do NOT set a timed duration. I like my tests to stop when a failing strain occurs or when I am about to reach the limits of the machine. Make sure to set this as needed. In this case, the internal shear LVDT has a stroke of ±1.25 mm. I set the duration of the test as 'Until A1-9 Internal Shear LVDT >= 1.2mm.'

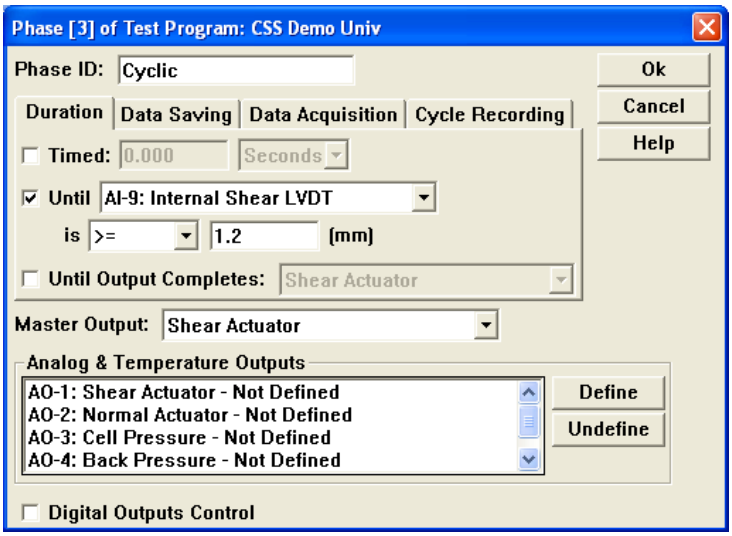


Figure G.36: The duration tab for the Cyclic phase.

20. In the Data Acquisition tab of this dialog, set the acquisition to something appropriate to your testing. I prefer to use only timed increments, making sure to get several data points for each cycle. Choose what is best for your testing conditions.
21. Double click on the 'A0-1: Shear Actuator - Not Defined' line to set the controls of the shear actuator during the cyclic phase (Figure G.37).

- Set the feedback to 'T - Shear Stress.'
- Set the waveform to 'Sine.' Change the delay and phase shift, if desired.

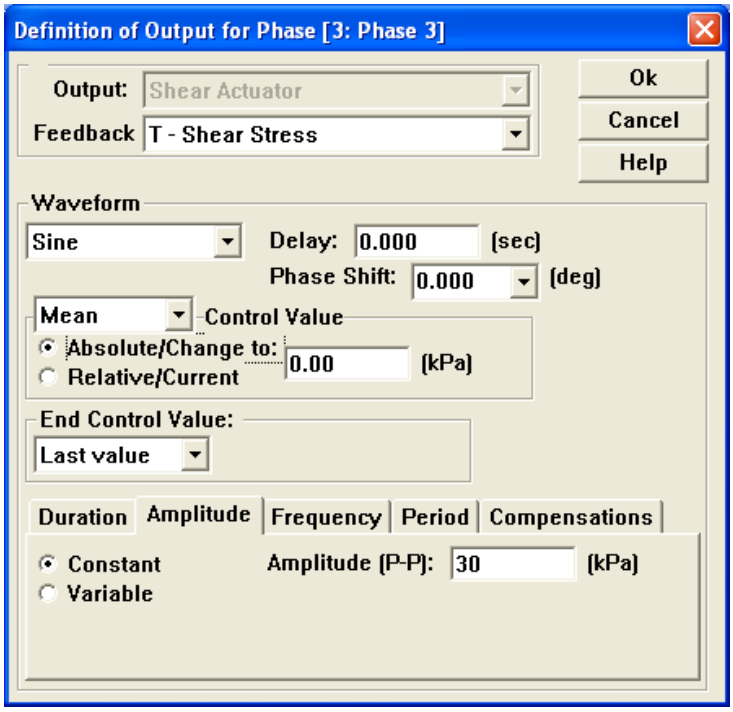


Figure G.37: The shear actuator definition dialog box for the Cyclic phase.

- Make sure it is set to a 'Mean' control value of 0. (If you are performing a test with some static shear stress value, you would have to change this setting to something other than 0 kPa.)
- On the 'Amplitude' tab at the bottom, type in the appropriate value. For a simple shear test, *Amplitude (P-P)* $= 2 \cdot CSR \cdot \sigma'_v$. The factor of 2 is to change from single-amplitude to double-amplitude (P-P). In this equation, σ'_v is your normal consolidation stress.
- On the 'Frequency' tab, input the frequency of the sinusoidal loading. I normally choose 0.25 Hz, but it seems like a lot of literature cite a frequency of 0.1 Hz (e.g. Wu et al. (2004)). (Alternately, you can adjust the period instead of the frequency.)
- Click 'Ok' when you are finished.

22. Now double-click on the 'A0-2 Normal Actuator - Not Defined' line. In the new dialog box:
    - Set the feedback to 'Normal Displacement.' (It's a *constant-volume* test.)
    - Set the waveform to 'Constant' and the control value set to 'Relative/Current.'
    - Click 'Ok' when finished.
23. When finished with the Cyclic phase definition, click 'Ok.' Continue to click 'Ok' in all the open test definition windows. You are finished creating a new test.

Arbitrary (Earthquake) Loading

An earthquake loading test is created in the same way as a normal sinusoidal test, except instead of choosing a sine waveform in the cyclic phase definition, 'User Defined' should be chosen as the waveform. Once 'User Defined' is chosen as the waveform, an input file can be chosen. You have to create the input file yourself. Section UNI 3.6.2.8 of the CATS Advanced and Universal Manual (v.1.96) contains details about how to use the 'User Defined' waveform.

The code snippet below shows how I have created the input file using Python. Since this is only a snippet, it is important to know that `name` is the name of the entry in `repofile` where I have stored several stress and strain time histories. The variable `tm0` contains the stress time history in this case.

```
        # extract the data node from the aggregate file
        data = repofile.getNode('/TimeHistoriesProfileN05/{}'.format(name))
        # extract the variables I need from data
        profile = data.col('profile')[0]
        depth = data.col('depth')[0]
        tm0 = data.col('taugam')[0][:,motind] # This is the stress time history
        dt = data.col('dt')[0]
        maxval0 = np.max(np.abs(tm0))

        # Normalize the time histories so that they all have max values of 1.
        tm = tm0 / maxval0

        # Limit: 4096 data points for each input file
        # See CATS Advanced and Universal Manual, pg 273
        if len(tm) > 4096:
            if (len(tm) % 4096) > 0:
                no_files = len(tm0) // 4096 + 1
        else:
            no_files = 1

        filepath = './Figures/ProfileN05/GCTSfiles/{}'.format(name)
        f2 = open(filepath + '-info.txt', 'w')

        # Create a file for each 4096 data points in my time history
        for i in xrange(no_files):
            f = open(filepath + '-{}_{}.txt'.format(i+1,no_files), 'w')

            # First line in the file is the number of data points
            if no_files > 1:
                if i < no_files-1:
                    f.write('{}\r\n'.format(4096))
                else:
                    f.write('{}\r\n'.format(len(tm) % 4096))
            else:
                f.write('{}\r\n'.format(len(tm)))

            temptm = tm[i * 4096:(i+1)*4096] # Create a temporary time hist var
            maxval = np.max(temptm)
            minval = np.min(temptm)
```

```
            meanval = (maxval + minval) / 2.
            valrange = maxval - minval
            Peak2Peak = valrange

            # Now write out the temp time history for this file.
            for j, val in enumerate(temptm):
                f.write('{}\r\n'.format(val))

            f.close()

            # To help myself with the data needed for the GCTS software, I also create an
                 additional file:
            if i == 0:
                f2.write('{}\r\n'.format(name) +
                'This is a {} time history in units of kPa '.format(motiontype) +
                ' for cyclic simple shear tests. \r\n' +
                'The motion is from a depth of {:.2f} m'.format(depth) +
                ' from {}.\r\n\r\n'.format(profile) +
                'No. of files: {}\r\n'.format(no_files) +
                'To get original amplitudes, multiply by {}.\r\n \r\n\r\n'.format(
                                                maxval0))
            else:
                f2.write(
                  'End Control Value: Given value: {}'.format(
                                            temptm[0] / 0.65) +
                ' * CSR * sig\'v (in kPa)\r\n\r\n\r\n')

            f2.write(
                'For file # {}:\r\n'.format(i+1) +
                'Mean Control Value: {} * NewScaleFactor '.format(meanval) +
                ' = {} * CSR * sig\'v \r\n'.format(meanval
                                                / (0.65)) +
                'Amplitude (P-P):  {} * NewScaleFactor '.format(Peak2Peak) +
                ' = {} * CSR * sig\'v \r\n'.format(Peak2Peak / 0.65) +
                '                  /  CSR * sig\'v  \ \r\n' +
                'NewScaleFactor = < -------------- >\r\n'+
                '                  \      0.65     / \r\n'+
                'Duration: {} points\r\n'.format(len(temptm)) +
                'Time Step: {:.4f}\r\n'.format(dt))
        f2.write('End Control Value: Last Value \r\n')
        f2.close()
```

G.4.2 Friction Baseline Tests

The GCTS equipment is not frictionless. This is especially apparent for dynamic/cyclic testing. To get a handle on the amount of extra load due to friction that the load sensors pick up (and the sample doesn't feel), a friction baseline test may be useful.

A friction baseline test is performed by running a strain-controlled test on an air- or water-filled sample. You may think of a better way to do it, but one way is to create a test in the GCTS software with a sinusoidal strain-controlled loading function at the same frequency as your normal testing regime. I begin the loading at very small strains but increase the

amplitude during loading so that the final cycles of the baseline test have as large of strains as I would expect from my soil test.

The results of this baseline test can be used to subtract friction from a regular soil test. The extra load due to friction is not always insignificant. For simple shear tests, I have found the friction to be a function of the displacement and load history (hysteresis loops are formed at large displacements), but I invite you to look closely at your own data. Figure G.38 shows some results from one baseline test on a sample of air.

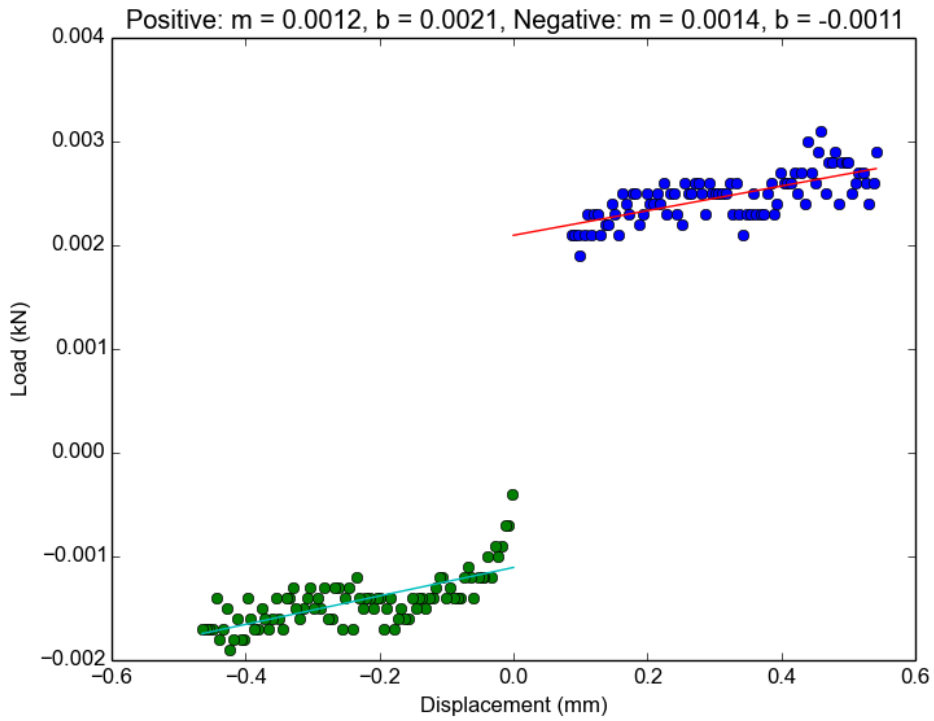


Figure G.38: Results of an air friction baseline test. The relationship between friction load and displacement can be approximated with a bi-linear fit. The points on this plot are each from the peaks of the sinusoidal loading.

G.4.3 Sensor Calibration

The calibration of the sensors should be checked periodically. Eventually, the sensors will need to be calibrated. To do so:

1. Select *Analog* from the *System* → *Inputs* menu.
2. In the 'Analog Inputs' window (Figure G.39), choose the appropriate sensor.

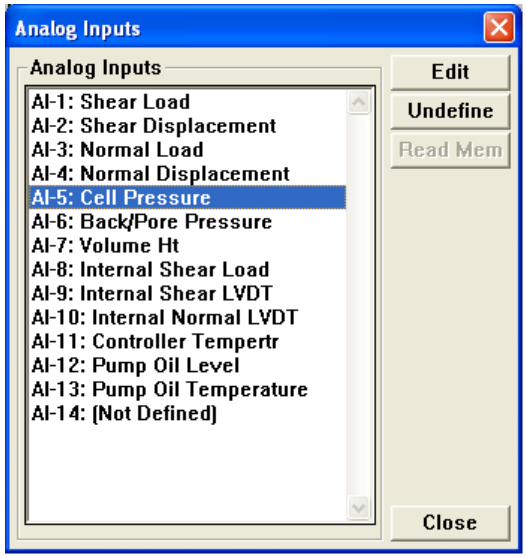


Figure G.39: The 'Analog Inputs' dialog.

3. In the 'Editing Analog Inputs...' window (Figure G.40), double-check that the 'Attached Sensor' is correct. If so, click on 'Calibrate.'
4. You are given three choices on how to calibrate your sensor (Figure G.41). I generally use either the two-point method (Figure G.42) repeated a few times, or the multi-point method (Figure G.43).
5. Whatever method of calibration, apply the forcing or loadings to the sensor, as needed, and put the known values of these forcing into the dialog window.

Load Cells

For the calibration of load cells, I suggest using one of two methods to apply variable loads: free weights or a calibration ring.

If you are using free weights, contact the lab supervisor to get a frame onto which you can apply the weights. These look like a triaxial cell without the acrylic plastic shell. You place the load cell of interest between the bottom of the frame and a vertical piston arm. This piston arm leads through the top cap of the frame to a plate onto which you can place consolidation weights. You can mass the piston arm and consolidation plates before beginning. This is how you obtain your known loading.

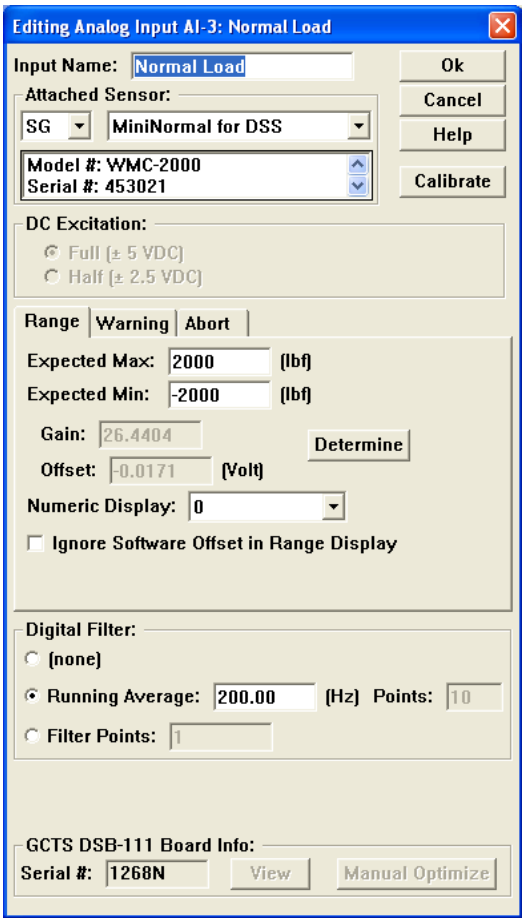


Figure G.40: The 'Editing Analog Inputs . . . ' window for the normal load cell.

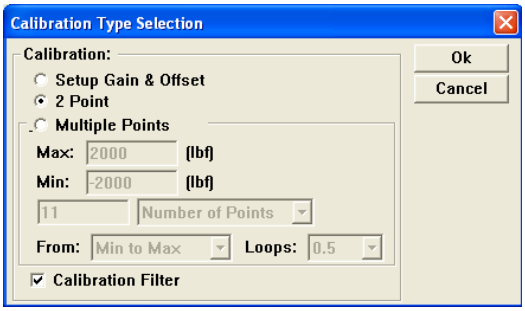


Figure G.41: Dialog box for choosing the calibration method.

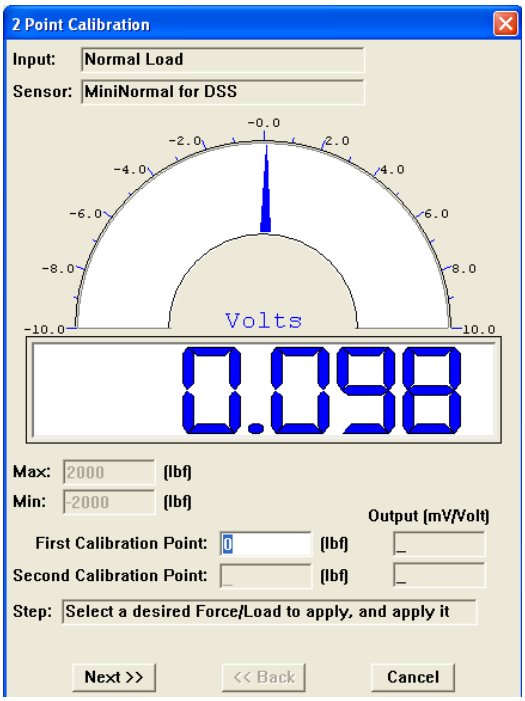


Figure G.42: The 2 point calibration method dialog.

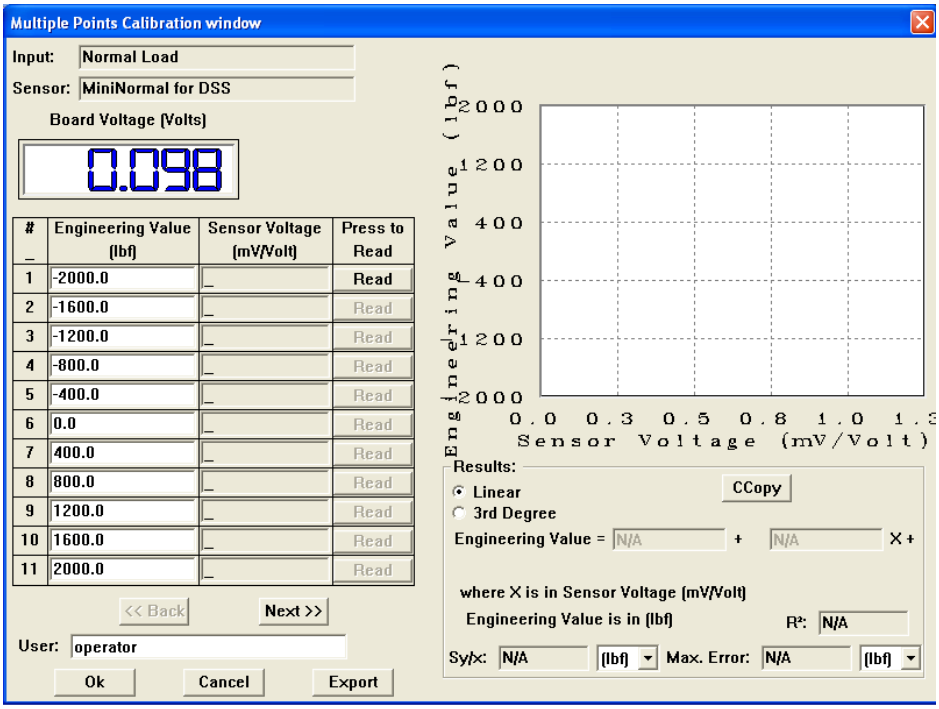


Figure G.43: The multiple point calibration method dialog.

There is a calibration ring in the lab, somewhere. If you use it, put the ring in the place of the triaxial sample with the normal piston attached to the normal actuator and the load frame tilted forward. Use the GCTS' normal actuator to apply incremental loadings to the ring. Measure the deflection of the ring and calculate the applied load from the deflection.

If you choose to use the calibration ring, you may need to calibrate the ring. At the very least, make sure it performs linearly. If it does, you can simple zero out your sensor when you are done with the calibration. It should be noted that the calibration ring will probably not work well for the shear load cells unless you mount them on the normal actuator.

Pressure Sensors

For calibration of the pressure sensors, find a pressure gauge that you trust. The pressure panel gauge has heretofore been accurate, so it might be a good place to start. If you don't trust it, one of the lab supervisors can help you find a better gauge.

G.4.4 Pore Pressure Servo Calibration

The back pore pressure servo may need to be calibrated, especially if you intend to perform automatic back pressure saturation. (Obviously this doesn't apply to constant-volume simple shear tests. For information about back pressure saturation, see the companion manual on triaxial testing.) To do so, consult the document `(PCP-200) Back Pressure E-P Valve setup procedure.pdf`. It was written by GCTS and should be with the other digital documents. To summarize:

1. Close the back pressure lines so that there is no flow out of the system and the back pressure sensors can get a pressure reading.
2. Open the 'Outputs Function' window. Change the 'Output' to 'Back Pressure' and the 'Feedback' to 'None - Open Loop.'
3. Change the Set-Point value to 5 pfs (percent full scale) and turn on the control.
4. Adjust the 'ZERO' screw on the back pressure servo (also known as the E-P valve; one of the black boxes mounted on the top back of the pressure panel) until the pore pressure sensor reads close to 50 kPa.
5. Turn off the back pressure control. Change the Set-Point value to 50 pfs. Turn the control back on.
6. Adjust the 'SPAN' screw on the back pressure servo until the pore pressure sensor reads close to 500 kPa. (House pressure is $\sim$ 1000 kPa. Fifty percent full scale then is $50\%/100\% * 1000kPa = 500kPa$.)
7. Turn off the back pressure control.
8. Repeat steps 3 to 7 until convergence.

G.4.5 Loading & Saving Configuration Files

In the GCTS software, configuration files save information about the current configuration of the system. This data includes the default height and diameter of the samples, the calibration of the sensors, which load cells are being used, and the PID values of of each control-feedback loop. When switching between simple shear and triaxial tests, it is more convenient to load the required configuration file than to make all the changes by hand. Additionally, it is helpful to be able to go back to a known working configuration.

After making changes to the system configuration and every few months, it is a good idea to create a new configuration file. This is done by going to the *System* → *Configuration* menu and selecting *Save As*. Likewise, to load a previously-saved configuration, go to the *System* → *Configuration* menu and select *Load*. Finally, you can export the configuration to a text file by selecting *Export* from that same menu.

Bibliography

Skempton, A. W. (1954), "The pore-pressure coefficients A and B." *Geotechnique*, 4(4), 143–147.

Schnabel, P., Seed, H. B., and Lysmer, J. (1972), "Modification of seismograph records for effects of local soil conditions." *Bulletin of the Seismological Society of America*, 62(6), 1649—-1664, URL http://www.bssaonline.org/content/62/6/1649.short.

Vaid, Y. P. and Negussey, D. (1984), "Relative density of pluviated sand samples." *Soils and Foundations*, 24(2), 101–105.

Vaid, Y. P. and Negussey, D. (1988), "Preparation of reconstituted sand specimens." *Advanced triaxial testing of soil and rock, ASTM STP*, 977, 405–417.

Wu, J., Kammerer, A. M., Riemer, M. F., Seed, R. B., and Pestana, J. M. (2004), "Laboratory study of liquefaction triggering criteria." In "13th World Conference on Earthquake Engineering, Vancouver, BC, Canada, Paper," 1–1.

Bradshaw, A. S. and Baxter, C. D. P. (2007), "Sample preparation of silts for liquefaction testing." *Geotechnical Testing Journal*, 30(4), 324.

Cubrinovski, M., Bradley, B., Wotherspoon, L., Green, R., Bray, J., Wood, C., Pender, M., Allen, J., Bradshaw, A., and Rix, G. (2011), "Geotechnical aspects of the 22 february 2011 christchurch earthquake." *Bulletin of the New Zealand Society of Earthquake Engineering*, 44(4), 205–226, URL http://ir.canterbury.ac.nz/handle/10092/6393.

Wang, S., Luna, R., and Stephenson, R. W. (2011), "A slurry consolidation approach to reconstitute low-plasticity silt specimens for laboratory triaxial testing." *Geotechnical Testing Journal*, 34(4), 1.

Appendix H

Cyclic Triaxial Testing Manual

H.1 Introduction

This manual is mean as a guide to aid the new student/researcher in the use of the GCTS equipment for cyclic triaxial testing. A separate manual should be available for cyclic simple shear testing. The treatment of the subject is by no means exhaustive; pertinent external literature should be reviewed as well as the manuals provided by GCTS.

A general overview of the equipment and setup are given. Additionally, best practices for sample preparation and equipment operation are discussed. In this manual, wet pluviation is presented as the method of sample preparation. However, with slight changes, other preparation methods can be easily implemented. Finally, discussion of data reduction is presented. The GCTS equipment has very high potential for academic research; however, it has its share of quirks. The over-arching purpose of this manual is to help the reader avoid the issues that prevent a successful test in the hope that proficiency can be gained in less time than that of the author.

H.2 Setup and Equipment

H.2.1 Physical Setup

The GCTS equipment is located in Lab 15 of the W.C. English Geotechnical Lab at 260 Inventive Lane (previously 104 Plantation Road). Facing the equipment, the components of the equipment are, from left to right:

- **Hydraulic Pump**- This gray-colored tank with valves, pipes, and gauges provides the force to the hydraulic-operated normal and shear actuators. The oil level,

temperature, and condition should be periodically checked.

- **Load Frame and Pressure Cell Components**- This is where samples are prepared and tested.
- **PCP-200 Pressure Control Panel** or *Pressure Panel*- This panel contains a vacuum and a pressure gauge, vacuum and pressure controls, burettes, valves, and switches for the Top Back Pressure lines, Botttom Back Pressure lines, and the Cell Pressure lines.
- **APC UPS**- This black box provides power in case of a power outage. During a test it should give the user about 10 minutes to shutdown the controller and PC, preventing data loss. Note: the hydraulic pump is *not* connected to the UPS, so if the power goes out, your test it over anyway. The UPS should also prevent power surges, etc. Documentation (registration info, manual, etc) for this device should be found on one of the shelves with other documents.
- **SCON-200 Digital System Controller** or *SCON* or *controller*- This blue cube controls the entire system and collects all data. The firmware of the controller was updated to version 1.96 in the summer of 2013.
- **ULT-100 Ultrasonic Interface**- This gray box on top of the the SCON controls and receives information from the bender-element platens.
- **Lab PC**- This PC and associated software provide an interface to the SCON controller, and, thus, the rest of the equipment.
- **Welch Duoseal Vacuum Pump 1402**- Located on the counter behind the SCON, this vacuum pump, with the Nold DeAerator, deaerates water for use in saturated samples. The oil in the vacuum pump should be changed about once a year (which never really happens) or anytime contaminants can be seen in the oil. A manual for this particular pump can be found online without too much effort. Effort should be made to prevent the introduction of any contaminants, even water. A valve on the back of the pump allows for 'gas ballast.' Operating the pump with the gas ballast (with the valve open) prevents the condensation of water vapor within the pump at the slight sacrifice of vacuum pressure (see the manual for more information).
- **Geokon Nold DeAerator Model 2100**- On the wall above the vacuum pump, the deaerator removes the dissolved air from water to aid in saturation of samples. Only distilled water should be used in the deaerator. To prevent water from being pulled into the vacuum pump, the deaerator tank should be filled only to three-fourths of the tank capacity. Water is deaerated by running the impeller while a vacuum is applied. Ten or 15 minutes should remove most of the air; see the manual for more information.

Figure H.1 shows some of these components and their approximate layout in the lab.

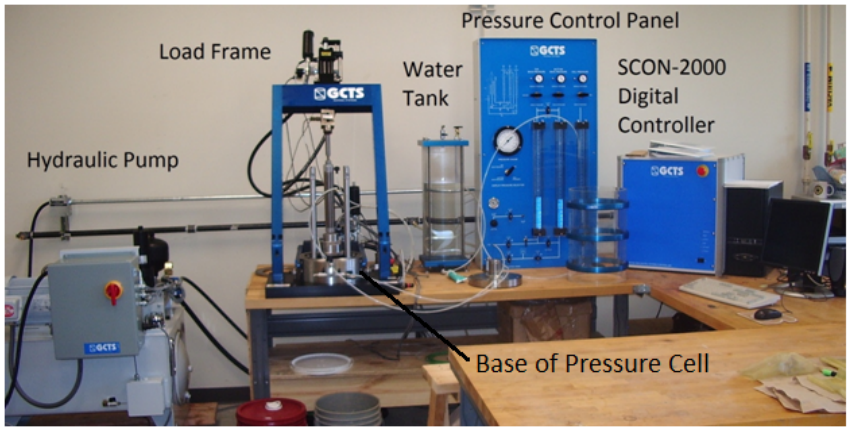


Figure H.1: The physical layout of some of the GCTS equipment.

H.2.2 The Cell

The cell is composed of many parts that fit together to allow the specimen to be pressurized (i.e. cell pressures greater than ambient air pressure). These parts include:

Base The base of the cell is short, stainless steel cylinder mounted below the normal load frame (Figure H.2). On the front are valves and poke-throughs for the top back pressure lines, bottom back pressure lines, and the cell drain line. To the sides are poke-through ports for internal sensors. Attached at the rear is the ball bushing guide for the shear actuator piston.

Columns The columns support the cell top cap. See Figure H.3.

Cell Wall This acrylic plastic shell fits over the columns and top cap and onto the base of the cell.

Top Cap The top cap mounts onto the top of the columns and contains a ball bushing guide for the normal piston and poke-through ports for internal sensors.

Top Ring The top ring seals the cell assembly.

If filled with water, the maximum cell pressure is 1000 kPa. If the cell is air-filled, the maximum pressure is 500 kPa.

H.2.3 The Pressure Panel

The pressure panel contains valves and regulators to used to apply pressures (including vacuums) to the sample specimens. See Figure H.4. These pressures can be piped via the cell lines, the bottom back pressure lines, and the top back pressure lines. At the top

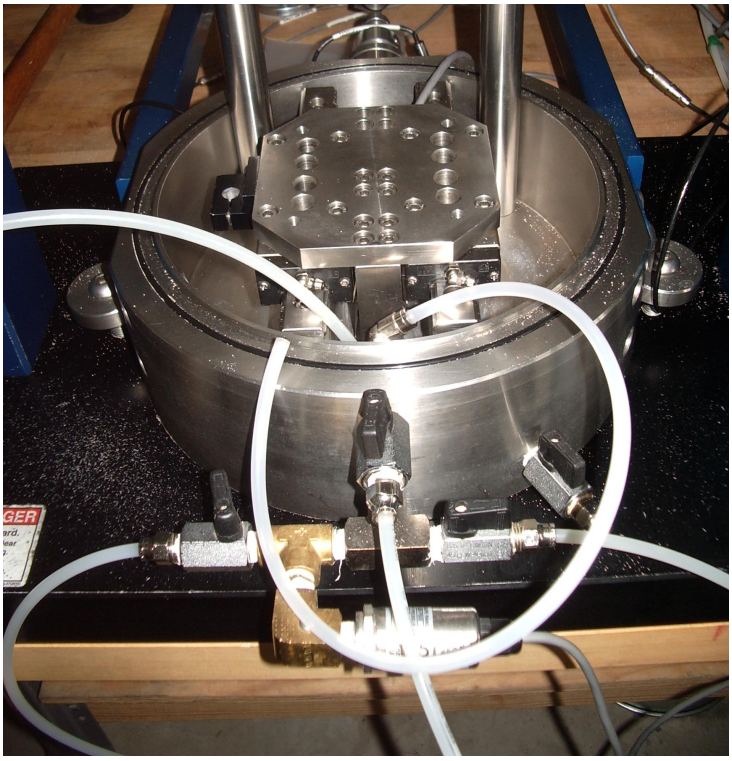


Figure H.2: The base of the cell with the shear carriage installed and two posts removed. Notice the back pressure sensor mounted at the front.

Figure H.3: Other parts of the cell. Clockwise from bottom left: the top cap, two of the four columns, the cell wall, and the top ring.

of the panel are the manual pressure regulators. Below these are the selector switches so that each line can be individually set to manual pressure control, vacuum control, vent, or servo pressure control. Below the selector switches are the burettes that show the level of water in each column, and below the burettes are valves for each line. To the left side of the panel is the vacuum pressure regulator, the vacuum port, the dial gauge, and the dial gauge selector switch. Note that the back pore pressure sensor has been moved from the back of the pressure panel to the front of the base of the cell.

H.2.4 Other Parts

A few other items are needed for a successful triaxial test. Figure H.5 shows some of them. They include:

- A soil sample
- A flask with stopper and spout
- A mold
- O-rings
- A latex membrane
- Top and bottom platens
- Pedestals to situate the specimen at the appropriate height.
- Filter paper
- A pi tape and a ruler
- The normal piston
- The mating collar and load-bearing piece for the normal actuator
- A mold extender.

H.2.5 The GCTS Software

The GCTS software on the lab PC provides the gateway to the GCTS system. On the PC, two GCTS programs have been installed, the CATS program and the ultrasonics program. The ultrasonics program is used in conjunction with the bender elements to measure the shear wave velocity of a specimen. This same functionality can be found in the CATS program, so the ultrasonics program will not be discussed here.

The GCTS CATS software is started by double-clicking the 'CATS(2)' shortcut on the desktop of the PC, or by finding the 'CATS' entry in the start menu. Before starting the software, the SCON controller (and the ULT-100 controller if you will use it) should be

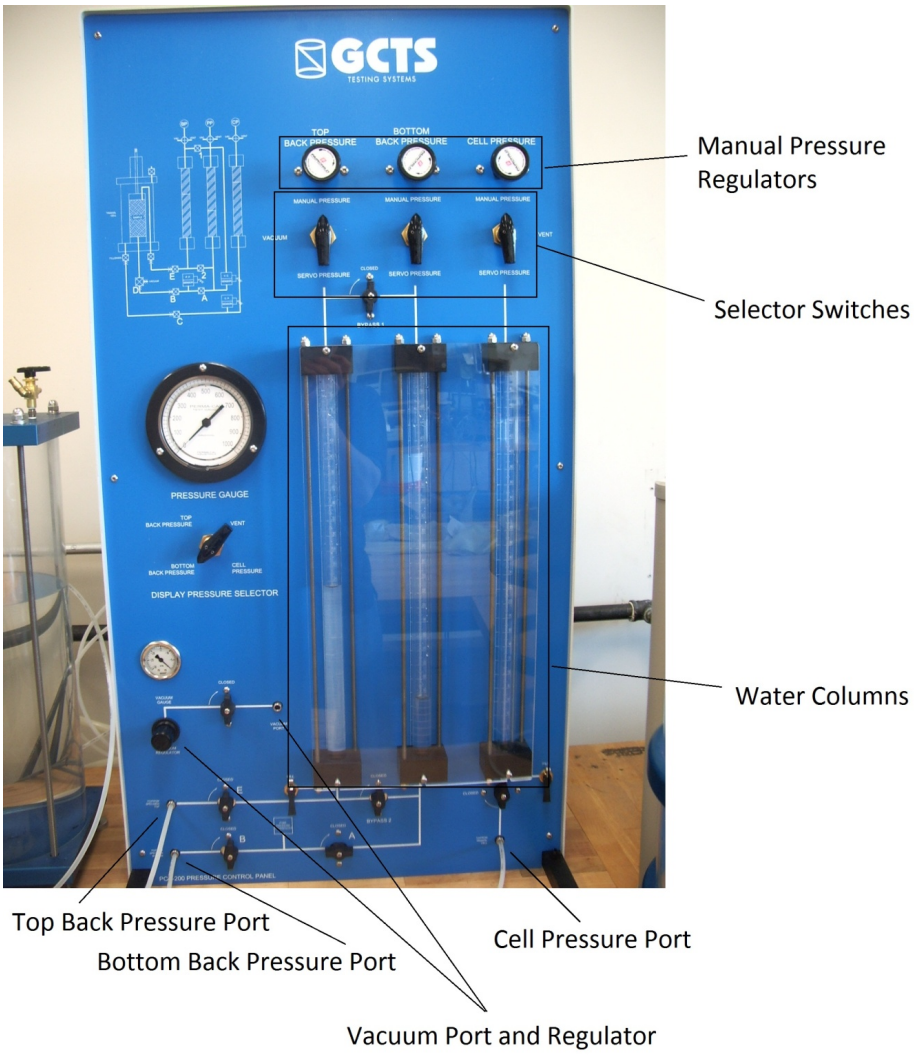


Figure H.4: The pressure panel.

Figure H.5: Assorted other parts used in a triaxial test. Clockwise from the left: sand-filled flask with spout, 2.8-inch triaxial mold, o-rings, black triaxial pedestal, additional pedestal for 2.8-inch specimens, the top and bottom platens, a latex membrane, a piece of filter paper, the normal piston, a pi tape, the normal actuator contact plate and mating collar, and a mold extender.

on. It may take a few moments for the software to connect with the SCON. Once it has done so, you will be prompted to login.

Two accounts are available in the CATS software: 'Administrator' and 'operator.' The operator account should be used for most day-to-day testing; the Administrator account is needed to change some sensitive settings. Log into either account using the password of *gcts*.

> There are some settings that cannot be changed with the regular Administrator account, e.g., changing the polarity of the internal LVDTs. To change these settings, you need to log in using the license number. To do so:
>
> 1. Log in as the Administrator.
> 2. In the *System* menu, select *Setup*.
> 3. From this system setup window, copy down the license number. It should end in `D0L0`. (Those are zeros, not o's.)
> 4. Log out from the administrator account, and log back into the administrator account using the license number as the password. Don't include the dashes and remember to capitalize the letters.
>
> Obviously, you should only log in like this when you need to, and be careful what you change.

Once the software logs in, you are left with a mostly blank window with a menubar at the top, a toolbar below the menubar, and another toolbar to the right. The right toolbar contains dials, gauges, graphs, and other readouts that will help you follow the progress of your test (see Figure H.6). Once you set up a layout that you like, you can save it under the *Views* → *Save Layouts* menu.

The top toolbar (Figure H.7) contains the tools you need to interact with the GCTS system. From left to right, these tools are:

Test Setup This tool allows you to create new 'tests' or instructions to run a test. You can add, rearrange, or edit stages of a test. See Figure H.8. Also see the 'Creating a Triaxial Test' section of this manual.

Projects The Projects tool gives you access to test results and allows you to run new tests. See Figure H.9.

Inputs Offset This tool allows you to set any sensor to any arbitrary value (in other words, zero-out your sensors). See Figure H.10.

PID The PID tool (Figure H.11) adjusts the settings of the *proportional-integral-derivative control loop feedback mechanism*. In other words, the PID settings affect the way the equipment controls itself based on the feedback it receives. The settings tell the equipment how much adjustment to make so that the actual

Figure H.6: The right toolbar in the GCTS main screen allows access to various sensor readouts.

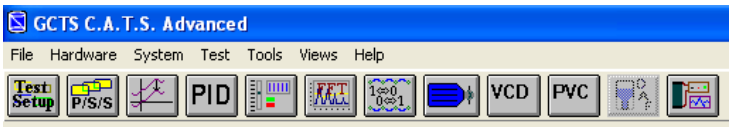


Figure H.7: The top toolbar in the GCTS main screen contains the tools needed to interact with the GCTS system.

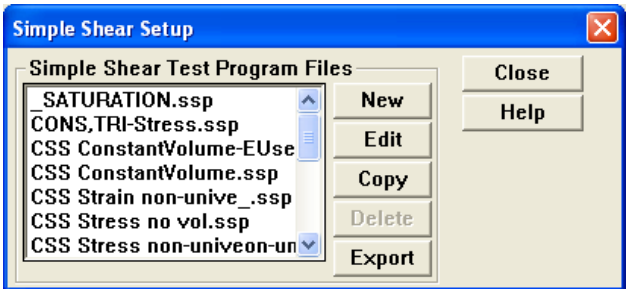


Figure H.8: The Test Setup dialog/window allows you to create and edit test instructions.

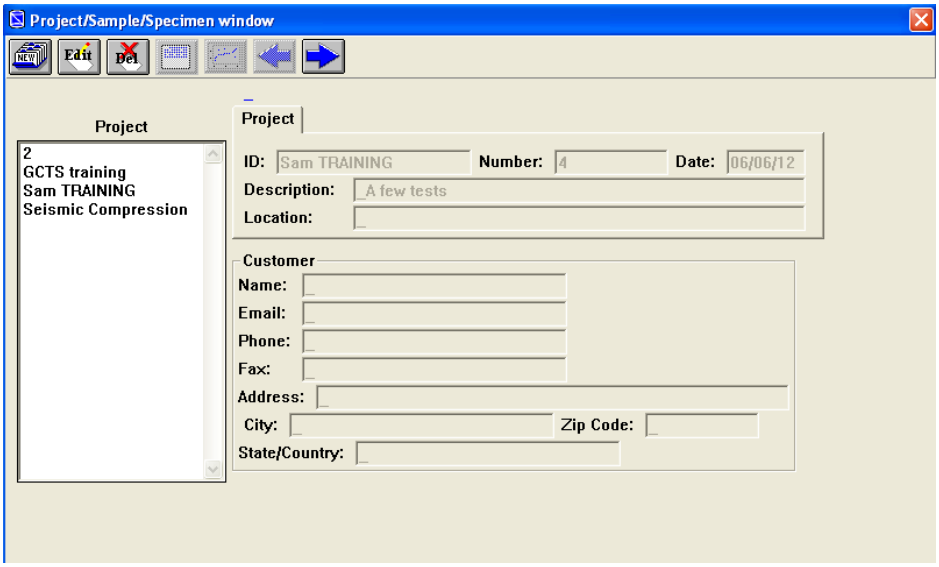


Figure H.9: The Projects dialog/window allows you to create and view test results.

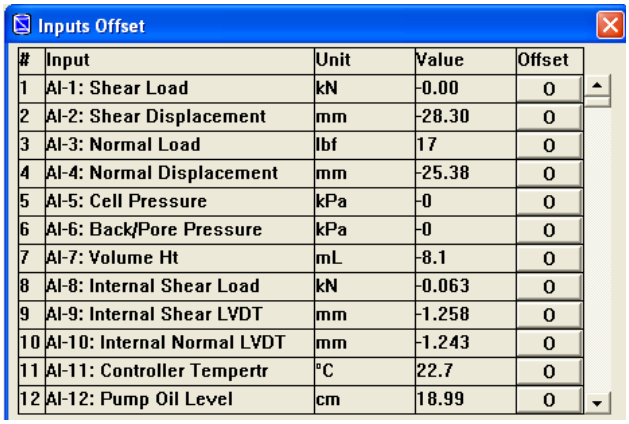


Figure H.10: The Inputs Offset dialog/window allows you to set the values of the sensors.

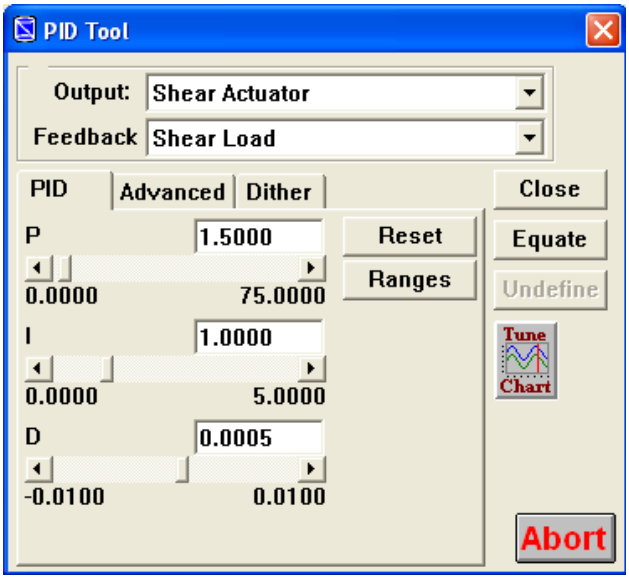


Figure H.11: The PID dialog/window allows you to set the sensitivity of the feedback mechanisms.

response matches the expected response. Wikipedia has a good article on it (search 'PID Controller'). Also see the GCTS manuals that came with the equipment.

Outputs Function This tool allows you to control the normal and shear actuators and the back and cell pressures. See Figure H.12.

Signal Analysis Chart This tool allows you to inspect a signal in time or frequency domain.

Digital Outputs The Digital Outputs tool allows you to turn the servos and digital controls on or off. It is most often used to turn on the Tilt control and turn off (or open) the automatic ball valve. See Figure H.13.

Hydraulic This tool gives you access to the hydraulic pump controls. See Figure H.14.

VCD Status Not implemented in our setup.

PVC Status Not implemented in our setup.

Flushing Status Not implemented in our setup.

Ultrasonics This tool gives you access to the ultrasonics/bender element equipment.

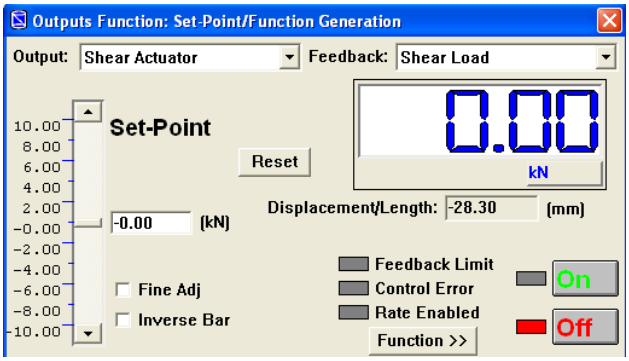


Figure H.12: The Outputs Offset dialog/window allows you to control the actuators and cell and back pressures.

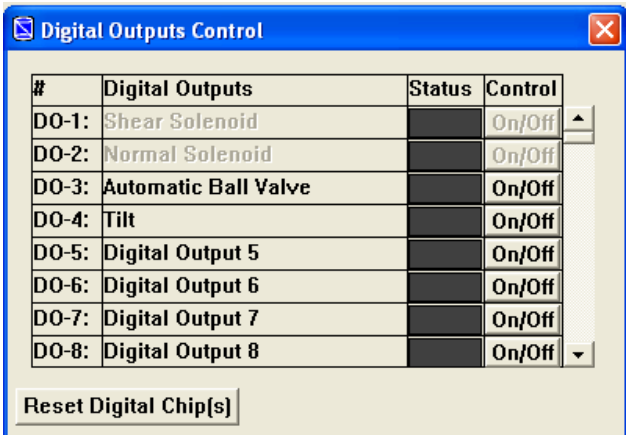


Figure H.13: The Digital Outputs dialog/window allows you to turn the servos on or off.

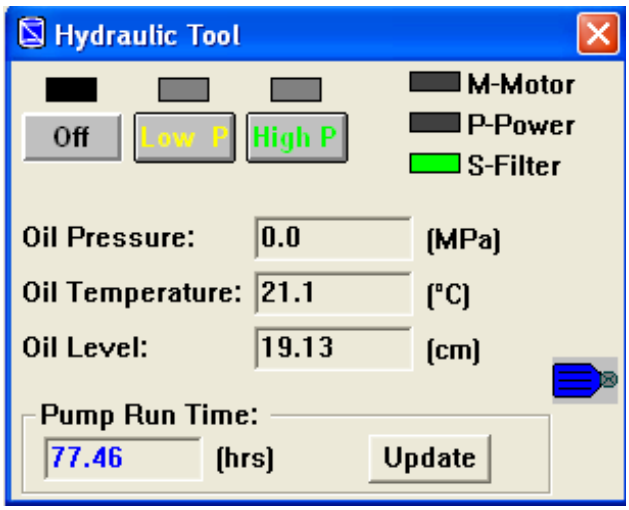


Figure H.14: The Hydraulic dialog/window allows you to turn the hydraulic pump on and off.

H.3 The Test Cycle

This section covers the steps that need to be taken to perform a test in its entirety. It includes setup, sample preparation, saturation, and cyclic loading.

H.3.1 Setup

Several preliminary steps need to be taken before preparing the sample:

1. Fill the deaerator and begin to deaerate a tankful of water. For a 2.8"-diameter sample, one tank should be enough.
   (a) On the orange-colored panel, make sure the rightmost valve ('Vent') is closed. The middle valve should be open ('To Deaerator'), and the leftmost valve should be closed (pointed to the left).
   (b) Turn on the vacuum pump. The vacuum should build fairly quickly. If not, one of your valves is open or something is broken.
   (c) Connect the intake tube with a container of distilled water. Switch the leftmost valve to 'From Bottle.' The Nold Deaerator tank should begin filling.
   (d) Switch out containers of distilled water, if needed, by closing the leftmost valve, moving the intake tube, and opening the leftmost valve again.
   (e) When the Nold tank is full, close the leftmost valve.
   (f) With the vacuum pump still on, turn on the Nold Deaerator. Allow these two to run for about 20 minutes. Consult the Nold manual for more accurate information.
   (g) After sufficient time, turn off the Nold and the vacuum pump.
   (h) Slowly switch the rightmost valve to 'Vent' to release the vacuum.
   (i) Switch the leftmost valve 'To GCTS.'

   > Remember, the Nold Deaerator should only be filled three-quarters full. Also, never turn on the deaerator without a few inches of water in the tank. This height of water is needed to lubricate the inner bearings.

2. In the 'Inputs Offset' tool, zero-out the load cell by hanging the normal piston, top platen, and two o-rings from the normal load cell.
3. Run an air or water 'friction baseline' test, if desired. See the subsection under the *Maintenace/Preparation* section.
4. Optional: Clean the top and bottom platens (or the stones alone, if they come out) in the ultrasonic cleaner.

I find this step helps with the saturation of the sample. After cleaning the platens, I keep the platens in a bowl of water flushed from the lines (next step) until I need them.

5. Thoroughly flush the top and bottom back pressure lines.

I like to open the *BYPASS 2* value on the panel and fill both back pressure burettes at once. I then empty both burettes together, twice for the top lines, and twice for the bottom lines.

H.3.2 Sample Preparation

Placement of Bottom Platen, Membrane, and Mold

1. Mount the bottom platen onto the required pedestals.

For a 2.8-inch-diameter specimen, the black cylinder pedestal and another silver-colored pedestal are needed. For a 4-inch-diameter specimen, only the black pedestal is needed.

2. Place the top cap of the cell onto the columns, and, using the normal piston and the top platen, mark the desired height of the specimen onto the normal piston. I use a rubber dummy specimen between the two platens and mark the intersection of the top platen and the normal piston with a permanent marker.
3. Place the membrane on the bottom platen and secure it with two o-rings.

I find it useful to use the mold as an o-ring stretcher; I place the o-rings on the top of the mold and thread the unattached end of the membrane through the mold until the top of the mold fits around the bottom platen. The o-rings can then be rolled off onto the bottom platen.

4. Place the un-tightened mold onto the the bottom platen, taking care not to bunch or pinch the membrane. Tighten the mold band.
5. Place a piece of filter paper in the annulus between the mold and the membrane.

Without the filter paper, the membrane is not pulled tight against the inside of the mold in all places.

6. Place the mold extender on top of the mold.

The mold extender is held together using a single o-ring. At this time, I like to have the last remaining o-ring on the top of the mold, before placing the mold extender. The mold extender was made to allow some freeboard during wet pluviation, allowing the sand grains to approach terminal velocity, even for the very top of the specimen. The mold extender may not be needed for other sample preparation techniques.

7. Place an old section of membrane over the gap between the mold and the mold extender.

> This piece of membrane should be about 4 inches long. It both centers the mold extender and seals the gap for vacuuming the membrane.

8. Apply a vacuum to the mold, and gently pull the top edge of the membrane up and over the upper lip of the mold extender. Make sure that the membrane is pulled tightly against the inside of the mold and that there are no wrinkles in the membrane inside the mold.

9. Fill the mold with distilled, de-aired water.

> As currently plumbed, you should be able to open the leftmost bottom back pressure valve located at the base of the cell. After a couple of inches of water are in the mold, I like to use the baster to squirt water into the annulus between the mold and the bottom platen, expelling any bubbles there. I also like to make sure there are no bubbles on the sides of the mold.

Wet Pluviation

Wet pluviation mimics the natural fluvian and lacustrine deposition of soils (Vaid and Negussey, 1984, 1988). However, wet pluviation works best for poorly-graded materials that are less likely to experience particle segregation. Additionally, materials with fines require long settlement times. If your material meets these conditions, another method of sample preparation is warranted. Moist tamping and slurry deposition are popular alternatives (see Bradshaw and Baxter (2007); Wang et al. (2011). The wet pluviation procedure outlined here closely follows that of Vaid and Negussey (1988).

1. Place an appropriate amount of soil into a glass flask. The appropriate amount can be determined with the following equation:

$$m = \frac{G_s \rho_w V_T}{e + 1} \tag{H.1}$$

where m is the mass in grams, G_s is the specific gravity of the material, ρ_w is the mass density of water ($1 g/cm^3$), V_T is the total target volume of the specimen in cubic meters, and e is the target void ratio of the specimen. For completeness, e can be calculated from the relative density (D_r) as follows:

$$e = e_{max} - \frac{D_r}{100}(e_{max} - e_{min}) \tag{H.2}$$

where e_{max} and e_{min} are the maximum and minimum void ratios of the material.

2. Fill the flask with distilled water and place it on a heat source. Allow it to boil for at least 20 minutes, and allow it to cool for some time after boiling.

Instead of boiling the soil and water mixture, you could try to add de-aired water to the flask or simply apply a vacuum to the flask instead of boiling. I find I get best saturation results when boiling. Saturation also seems to be better when the sand grains are still warm when pluviated.

3. Fill the flask to the brim with de-aired water, place the stopper and spout firmly into the flask, and fill the spout with de-aired water.

I find that the boiling causes some particle segregation in the flask. Now is a good time to put your finger over the spout and shake/mix/desegregate the sand particles.

4. Pour the soil slurry from the flask into the mold. If the spout is kept under the surface of the water, none of the slurry should spill. Vaid and Negussey (1988) call this 'mutual displacement of sand and water under gravitational influence.' Gently level the surface of the sand after you have emptied the flask.
5. Place the top of the cell on the vertical columns and bolt it down. Insert the normal actuator piston through the top of the cell and screw it onto the top platen.
6. Slowly lower the top platen through the mold extender until it no longer moves under its own weight. The tubing to the top platen should not yet be attached.
7. Fill both back pressure burettes with distilled water. Leave the 'Bypass 2' valve open. Open all valves in the bottom back pressure lines so that water passes through the sample and begins to come out of the top platen port. Open the valves of the top back pressure lines, clearing the unattached top platen tubing of air bubbles. Attach the top platen tubing to the port of the top platen.
8. While applying moderate pressure on the normal piston apply vibrations to the side of the sample until the desired (and premarked) height is reached.

I originally used a blunt-tipped engraver tool applied to the side of the mold to cause densification, but lately a few decent taps with a rubber mallet has been my preferred method.

9. Tighten the shaft lock to prevent movement (optional).
10. Place the mating collar around the normal piston and screw the normal actuator contact plate onto the end of the normal piston.
11. Turn off the vacuum applied to the mold, and close the vacuum port valve.
12. Place the membrane onto the top platen with two o-rings to secure it.
    (a) Pull the top of the membrane up and off of the mold extender and onto the top platen.
    (b) Pull the small membrane from the mold extender to down around the mold.
    (c) Roll the o-ring from the mold extender to above the top platen.

(d) Remove the mold extender.

(e) Roll the other o-ring from the mold to the top platen and place the previous o-ring on the platen, as well.

(f) Now pull the small piece of membrane (the one that was used to seal the mold extender) up and out of the way. You can't remove it entirely because it is around the normal actuator, but at least move it so it isn't touching the specimen at all (i.e. around the top platen or around the piston).

(g) Ensure that the o-rings are well placed on the platen and that there are not any wrinkles in the membrane or sand in the annulus between the membrane and the platen.

> The pool filter sand we originally used would often get stuck in that annulus. This often caused leaks. I had a few methods to deal with it. First, I added some tape to the inside of the mold extender so that the annulus is small. Second, if the top platen is lowered into position very slowly, the exiting water doesn't have the velocity to pick up particles. Third, you can spray a stream of water into the annulus before pulling the membrane off of the mold extender and onto the top platen.

13. Close the 'Bypass 2' valve, valve 'B', and the bottom back pressure valve at the cell. Switch the top pore pressure line to 'vacuum.' Slowly apply a vacuum to the sample, looking for leaks. You should see the level in the top back pressure burette rise slightly then stop. You may also see a bubble or two in the top pressure line coming out of the top platen. If the bubbles continues to move, you have a leak, obviously.

> It may not be necessary to close both the valve 'B' and the bottom back pressure valve at the cell, but I once thought that the valve at the cell was leaking, and I also like to isolate the lines as much as possible, out of habit.

14. If you don't have a leak, remove the mold from around the specimen. Adjust the top and bottom o-rings, if necessary.

> If you do have a leak, you need to find it. It could be a hole in a membrane, a leak or wrinkle at the interface between the membrane and the platens, or it could simply be a piece of tubing that hasn't been completely inserted into a connection. If you have some vacuum applied to the specimen, you can remove the mold and try to find leaks in the membrane. If it is a slow leak, sometimes filling the cell and applying a cell pressure will clear up the issue enough for a test. Otherwise, you may need to prepare a new sample.

15. Measure and record the height and diameter of the sample.

> I measure the diameter in three places and take the average. The best way to measure the diameter is using a π tape. Remember to account for the thickness of the membrane when putting the diameter into the software. Neglecting to

subtract the thickness of the membrane can significantly change your calculated relative density.

H.3.3 Assembly of Equipment

After preparing the sample and placing it under vacuum, the cell needs to be assembled to continue. It is done as follows:

1. Make sure the top of the cell is firmly attached to the posts.
2. Place the acrylic cell wall over the sample, columns, and top of cell and onto the base of the cell. Make sure the o-ring on the top of the base of the cell is in its channel and lightly greased.
3. Place the top ring onto the top of the cell and securely bolt it on. Take the time now to make sure that all the parts of the cell are fitting well together.
4. Use a piece of tubing to connect the faucet to the drain valve at the base of the cell. Both the drain valve and the valve at the top of the cell should be open.
5. Slowly turn on the water at the faucet and fill the cell with tap water.

   An approximately one-centimeter gap of air should be left at the top of the cell. This is to prevent the dynamic motion of the normal actuator from increasing and decreasing the cell pressure during the cyclic phase of the test.

6. Once the cell is full of tap water, turn off the tap, close the bottom cell valve, and attach a piece of tubing from the top of the cell to the cell pressure port on the pressure panel.
7. Manually increase the cell pressure while decreasing the vacuum. I like to keep the effective stress between 15 and 30 kPa. A slight vacuum can be left on the sample; we will be using it soon.

Normal Actuator Connection

At this point, the cell should be completely assembled, and, before continuing, I like to place the normal actuator into contact with the normal piston. You could wait to do this until after the manual saturation section, but I prefer to do it now. To save time, I often connect the normal actuator and piston while the cell is filling with water.

1. In the software, open the Digital Outputs window ($1 \Longleftrightarrow 0$, $0 \Longleftrightarrow 1$) and make sure the *Tilt* output control is on (green).
2. Flip the tilt switch (mounted to the right of the cell base) to the *FORWARD* position. As the load frame arms swing up, make sure that everything is clear, including the normal actuator and piston.

3. Turn on the hydraulic pump to low pressure by opening the Hydraulic Tool pump window and clicking on the 'Low P' button.
4. In the Outputs Function window, from the 'Output:' drop down box, select 'Normal Actuator.' Now, in the 'Feedback' drop down box select Axial Stress Difference.
5. If the readings in the windows on the left and right side of the Output Function window do not match, click on the Reset button to make them equal.
6. Open the PID window, and find the Normal Actuator and Axial Stress Difference options. Change the P value for Axial Stress Difference so that it falls around 1. Remember the old value because we will change it back in a minute.

   > If the P value is too high, the software will have trouble coping with the sudden contact between the normal actuator and the normal piston. Usually, this causes the actuator to lose control and go into high frequency oscillations, perhaps prematurely liquefying your specimen.

7. In the Output Functions window, turn on the Axial Stress Difference controls. The normal actuator may move upward at this point. If it moves downward, turn it off immediately and decrease the Axial Stress Difference and turn it back on.

   > If you haven't caught on yet, the Set Point control values can be changed by either typing a value into the left box (and pressing enter), dragging the left scrollbar, or by clicking on the up and down arrows at the ends of the scroll bar. For example, if I want to increase the axial stress difference, I would click on the down arrow (or the up arrow if the inverse bar box is selected).

8. Pull down on the normal actuator below the load cell. The actuator should begin to move in the direction pulled. Make sure to keep your person out from in-between the actuator and the normal piston head! Allow the actuator to make contact with the piston head. You may have to slow it down as it arrives.
9. Screw the collar onto the actuator until the two parts (actuator and piston) are tightly mated.
10. Zero the normal displacement in the software using the inputs offset tool.
11. Turn off the normal actuator in the Outputs Function window, and turn off the pump.
12. Change the P value of the Axial Stress Difference back to what it was before.

H.3.4 Saturation

Most tests require that the sample is saturated. Accurate pore pressure and volume change measurements are very difficult without a saturated sample. If you know that you are studying the response of unsaturated or partially saturated samples, feel free to skip this section. Otherwise, you will need to saturate your sample specimen.

Flushing the Specimen

We took care to prepare our sample so that it would be as saturated as possible: we flushed the lines before starting, we boiled our sample, and we filled the mold with de-aired water. Despite our best efforts, however, the B value of our specimen (Skempton, 1954) is probably less than 80% at this point. Most testing regimes will require the saturation (as measured by the B value) to be upwards of 95%. Frankly, the higher saturation, the better. One way to increase our saturation before back pressure saturation is to flush the sample (and lines, equipment, etc.) with de-aired water. Physically, there are several different ways that this flushing may be performed, and the researcher should feel free to try whatever they deem prudent. However, I have found the following method to give the best saturation values:

1. Make sure you have at least half a tank of de-aired water in the Nold deaerator (more may be needed).
2. With a slight vacuum on the top pore pressure lines and the selector switch of the bottom pore pressure lines set to vent, open all pore pressure valves (A, B, E, and the pore pressure valves at the cell).
3. Allow the column of water in the bottom pore pressure burette to move into the sample and into the top pore pressure burette.
4. When the top pore pressure burette is full or the bottom pore pressure burette is nearly empty, close both the bottom and top pore pressure valves at the cell.

> The purpose of flushing the sample is to remove as many air bubbles as possible from the sample specimen and equipment before beginning the back-pressure saturation. If the water level in the burettes is allowed to completely exit the bottom of the burette, air bubbles will be trapped in the piping when the burette is refilled. Additionally, the water levels should not be allowed to overflow the top of the burettes; the pressure regulators work best and last longest if they remain dry. In other words, keep the water levels in the top and bottom back pressure burettes where you can see them.

5. Change the top back pressure selection switch to 'Vent' and the bottom back pressure selection switch to 'Vaccuum.'
6. At the cell, unplug the top back pressure line and allow the water to drain from the top back pressure burette (but not completely!).

> Once the water has been through the sample once, we consider it waste water and do not want it to enter the sample again. In order to facilitate the previous step, I keep a bowl under the cell connection. I do not close the top back pressure valve at the panel (valve E), but unplug and replug the tubing at the cell connection. I allow the bowl to fill up as I continue to flush; it provides me a way to gauge how much water has been flushed through the sample specimen.

7. Refill the top back pressure burette by opening the fill switch that resides to the lower left of the top pore pressure burette. (Again, don't let it overfill!)

8. Transfer the water from the top pore pressure burette to the bottom back pressure burette by opening valve C. Close valve C when finished.
9. Switch the top back pressure selection valve to 'Vacuum' and the bottom back pressure selection valve to 'Vent.'
10. Repeat steps 2 to 9 until a few pore volumes of water have been flushed. Generally, I use the rest of the water in the deaerator tank, or about two small bowlfuls of water.

> On writing this section, I tried to find the paper that discusses flushing and how much. I couldn't find the paper. If I remember correctly, it said to flush two pore volumes. In any case, flushing isn't absolutely necessary, and you may not be able to flush fine-grained samples. However, for sand specimens, the longer I have flushed the specimen, the greater success I have had at achieving high B values at low confining pressures.

Manual Back-Pressure Saturation

Once the sample has been flushed, you are ready to begin back pressure saturation. The GCTS equipment can perform automatic back pressure saturation, but I prefer to manually back pressure saturate. Before beginning the following steps, make sure that you have cell pressure and back pressure selection valves set to 'Manual Pressure,' and valves A, B, and C on the pressure panel should be open. The bottom back pressure and cell pressure valves at the cell should also be open. Bypass valves 1 and 2, valve E, and the top pore pressure valve at the cell should all be closed. A few inches of water should remain in the back pressure burette.

> This is a good point to zero and bleed the pore pressure sensor.
>
> 1. Change the bottom back pressure selector valve to 'Vent.'
> 2. Measure the approximate height of the midpoint of the specimen, and mark the corresponding height on the bottom back pressure burette. (It should be somewhere around the 120 point of the burette for a 2.8-inch-diameter specimen.)
> 3. Make sure the water level in the burette is a few inches higher than the point we just marked.
> 4. Remove the bleed screw from the pore pressure transducer, allowing water to bleed out. Once the burette level arrives at the mark from step 2, replace and secure the bleed screw.
> 5. Open the Inputs Offset window in the software. Zero the 'Back/Pore Pressure' reading. (We want it to be zero at the center of the specimen.)
> 6. Change the bottom back pressure selector valve back to 'Manual Pressure.'

In this section, several very specific pressures will be given, but there is no need for exact and specific values to be used. In back pressure saturation and the measurement of the B value, the relative cell and back pressures are what matter. In all cases, the cell pressure should be greater than the back pressure. I like the effective stress ($\sigma'_c = \sigma_c - u$) to be somewhere between 10 and 40 kPa. It should remain below about 40 kPa so that I don't consolidate my sample during this saturation stage. The effective stress should be greater than about 10 kPa so that the specimen retains some stiffness.

Also, within this manual back pressure saturation section, when I refer to changing pressures, it is always using the manual pressure regulators at the top of the pressure panel. Additionally, when I refer to changing the pore pressures, it is always the bottom back pressure regulator to which I refer.

1. With the cell pressure around 20 kPa, increase the bottom back pressure to around 10 kPa.
2. Close the bottom back pressure valve at the cell (assuming the pore pressure sensor is still mounted at the cell). (I generally also close valve B, but this is not absolutely necessary.)
3. Gently raise the cell pressure by some amount. You should see the back pressure also rise. I like to increase the cell pressure each time by 20 kPa. It is a nice round number, and it makes calculating the B value is easy.
4. Calculate the B value of the specimen.

   The B value is the change in back pressure divided by the change in the cell pressure ($B = \frac{\Delta u}{\Delta \sigma_c}$). A B value of 1 corresponds to a fully-saturated specimen. We would like our B value to be greater than 0.95.

5. Manually increase the back pore pressure at the pressure panel until it equals the back pressure felt by the sample. You will need to use the pressure gauge on the pressure panel in conjunction with the software readout of back pressure. (We want to equalize the pressure on both sides of the closed back pressure valve before we open it so that we don't overconsolidate or otherwise disturb our specimen.)
6. Open the bottom back pressure valve (and valve B, if you closed it earlier).
7. Continue to increase the back pressure until you have reached your target effective stress (maybe 20 kPa).
8. Wait for about 20 to 30 minutes to allow the internal pressure differences to dissipate throughout the specimen and to allow the air to go into solution.
9. Repeat steps 2 to 8 until the target B value has been reached.

   I generally move quickly up to about 100 kPa before I wait in between iterations. Also, for sands, 20 to 30 minutes of waiting may be overkill; you might achieve 80% of the increase in saturation by waiting only 5 or 10 minutes. However, your mileage may vary.

Switching From Manual to Servo Pressure Control

> This is a slightly delicate procedure; I have several times lost a sample by accidentally removing all effective strength. Be sure to read through the directions before beginning these steps.

In preparation for cyclic loading, the control of the cell and back pressure must be switched from manual to servo.

1. Isolate the sample by closing the cell and pore pressure valves at the cell.
2. Open the Outputs Function window in the GCTS software, and select 'Cell Pressure' at the 'Outputs:' drop-down list. The 'Feedback:' drop-down list should read 'Cell Pressure,' as well.
3. Manually lower the cell pressure at the pressure panel until there is none applied.
4. Switch the cell pressure selector switch to 'Servo.'
5. In the Outputs Function window, turn on the servo-controlled cell pressure. The cell pressure should automatically increase to its previous value. If it doesn't, type the correct value into the small window on the left side of the Outputs Function window.
6. Once the cell pressure has stabilized, open the cell pressure valve at the cell for a few moments and close it again. This ensures that the cell pressure within the cell is really what we think it is before we continue.
7. In the Outputs Function window, select 'Back Pressure' from the 'Outputs:' drop-down list. Select 'Back/Pore Pressure' from the 'Feedback:' drop-down list. Once the back pressure reading shows up on the Outputs Function windows, switch the 'Feedback:' to 'None-Open Loop.'

   > Because I moved the pore pressure sensor from the back of the pressure panel to the front of the cell, it is necessary to use the Open Loop feedback. 'Open Loop' simply means that the system guesses the correct pressure, but does not correct the guessed pressure; i.e. there is no feedback. If we tried to set the pore pressure with the servo control with the feedback set to back/pore pressure and the back pressure valve at the cell closed, the intended pressure and the pore pressure read from the sensor would not agree exactly. The system would then open or close a servo regulator to adjust the pressure. When this adjustment made no difference (because the sensor is behind the closed valve), it would open or close the servo regulator to a greater degree. Eventually, the pressures on the servo side will be as high or as small as possible and the system will abort. Feel free to try it yourself, just make sure to have your sample isolated, or you will waste several hours of preparation and saturation.

8. With the feedback set to 'None-Open Loop,' manually remove back pressure until there is none applied.

9. Change both the bottom back pressure and top back pressure selector switches to servo. Open both bypass valves. (The servo pressure port for the top back pressure selector valve is plugged; we can control both top and bottom back pressures by opening the bypass valves.)

10. In the Outputs Function window, turn on the Back Pressure control. (Feedback should still be 'None - Open Loop.') The pore pressures in the panel should rise (see the gauge on the pressure panel).

11. In the Outputs Function window, adjust the pore pressure (as read from the pressure panel gauge) until it matches the pressure given by the sensor. (The 'pfs' units in the Outputs Function window under open loop stands for *percent full scale*. Full scale pressure should be around 1000 kPa. So if your back pressure is, say, 350 kPa, the pfs value should be around 35 ($350/1000 * 100\% = 35$).

12. Make sure the locking collar on the normal actuator is firmly locked.

13. In the Outputs Function window, select 'Normal Actuator' from the 'Outputs:' drop-down list. Select 'Normal Displacement' from the 'Feedback:' drop-down list.

14. Turn on the pump.

    I am assuming that the normal actuator is in contact with the normal piston, as I described in section H.3.3. If the normal actuator and piston are not currently connected, go back and follow the directions in that section first, then come back here.

15. Turn on the normal actuator in the Outputs Function window (still set to displacement controlled). Ensure that the command and response values in the Outputs Function window match before you turn it on.

    I like to turn on the normal actuator when it is set to displacement controlled because the actuator is less likely to move unexpectedly when it is displacement controlled (versus stress-controlled).

16. Now switch the normal actuator feedback to 'Axial Stress Difference.' Change the control value so that the axial stress difference it equal to zero.

    If you forget to change the axial stress difference to zero, it will carry the same value through the consolidation phase. Be careful changing this value, however, because if it is a big change, the change in normal displacement may be significant.

17. Gently unlock the normal actuator locking collar.

18. Once the normal actuator settles down, slowly open all top and bottom back pressure valves. I suggest starting with the bottom back pressure valves first.

19. Once the pore pressures have stabilized, return to the Outputs Function window. Switch the output to Back Pressure. Now change the feedback from 'None - Open Loop' to 'Back/Pore Pressure.'

20. Open all valves in the cell pressure lines. Now all valves in the pressure panel and at the cell (except the cell drain valve) should be open. Double check that these are all open! If you forget to open one of these, you will ruin your specimen and your day!.

> Your specimen should now be entirely under the SCON control: normal actuator, cell pressure, top and bottom back pressure. Again, make sure all valves are open and the normal actuator locking collar is loose. I had you turn on the normal actuator in the middle of switching from manual to servo pressure control. In reality, you could turn on the normal actuator at any time during saturation or after switching to servo-controlled pressure. I like to turn it on at this point, however, so that my specimen is still isolated. On multiple occasions I have turned on the normal actuator to immediately have a normal actuator control error and subsequent abort. When this occurs, control of the pressures is also lost, and, likely, your specimen will also be lost. Having the cell and pore pressure valves closed for these steps gives me an extra safeguard.

Automatic Back-Pressure Saturation

As mentioned, the GCTS software can perform automatic back-pressure saturation as another phase of testing. For several reasons, I stopped using this feature. First, the back-pressure saturation phase increases the time that the pump must be running, and I dislike the noise of the pump. Also, if the back pressure servo-regulator is not well calibrated (this seems to always be the case, despite my efforts), the specimen is often overconsolidated or allowed to soften as the effective stresses stray from the intended values.

Of course, you may find the auto-saturation phase to be helpful; feel free to experiment. It is pretty straightforward to add the phase at the beginning of your test.

H.3.5 GCTS Control of Testing

This section will guide you through the actual running of the test in which the GCTS system controls the specimen and collects data.

It is assumed that a suitable test file already exists. If not, you need to create one. See the GCTS manuals and section H.5.1 in this manual. *This* section will walk you through starting and running a test with a consolidation and cyclic loading phase.

1. Before starting, note the height of the water column in the pore pressure burettes.
2. Click on the 'P/S/S' (or 'Projects') button to open the 'Projects/Samples/Specimen' window. Select the appropriate project and sample, or create new ones.
3. Click 'New' to create a new specimen. Give an appropriate 'ID' (and description, if desired). Fill in the boxes for the measurements of height and diameter. On the next tab, change the specific gravity (G_s) as needed. When ready, click 'Ok.'

4. Select the correct test. Click on the 'Edit' button if you need to make changes. When finished, click 'Ok.'
5. A screen will ask you if you want to apply a 5 kPa axial stress difference. If your normal actuator is already in contact, click 'Skip.' (I've had problems with this step in the past, so I always skip it. Your mileage may vary.)
6. A screen may come up asking you if you want to correct the specimen dimensions. Choose how you deem prudent and continue.

> Now the test window should have come up. The test window shows the status of the test including the stage and phase of the test. It also has buttons at the top to stop and pause the test. If you have chosen the correct triaxial test, the cell pressure should increase until it reaches the predefined effective consolidation pressure. This is the consolidation phase. You should allow your specimen to consolidate for a minute or two, at least.

7. When normal displacements have steadied, make a note of the height of the water column in the pore pressure burettes. The difference in height from before is the amount of volume change that occurred during the consolidation phase. Each tick mark is equal to 0.25 mL.
8. When ready to proceed, close all back/pore pressure valves going into the cell. We want a cyclic test with no drainage.
9. In the test window, click 'Next' to go to the cyclic loading phase. If you are sure, click 'Ok.'
10. Behold the cyclic loading phase! Depending on your settings in the test setup, you may need to end the test manually by clicking on the 'Stop' button. Otherwise, let the test run its course.

> From the PID window, I like to open the 'Tune Chart.' If the feedback doesn't match the command of the normal actuator, you may need to adjust the P-I-D values. Usually, I only have to adjust the P values.

11. Wait for the GCTS system to transfer the test file from the SCON to the PC.

H.3.6 Cleanup/Tear Down

After the test file has transferred from the SCON to the PC, you can regain control of the GCTS system. A few final steps are needed before we are finished.

1. Lock the collar around the normal actuator.
2. Close the test window and the 'Projects/Samples/Specimens' window. Find your 'Outputs Function' window.

3. With the top and bottom pore pressure valves still closed, select 'Cell Pressure' from outputs drop-down list. Lower the cell pressure to around 20 kPa.

> I like to lower the cell pressure by degrees, making sure that my back pressure is lowering at the same time.

4. In the 'Outputs Function' window, turn off the cell pressure control as well as the back pressure. When both cell and back pressure are turned off, change the 'Output' to Normal Actuator.
5. Change the axial stress difference to zero. Take off the normal actuator mating collar and gently push up the normal actuator until it is completely clear.
6. In the 'Outputs Function' window, turn off the normal actuator. Turn off the pump.
7. Tilt the load frame back and, on the pressure panel, switch all pressure selector switches to 'Vent.'
8. In the software, open the 'Digital Outputs' dialog and turn off the 'Automatic Ball Valve.'
9. Gently open the pore pressure valves to release any remaining pressure in the specimen. Watch the back pressure burettes to make sure they don't overflow.
10. Open the cell drain valve and allow the cell to drain.

> I am usually impatient, so I manually add a cell pressure to the top of the cell to speed drainage.

11. Once the cell has drained, disassemble the cell. Unscrew the normal actuator from the top platen.
12. Remove the bolts holding the bottom platen.
13. Remove the top platen o-rings and the top platen. Using a laboratory water bottle, rinse off the sand from the top platen into a clean bowl.
14. Dump the specimen into the same bowl. Rinse off the the membrane and bottom platen.
15. Place the bowl into the oven and allow the specimen to dry for approximately 24 hours.
16. When the specimen has dried, mass the specimen and record. This is needed to calculate the relative density of the specimen.

H.4 Data Reduction

The data reduction can be performed using most data processing software. Before beginning data reduction, it is first necessary to obtain the output file from the GCTS software.

1. Open the 'P/S/S' window in the GCTS CATS software. Navigate through the projects and samples to the specimen you would like to examine.
2. Select the specimen of interest.
3. In the toolbar at the top of the 'P/S/S' window, click on the 'Edit' button. Click 'Ok' to dismiss the popup.
4. Find the appropriate space and enter the dry mass of the specimen. Click 'Ok' when done.
5. In the toolbar at the top of the 'P/S/S' window, click on the 'Export' button.
6. In the new dialog window, select an output format and the data that you would like to export. I prefer the .csv format and I select all data to be exported.
7. When finished, click 'Ok' and pick a folder to which you would like to export the data file.

For cyclic tests, several values must be calculated from the output file. These include the void ratio and relative density of the sample, the cyclic stress ratio, the number of cycles to liquefaction, and the dissipated energy to liquefaction.

H.4.1 Void Ratio and Relative Density

The void ratio (e) is calculated using the total specimen volume and the volume of solids (V_s) in the specimen. The volume of solids is calculated as

$$V_s = \frac{m_{specimen}}{G_s \rho_{water}} \tag{H.3}$$

where $m_{specimen}$ is the mass of the specimen, G_s is the specific gravity of the sample, and ρ_{water} is the density of water. The void ratio is

$$e = \frac{V_{specimen} - V_s}{V_s} = \frac{V_v}{V_s} \tag{H.4}$$

where $V_{specimen}$ is the volume of the entire specimen and V_v is the volume of voids in the specimen.

The relative density (D_r) is calculated as

$$D_r = \frac{e_{max} - e}{e_{max} - e_{min}} * 100\% \tag{H.5}$$

where e_{max} and e_{min} are the maximum and minimum void ratios of the sample, respectively.

H.4.2 Number of Cycles to Liquefaction

There are several methods used to determine the point of liquefaction during cyclic loading. Wu et al. (2004) discusses many of these methods. The most common methods define the onset of initial liquefaction as either the point at which the pore pressure ratio (r_u) is equal to unity, or point at which a certain strain is reached. You will have to decide the correct criteria for your laboratory testing needs.

H.4.3 Calculation of Dissipated Energy

Dissipated energy is defined as the area bound by the hysteresis loops of a stress-strain plot (see Figure H.15). For a triaxial test, it can be calculated by using the trapezoidal rule:

$$\Delta W = \frac{1}{2}\sum_{i=1}^{n-1}(\sigma_{d,i+1} + \sigma_{d,i})(\varepsilon_{a,i+1} - \varepsilon_{a,i}) \tag{H.6}$$

where ΔW is dissipated energy in units of stress, n is the number of data points in the stress and strain time histories, σ_d is the deviator stress (axial stress difference), and ε_a is the axial strain.

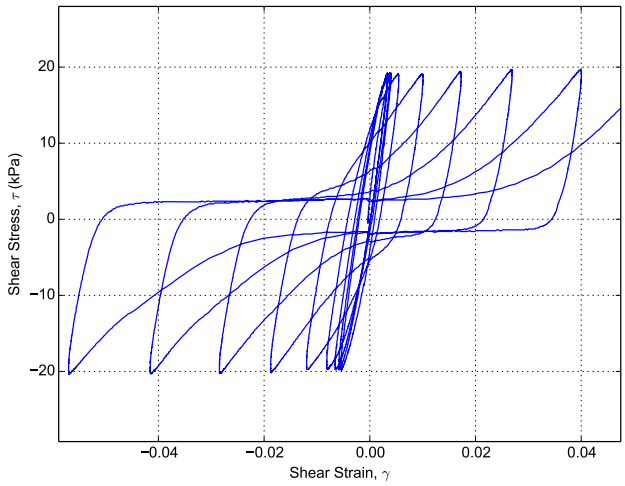


Figure H.15: The hysteresis loops of a stress versus strain plot.

H.4.4 Plotting Results

Plotting results is an important part of data reduction. Plots can help you diagnose poor tests and discover new results. You should plot both results from a single test, and the results for multiple tests. Plots for the results of a single test may be a plot showing stress versus strain, a stress time history, a strain time history, a pore pressure time history, etc. Plots of multiple test results might include CSR versus number of cycles to liquefaction (on log-log scales, the trend should be linear), and dissipated energy versus effective consolidation pressure or versus relative density. Figure H.16 shows an example of a diagnostic output plot.

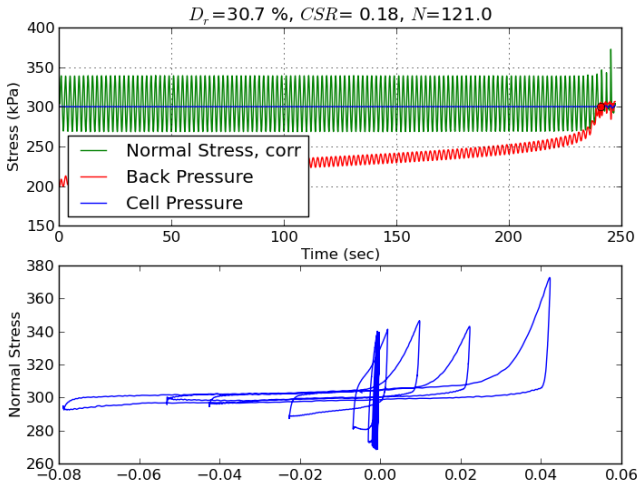


Figure H.16: An example of an output plot.

H.5 Maintenance/Preparation

H.5.1 Creating a Triaxial Test File

A cyclic triaxial test consists of two or three stages: auto-saturation (optional), consolidation, and cyclic loading. Additionally, you may wish to add a final phase to ramp down pressures and move the normal actuator out of the way. In this section, only the consolidation and cyclic loading stages will be created. You are left to your own devices to add any other stages.

1. Open the 'Test Setup' window (Figure H.17), and select 'New.'

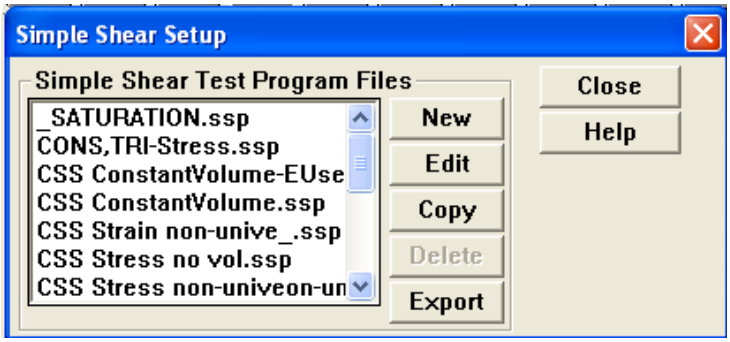


Figure H.17: The 'Test Setup' dialog box.

2. In the 'Simple Shear Program Definition' dialog box (Figure H.18) , type in the appropriate information into the 'ID' and 'Description' boxes. Click on the 'Save Objects' button.
3. In the 'Test Data Save Options' dialog box (Figure H.19), select the inputs and analog outputs that you are interested in saving. I generally select them all. Close this dialog when you are finished.
4. Once again in the 'Simple Shear Program Definition' dialog box, click on the 'New' button to create a new stage. In this dialog box (Figure H.20), select 'Consolidation' and 'Ok.'
5. In the consolidation dialog box (Figure H.21), set values as you need for your test. I like to 'Update [the] Specimen Dimensions' at the beginning of this stage, but you should make sure that it is taking care of your strains as you wish. Click 'Ok' when you are done.
6. Back to the 'Simple Shear Program Definition' dialog box, click on 'New' again, and in the next dialog box, select 'Universal' before 'Ok.'

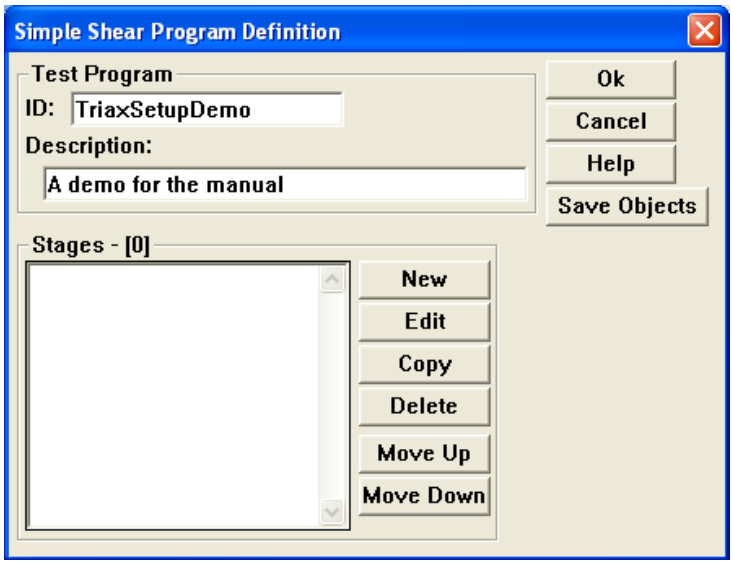


Figure H.18: The 'Simple Shear Program Definition' dialog box.

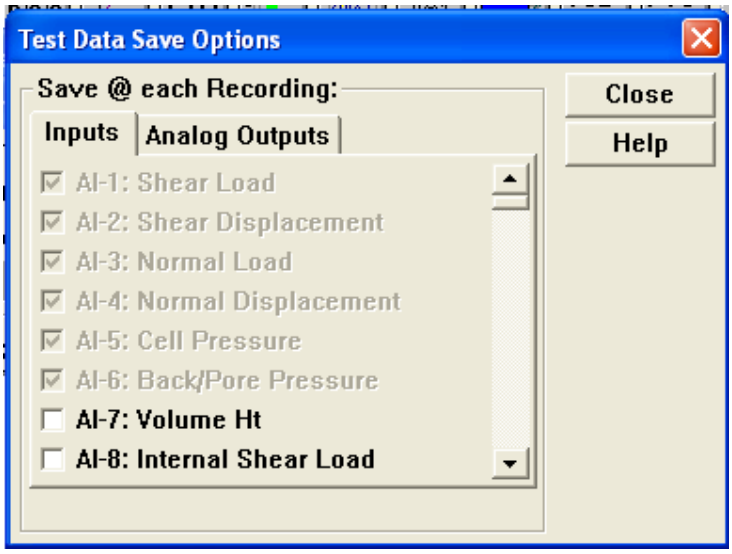


Figure H.19: The 'Test Data Save Options' dialog box.

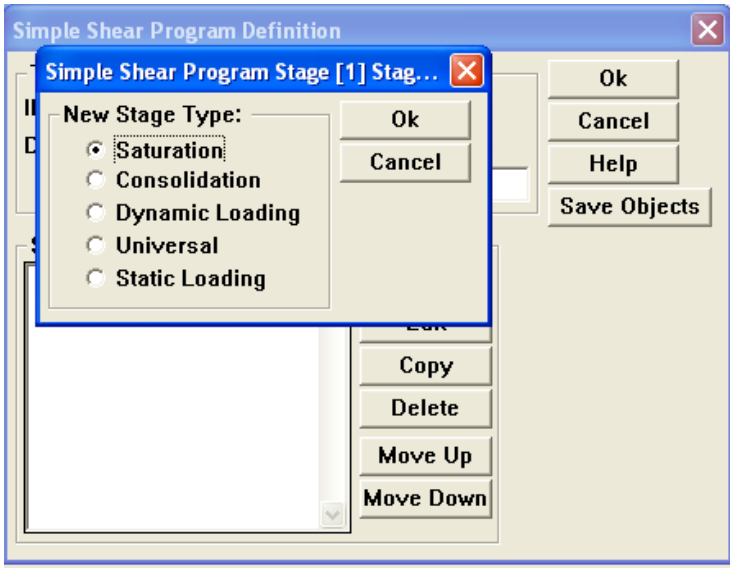


Figure H.20: The new stage dialog box.

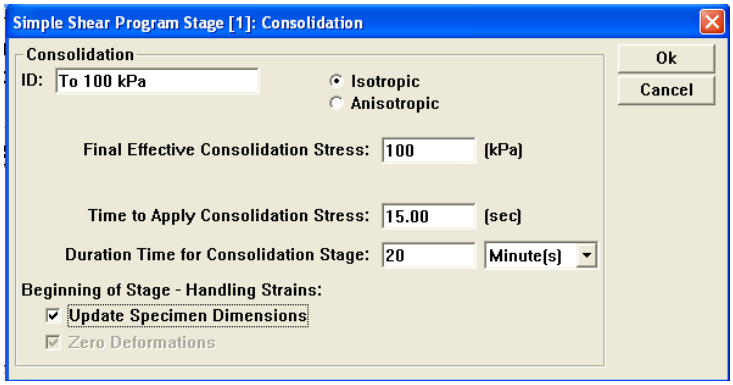


Figure H.21: The consolidation stage dialog box.

7. In the universal stage dialog box, create a new universal program (Figure H.22). Give the new universal program an 'ID' and 'Description' (Figure H.23), and create a new phase.

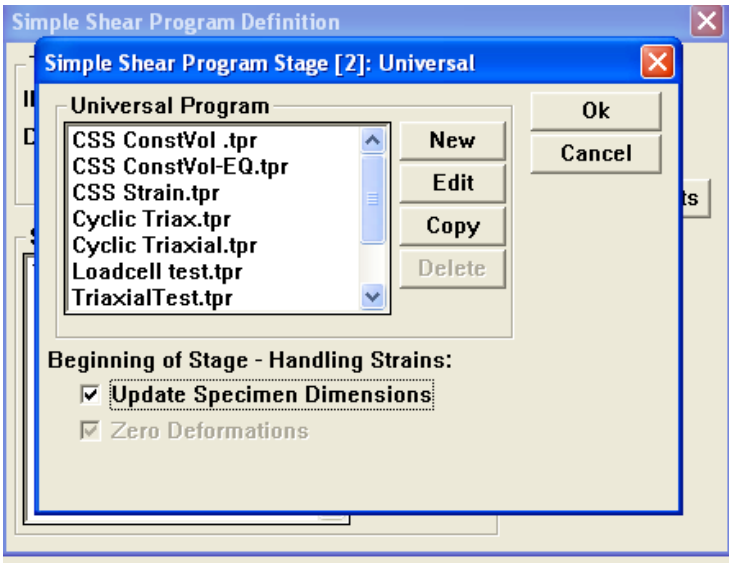


Figure H.22: The universal programs dialog box.

8. In the phase dialog box, in the Duration tab (Figure H.24), set an appropriate duration. I like my tests to stop when a failing strain occurs or when I am about to reach the limits of the machine. Make sure to set this as needed.
9. In the 'Data Acquisition' tab (Figure H.25), select the type of data acquisition that you would like. I like to have timed increments. Pick an increment that will give you a good idea of the loading function without creating too big of a data file. The other options may be useful in addition to the timed increments.
10. Still in the phase dialog box, double-click on the 'A0-2: Normal Actuator...' in the bottom of the box. This opens another dialog box (Figure H.26); make the following changes:
    - Select 'Sd - Axial Stress Difference' from the feedback drop-down list.
    - Select the 'Sine' waveform. (Or whatever wave form you want. Change the delay or phase shift options if for some reason you need it.)
    - Select 'Mean' control value and make sure it reads 'Absolute/Change to 0 kPa.' (If you are performing a test with loading oscillating around some static shear stress value, you would have to change this setting to something other than 0 kPa.)

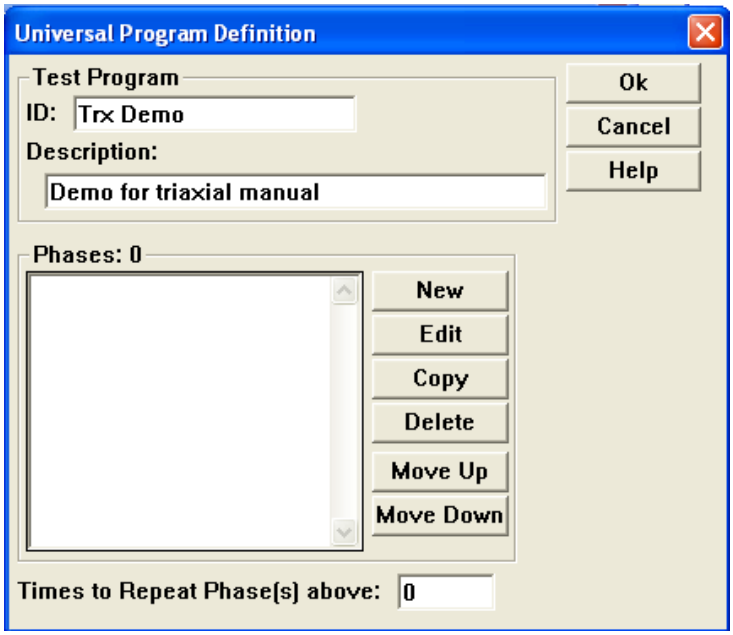


Figure H.23: Creating a new universal program.

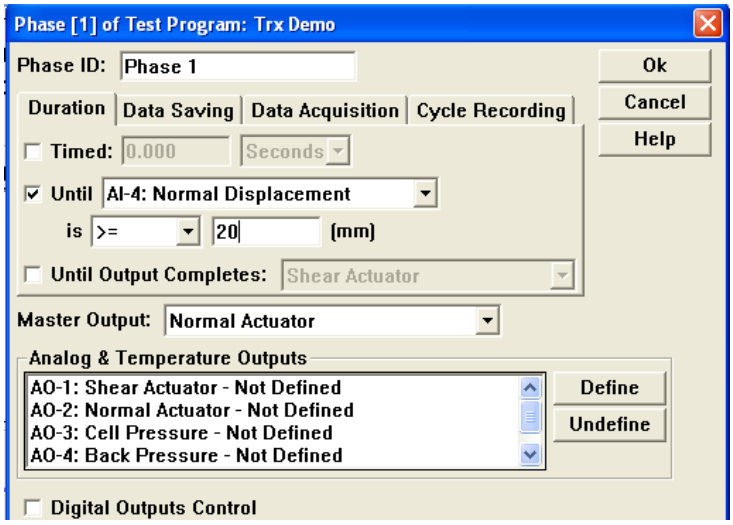


Figure H.24: The duration tab of the phase dialog box.

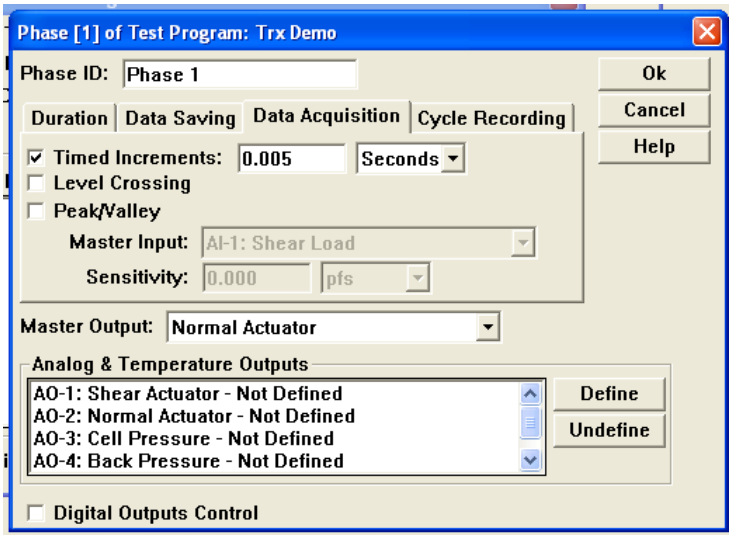


Figure H.25: The data acquisition tab of the phase dialog box.

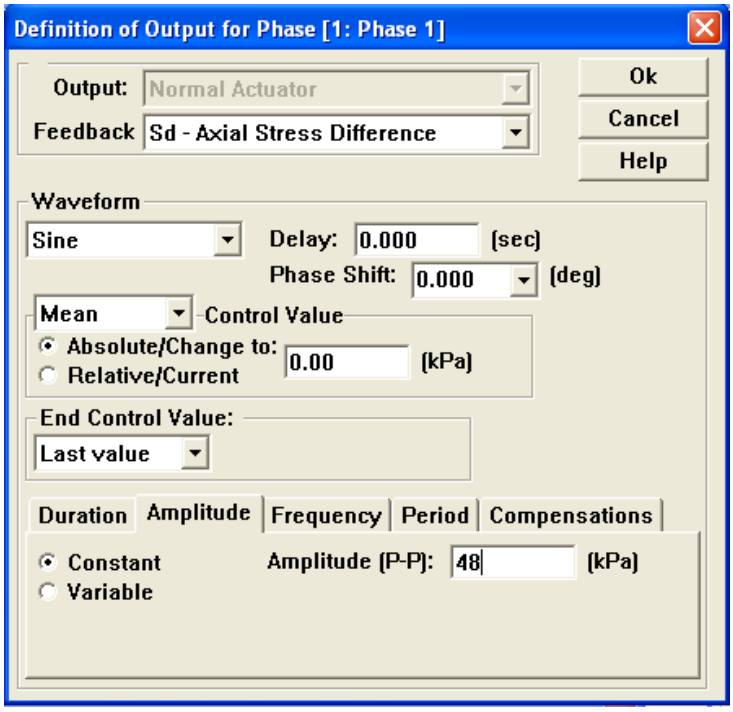


Figure H.26: The normal actuator definition dialog box.

- On the 'Amplitude' tab at the bottom, type in the appropriate value. (For an isotropically consolidated triaxial test, $\textit{Amplitude (P-P)} = 2 \cdot 2 \cdot CSR \cdot \sigma'_c$. One 2 is to change from single-amplitude to double-amplitude (P-P), and the other is needed for the calculation of CSR for triaxial tests.)
- On the 'Frequency' tab, input the frequency of the sinusoidal loading. I normally choose 0.25 Hz, but it seems like a lot of literature cite a frequency of 0.1 Hz (e.g. Wu et al. (2004)). (Alternately, you can adjust the period instead of the frequency.)
- Click 'Ok' when you are finished.

11. Now double-click on the 'A0-3 Cell Pressure ...' line. In the new dialog box (Figure H.27), make sure the feedback is cell pressure and the waveform is constant. Click 'Ok' when finished.

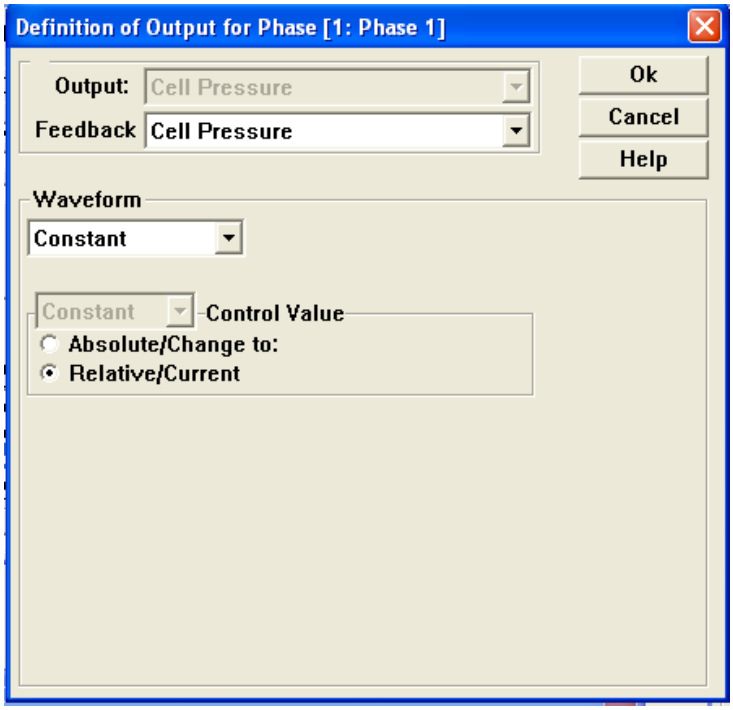


Figure H.27: The cell pressure definition dialog box.

12. Now double-click on the 'A0-4 Back Pressure ...' line. In the new dialog box (Figure H.28), make sure the feedback is 'None - Open Loop' and the waveform is constant. Click 'Ok' when finished.
13. Back in the phase dialog box, click on the 'Digital Outputs Control' check box at the bottom of the window. In the new part (Figure H.29) of the window, change

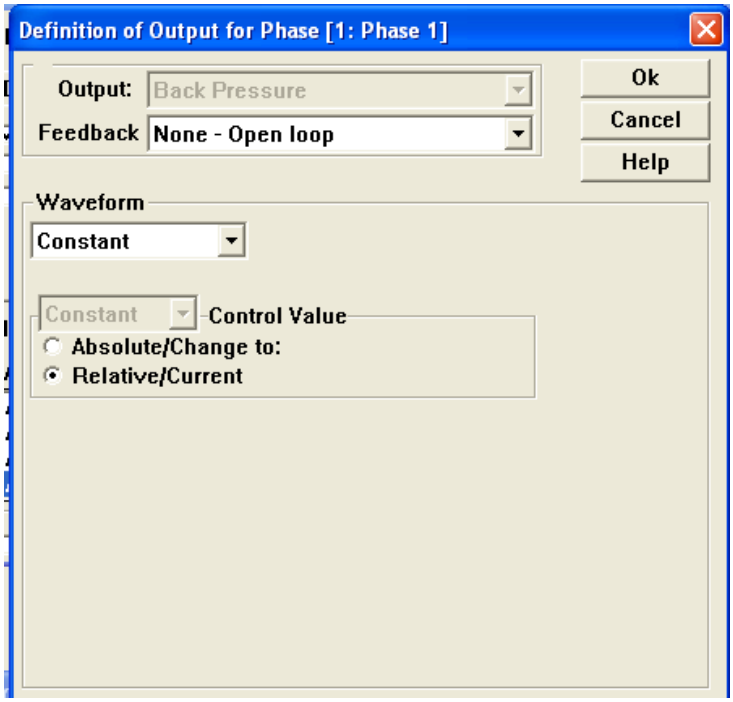


Figure H.28: The back pressure definition dialog box.

the automatic ball valve control to 'On'. This will close a valve in the bottom back pressure line in the pressure panel at the beginning of this phase. For our liquefaction tests, we assume the samples are undrained during loading, and this makes sure no drainage can occur via the bottom back pressure lines. However, if you have the top back pressure selector switch set to 'Servo' and your top back pressure and bypass 1 & 2 valves open, it really doesn't matter if you have this set to on or off.

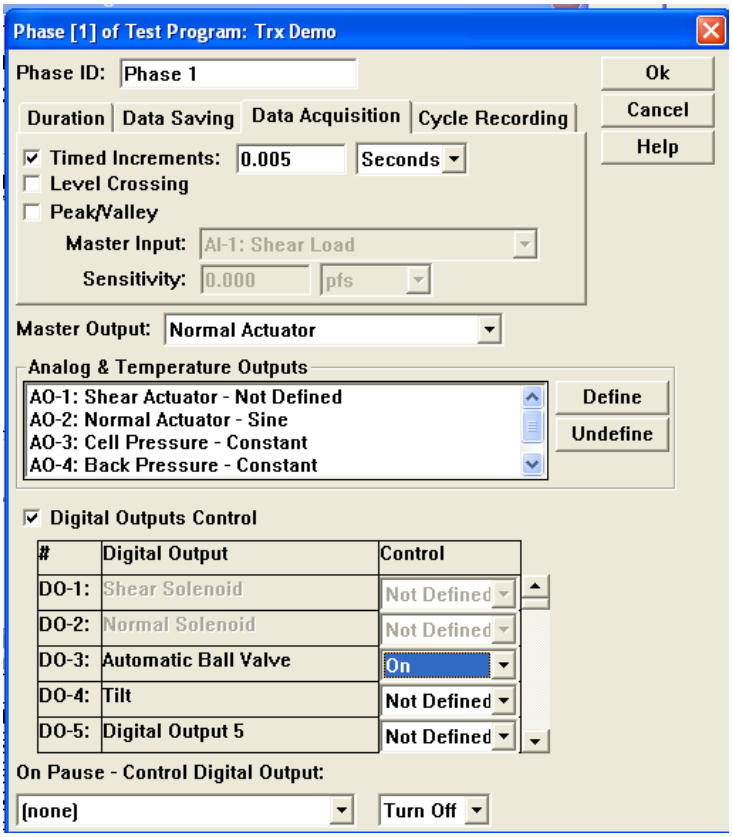


Figure H.29: The digit outputs section of the phase dialog box.

14. When finished, click 'Ok' in all the open test definition windows. You are finished creating a new test.

Arbitrary (Earthquake) Loading

An earthquake loading test is created in the same way as a normal sinusoidal test, except instead of choosing a sine waveform, 'User Defined' should be chosen as the waveform. Once 'User Defined' is chosen as the waveform, an input file can be chosen. You have to create the input file yourself. Section UNI 3.6.2.8 of the CATS Advanced and Universal Manual (v.1.96) contains details about how to use the 'User Defined' waveform.

The code snippet below shows how I have created the input file using Python. Since this is only a snippet, it is important to know that `name` is the name of the entry in `repofile` where I have stored several stress and strain time histories. The variable `tm0` contains the stress time history in this case.

```
        # extract the data node from the aggregate file
        data = repofile.getNode('/TimeHistoriesProfileN05/{}'.format(name))
        # extract the variables I need from data
        profile = data.col('profile')[0]
        depth = data.col('depth')[0]
        tm0 = data.col('taugam')[0][:,motind] # This is the stress time history
        dt = data.col('dt')[0]
        maxval0 = np.max(np.abs(tm0))

        # Normalize the time histories so that they all have max values of 1.
        tm = tm0 / maxval0

        # Limit: 4096 data points for each input file
        # See CATS Advanced and Universal Manual, pg 273
        if len(tm) > 4096:
            if (len(tm) % 4096) > 0:
                no_files = len(tm0) // 4096 + 1
        else:
            no_files = 1

        filepath = './Figures/ProfileN05/GCTSfiles/{}'.format(name)
        f2 = open(filepath + '-info.txt', 'w')

        # Create a file for each 4096 data points in my time history
        for i in xrange(no_files):
            f = open(filepath + '-{}_{}.txt'.format(i+1,no_files), 'w')

            # First line in the file is the number of data points
            if no_files > 1:
                if i < no_files-1:
                    f.write('{}\r\n'.format(4096))
                else:
                    f.write('{}\r\n'.format(len(tm) % 4096))
            else:
                f.write('{}\r\n'.format(len(tm)))

            temptm = tm[i * 4096:(i+1)*4096] # Create a temporary time hist var
            maxval = np.max(temptm)
            minval = np.min(temptm)
            meanval = (maxval + minval) / 2.
```

```
            valrange = maxval - minval
            Peak2Peak = valrange

            # Now write out the temp time history for this file.
            for j, val in enumerate(temptm):
                f.write('{}\r\n'.format(val))

            f.close()

            # To help myself with the data needed for the GCTS software, I also create an
                 additional file:
            if i == 0:
                f2.write('{}\r\n'.format(name) +
                'This is a {} time history in units of kPa '.format(motiontype) +
                ' for cyclic simple shear tests. \r\n' +
                'The motion is from a depth of {:.2f} m'.format(depth) +
                ' from {}.\r\n\r\n'.format(profile) +
                'No. of files: {}\r\n'.format(no_files) +
                'To get original amplitudes, multiply by {}.\r\n \r\n\r\n'.format(
                                                maxval0))
            else:
                f2.write(
                  'End Control Value: Given value: {}'.format(
                                            temptm[0] / 0.65) +
                ' * CSR * sig\'v (in kPa)\r\n\r\n\r\n')

            f2.write(
                'For file # {}:\r\n'.format(i+1) +
                'Mean Control Value: {} * NewScaleFactor '.format(meanval) +
                ' = {} * CSR * sig\'v \r\n'.format(meanval
                                                / (0.65)) +
                'Amplitude (P-P):  {} * NewScaleFactor '.format(Peak2Peak) +
                ' = {} * CSR * sig\'v \r\n'.format(Peak2Peak / 0.65) +
                '                  /  CSR * sig\'v  \ \r\n' +
                'NewScaleFactor = < -------------- >\r\n'+
                '                  \     0.65     / \r\n'+
                'Duration: {} points\r\n'.format(len(temptm)) +
                'Time Step: {:.4f}\r\n'.format(dt))
        f2.write('End Control Value: Last Value \r\n')
        f2.close()
```

H.5.2 Friction Baseline Tests

The GCTS equipment is not frictionless. This is especially apparent for dynamic/cyclic testing. To get a handle on the amount of extra load due to friction that the load sensors pick up (and the sample doesn't feel), a friction baseline test may be useful.

A friction baseline test is performed by running a strain-controlled test on an air- or water-filled sample. You may think of a better way to do it, but one way is to create a test in the GCTS software with a sinusoidal strain-controlled loading function at the same frequency as your normal testing regime. I begin the loading at very small strains but increase the amplitude during loading so that the final cycles of the baseline test have as large of strains

as I would expect from my soil test.

The results of this baseline test can be used to subtract friction from a regular soil test. The extra load due to friction is not always insignificant. For triaxial tests, I have found the friction to be a function of the velocity of the normal actuator, but I invite you to look closely at your own data. Figure H.30 shows some results from one baseline test on a sample of air.

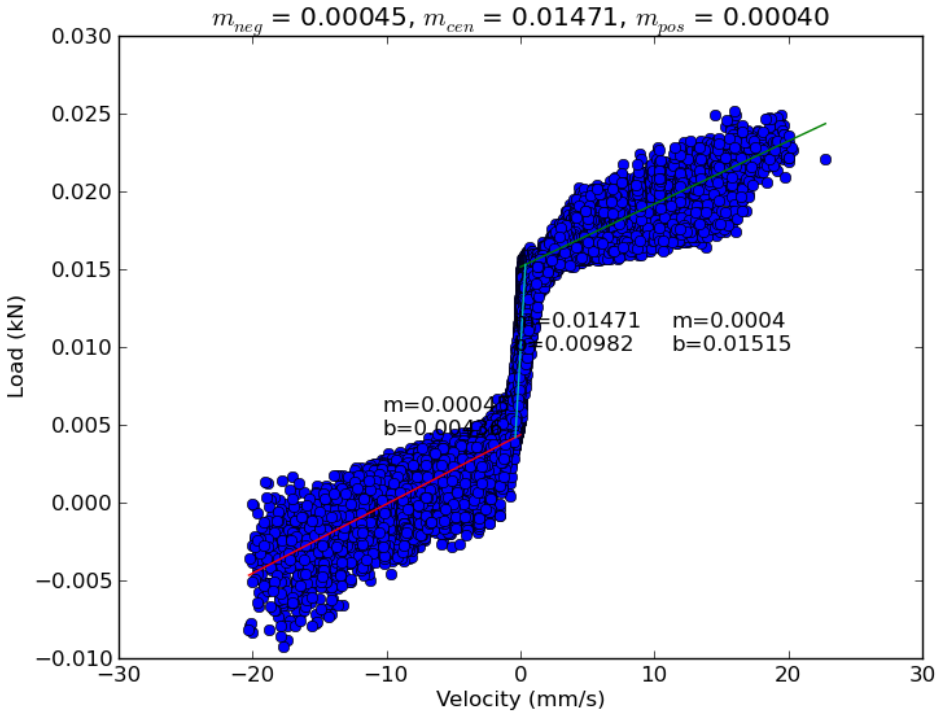


Figure H.30: Results of an air friction baseline test. The relationship between friction load and velocity can be approximated with a tri-linear fit.

H.5.3 Sensor Calibration

The calibration of the sensors should be checked periodically. Eventually, the sensors will need to be calibrated. To do so:

1. Select *Analog* from the *System* → *Inputs* menu.

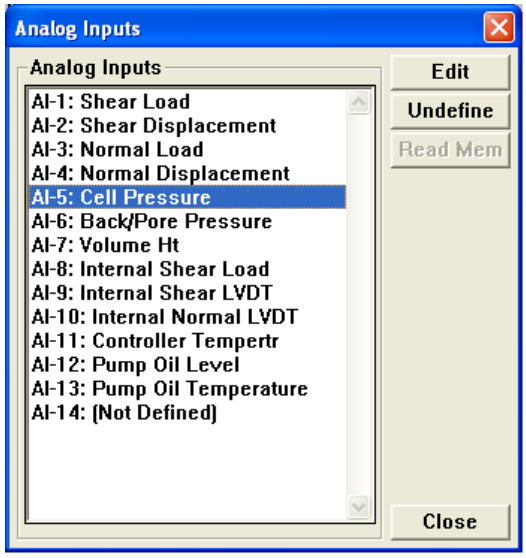


Figure H.31: The 'Analog Inputs' dialog.

2. In the 'Analog Inputs' window (Figure H.31), choose the appropriate sensor.
3. In the 'Editing Analog Inputs...' window (Figure H.32), double-check that the 'Attached Sensor' is correct. If so, click on 'Calibrate.'
4. You are given three choices on how to calibrate your sensor (Figure H.33). I generally use either the two-point method (Figure H.34) repeated a few times, or the multi-point method (Figure H.35).
5. Whatever method of calibration, apply the forcing or loadings to the sensor, as needed, and put the known values of these forcing into the dialog window.

Load Cells

For the calibration of load cells, I suggest using one of two methods to apply variable loads: free weights or a calibration ring.

If you are using free weights, contact the lab supervisor to get a frame onto which you can apply the weights. These look like a triaxial cell without the acrylic plastic shell. You place the load cell of interest between the bottom of the frame and a vertical piston arm. This piston arm leads through the top cap of the frame to a plate onto which you can place consolidation weights. You can mass the piston arm and consolidation plates before beginning. This is how you obtain your known loading.

There is a calibration ring in the lab, somewhere. If you use it, put the ring in the place of the triaxial sample with the normal piston attached to the normal actuator and the

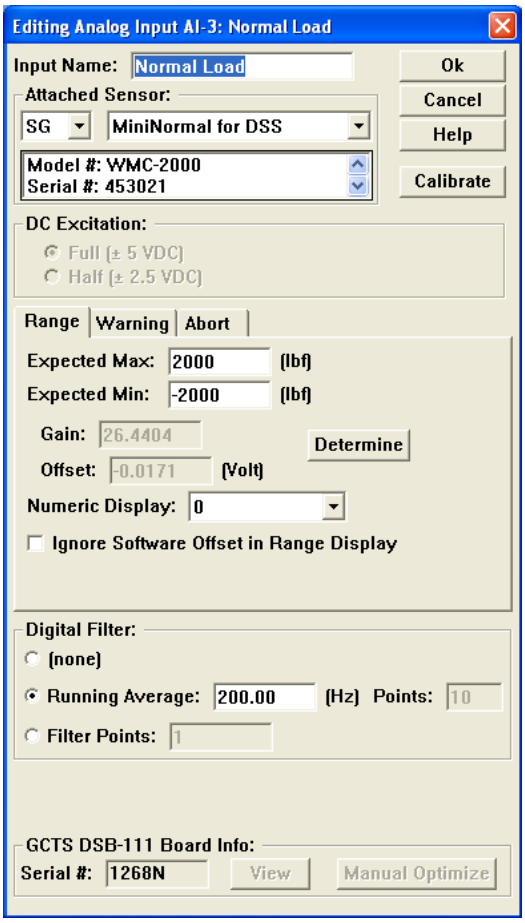


Figure H.32: The 'Editing Analog Inputs . . . ' window for the normal load cell.

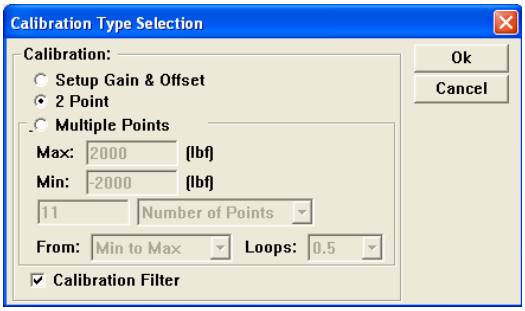


Figure H.33: Dialog box for choosing the calibration method.

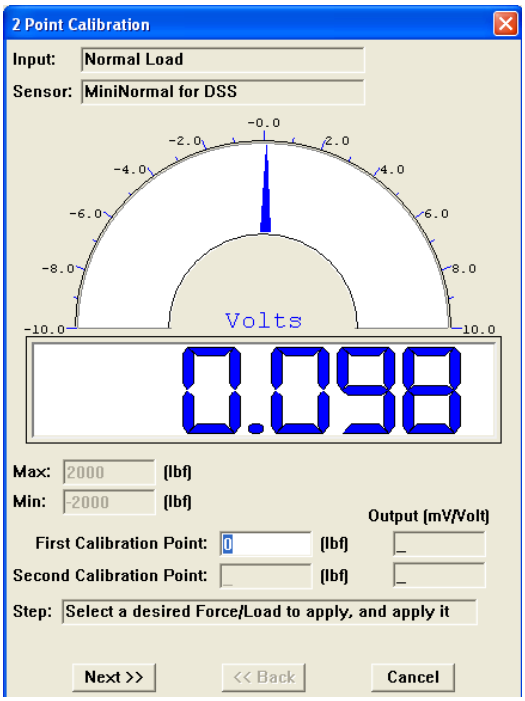


Figure H.34: The 2 point calibration method dialog.

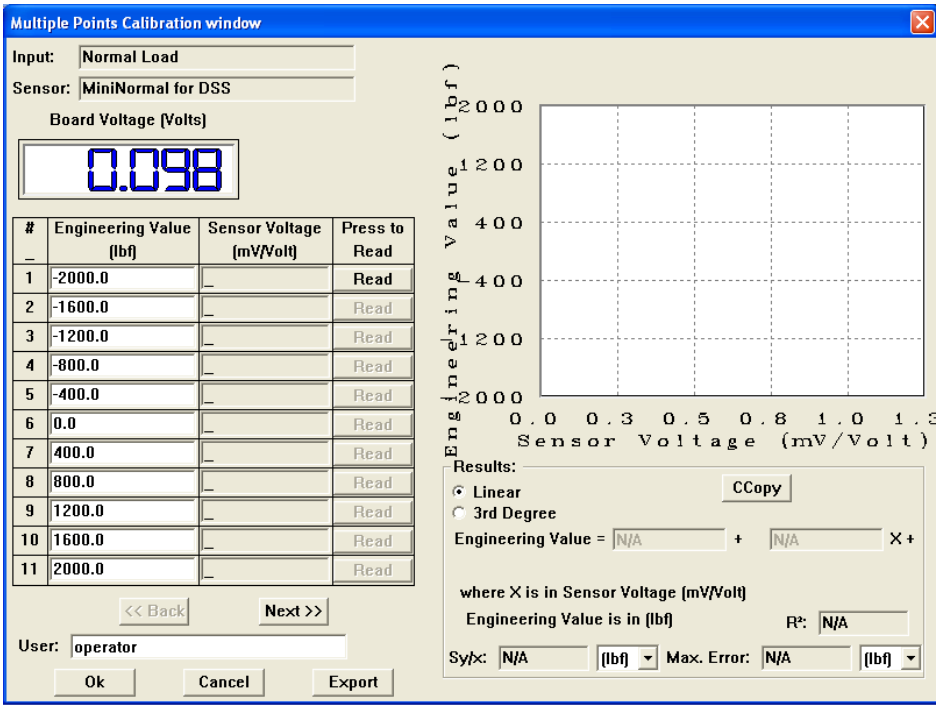


Figure H.35: The multiple point calibration method dialog.

load frame tilted forward. Use the GCTS' normal actuator to apply incremental loadings to the ring. Measure the deflection of the ring and calculate the applied load from the deflection. If you choose to use the calibration ring, you may need to calibrate the ring. At the very least, make sure it performs linearly. If it does, you can simple zero out your sensor when you are done with the calibration.

Pressure Sensors

For calibration of the pressure sensors, find a pressure gauge that you trust. The pressure panel gauge has heretofore been accurate, so it might be a good place to start. If you don't trust it, one of the lab supervisors can help you find a better gauge.

H.5.4 Pore Pressure Servo Calibration

The back pore pressure servo may need to be calibrated, especially if you intend to perform automatic back pressure saturation. To do so, consult the document `(PCP-200) Back Pressure E-P Valve setup procedure.pdf`. It was written by GCTS and should be with the other digital documents. To summarize:

1. Close the back pressure lines so that there is no flow out of the system and the back pressure sensors can get a pressure reading.
2. Open the 'Outputs Function' window. Change the 'Output' to 'Back Pressure' and the 'Feedback' to 'None - Open Loop.'
3. Change the Set-Point value to 5 pfs (percent full scale) and turn on the control.
4. Adjust the 'ZERO' screw on the back pressure servo (also known as the E-P valve; one of the black boxes mounted on the top back of the pressure panel) until the pore pressure sensor reads close to 50 kPa.
5. Turn off the back pressure control. Change the Set-Point value to 50 pfs. Turn the control back on.
6. Adjust the 'SPAN' screw on the back pressure servo until the pore pressure sensor reads close to 500 kPa. (House pressure is $\sim$ 1000 kPa. Fifty percent full scale then is $50\%/100\% * 1000kPa = 500kPa$.)
7. Turn off the back pressure control.
8. Repeat steps 3 to 7 until convergence.

H.5.5 Loading & Saving Configuration Files

In the GCTS software, configuration files save information about the current configuration of the system. This data includes the default height and diameter of the samples, the

calibration of the sensors, which load cells are being used, and the PID values of of each control-feedback loop. When switching between simple shear and triaxial tests, it is more convenient to load the required configuration file than to make all the changes by hand. Additionally, it is helpful to be able to go back to a known working configuration. After making changes to the system configuration and every few months, it is a good idea to create a new configuration file. This is done by going to the *System* → *Configuration* menu and selecting *Save As*. Likewise, to load a previously-saved configuration, go to the *System* → *Configuration* menu and select *Load*. Finally, you can export the configuration to a text file by selecting *Export* from that same menu.

Bibliography

Skempton, A. W. (1954), "The pore-pressure coefficients A and B." *Geotechnique*, 4(4), 143–147.

Schnabel, P., Seed, H. B., and Lysmer, J. (1972), "Modification of seismograph records for effects of local soil conditions." *Bulletin of the Seismological Society of America*, 62(6), 1649—-1664, URL http://www.bssaonline.org/content/62/6/1649.short.

Vaid, Y. P. and Negussey, D. (1984), "Relative density of pluviated sand samples." *Soils and Foundations*, 24(2), 101–105.

Vaid, Y. P. and Negussey, D. (1988), "Preparation of reconstituted sand specimens." *Advanced triaxial testing of soil and rock, ASTM STP*, 977, 405–417.

Wu, J., Kammerer, A. M., Riemer, M. F., Seed, R. B., and Pestana, J. M. (2004), "Laboratory study of liquefaction triggering criteria." In "13th World Conference on Earthquake Engineering, Vancouver, BC, Canada, Paper," 1–1.

Bradshaw, A. S. and Baxter, C. D. P. (2007), "Sample preparation of silts for liquefaction testing." *Geotechnical Testing Journal*, 30(4), 324.

Cubrinovski, M., Bradley, B., Wotherspoon, L., Green, R., Bray, J., Wood, C., Pender, M., Allen, J., Bradshaw, A., and Rix, G. (2011), "Geotechnical aspects of the 22 february 2011 christchurch earthquake." *Bulletin of the New Zealand Society of Earthquake Engineering*, 44(4), 205–226, URL http://ir.canterbury.ac.nz/handle/10092/6393.

Wang, S., Luna, R., and Stephenson, R. W. (2011), "A slurry consolidation approach to reconstitute low-plasticity silt specimens for laboratory triaxial testing." *Geotechnical Testing Journal*, 34(4), 1.

Appendix I

Conference Paper: Verification of ShakeVT2

Tenth U.S. National Conference on Earthquake Engineering
Frontiers of Earthquake Engineering
July 21-25, 2014
Anchorage, Alaska

COMPARISON OF EQUIVALENT-LINEAR SITE RESPONSE ANALYSIS SOFTWARE

S.J. Lasley[1], R.A. Green[2] and A. Rodriguez-Marek[3]

ABSTRACT

The objective of the study presented herein is to compare equivalent-linear site response analysis software. Ground motions recorded at soft soil sites can be drastically different from those recorded at rock sites. Site-specific response analyses provide an important way to predict surface ground motions from bedrock ground motions. Of all methods of site response analysis, the equivalent-linear method is, perhaps, the most popular. It has become an important method of site response analysis since first proposed by Schnabel et al. (1972) and has been shown to give a good approximation for many scenarios. In the years since the algorithm was released, several software programs that implement the equivalent-linear procedure have been written and made available to the engineering community. As part of ongoing research at Virginia Tech, a new equivalent-linear site response program, ShakeVT2, has been written in the Python programming language. In an effort to verify the implementation of the equivalent linear algorithm, extensive comparisons were made between ShakeVT2 and other equivalent-linear codes (i.e., SHAKE91, SHAKEVT, Strata, and DEEPSOIL). It was found that these implementations give similar results when supplied the same inputs. However, the choice of complex shear modulus and effective strain ratio have an impact on the results, especially for motions that contain a lot of energy at high frequencies. Additionally, it has been shown that when using DEEPSOIL, SHAKEVT, or SHAKE91 the discretization of the profile may obscure peaks when plotting the maximum shear strain, shear stress, or acceleration with depth. Finally, SHAKE91 and SHAKEVT suffer from a bug that gives incorrect results when the input motion file has an odd number of columns.

[1]Graduate Student Researcher, Charles E. Via, Jr. Dept. of Civil Engineering, Virginia Tech, Blacksburg, VA 24060
[2]Professor, Charles E. Via, Jr. Dept. of Civil Engineering, Virginia Tech, Blacksburg, VA 24060
[3]Associate Professor, Charles E. Via, Jr. Dept. of Civil Engineering, Virginia Tech, Blacksburg, VA 24060

Lasley SJ, Green RA, Rodriguez-Marek A. Comparison of Equivalent-Linear Site Response Analysis Software. *Proceedings of the 10th National Conference in Earthquake Engineering*, Earthquake Engineering Research Institute, Anchorage, AK, 2014.

DOI: 10.4231/D3P26Q42R

Comparison of Equivalent-Linear Site Response Analysis Software

S.J. Lasley[1], R.A. Green[2] and A. Rodriguez-Marek[3]

ABSTRACT

The objective of the study presented herein is to compare equivalent-linear site response analysis software. Ground motions recorded at soft soil sites can be drastically different from those recorded at rock sites. Site-specific response analyses provide an important way to predict surface ground motions from bedrock ground motions. Of all methods of site response analysis, the equivalent-linear method is, perhaps, the most popular. It has become an important method of site response analysis since first proposed by Schnabel et al. (1972) and has been shown to give a good approximation for many scenarios. In the years since the algorithm was released, several software programs that implement the equivalent-linear procedure have been written and made available to the engineering community. As part of ongoing research at Virginia Tech, a new equivalent-linear site response program, ShakeVT2, has been written in the Python programming language. In an effort to verify the implementation of the equivalent linear algorithm, extensive comparisons were made between ShakeVT2 and other equivalent-linear codes (i.e., SHAKE91, SHAKEVT, Strata, and DEEPSOIL). It was found that these implementations give similar results when supplied the same inputs. However, the choice of complex shear modulus and effective strain ratio have an impact on the results, especially for motions that contain a lot of energy at high frequencies. Additionally, it has been shown that when using DEEPSOIL, SHAKEVT, or SHAKE91 the discretization of the profile may obscure peaks when plotting the maximum shear strain, shear stress, or acceleration with depth. Finally, SHAKE91 and SHAKEVT suffer from a bug that gives incorrect results when the input motion file has an odd number of columns.

Introduction

Soft soil profiles can greatly affect the amplitude of incoming earthquake motions. This fact was drastically demonstrated in various earthquakes, including the 1989 Loma Prieta earthquake [1]. Consequently, modern building codes (e.g. [2]) apply a factor to modify motions from rock profiles to those for soil profiles. Moreover, under some conditions, codes require that site-specific response analyses are performed. The equivalent-linear procedure is a popular method of site response analyses because, in part, it is easy to perform compared to other non-linear site response analyses.

Ongoing research at Virginia Tech on liquefaction assessment includes the calculation of dissipated energy from site response analyses of multiple soil profiles and earthquake motions. To aid the research, it was decided to re-implement the equivalent-linear algorithm [3][4] instead of using existing codes (e.g. SHAKE91, SHAKEVT, Strata, DEEPSOIL). This paper gives an

[1]Graduate Student Researcher, Charles E. Via, Jr. Dept. of Civil Engineering, Virginia Tech, Blacksburg, VA 24061
[2]Professor, Charles E. Via, Jr. Dept. of Civil Engineering, Virginia Tech, Blacksburg, VA 24061
[3]Associate Professor, Charles E. Via, Jr. Dept. of Civil Engineering, Virginia Tech, Blacksburg, VA 24061

Lasley SJ, Green RA, Rodriguez-Marek A. Comparison of Equivalent-Linear Site Response Analysis Software. *Proceedings of the 10th National Conference in Earthquake Engineering*, Earthquake Engineering Research Institute, Anchorage, AK, 2014.

overview and comparison of the existing implementations of the equivalent linear algorithm.

Equivalent-Linear Site Response Analysis

The 1-D equivalent-linear site response analysis first introduced by Schnabel et al. in 1972 [3][4] is a well-known and well-used method. It is used to estimate the transformation of earthquake motions as they propagate upward through a soil profile. It assumes the vertical propagation of shear waves from a uniform half-space through horizontal layers of a soil profile modeled as visco-elastic material having a constant damping ratio across all frequencies [5]. These calculations are performed in the frequency domain, greatly increasing the speed and numerical stability of the calculations. However, in order to model the non-linear response of soil in the frequency domain, an iterative procedure is required.

For each layer in a soil profile, the equivalent-linear procedure approximates the non-linear stress-strain behavior using shear modulus reduction and damping curves. Fig. 1 shows examples of these curves for a range of overburden pressures. An iterative procedure is used to determine the degraded soil properties due to strain induced nonlinearities. For the first iteration of the algorithm, the response of the soil profile is calculated using small-strain values of shear modulus and damping.

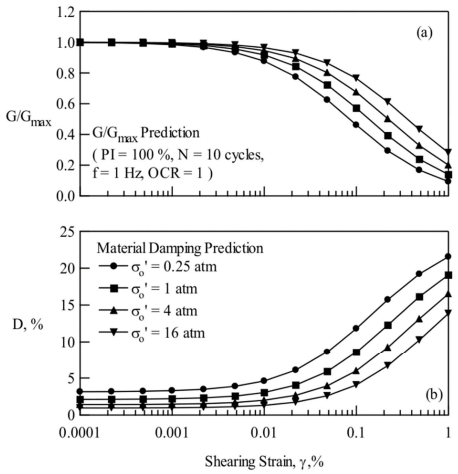


Figure 1. Shear modulus and damping degradation curves [6].

Using the initial shear modulus and damping values, a shear strain time history response is calculated, wherein the soil properties are held constant from the beginning to the end of the earthquake motion. From this time history response of each layer, a representative shear strain, γ_{eff}, is chosen. This representative strain is used to determine the degraded values of shear modulus and damping that are used in the next iteration of the analysis (see Fig. 1), and the process repeats until modulus and damping values reasonably converge to the specified degradation curves. Historically, various methods and values have been used to calculate the effective shear strain (γ_{eff}). An effective strain value of 65% of the maximum shear strain is most commonly used today.

Implementation of the Equivalent-Linear Algorithm

Several sources were consulted ([3][4][5][7][8][9][10][11][12]) in order to code the core of the equivalent-linear algorithm. As mentioned, the equivalent-linear algorithm requires iteration until the assumed shear modulus and damping from the beginning of the iteration agree with the shear modulus and damping from the calculated effective strain. Both Kramer [7] and the EERA manual [8] give the equation for the time-domain shear strain within a layer for a harmonic motion. However, the frequency-domain equation for shear strain in a layer for non-harmonic motions is not explicitly defined in these references. To clarify, the equation for strain in the frequency domain is given by:

$$\gamma_j(z_j, \omega) = ik_j^*\{A_j(\omega)\exp(ik_j^* z_j) - B_j(\omega)\exp(-ik_j^* z_j)\} \quad (1)$$

where z_j is the depth from the top of layer *j*; ω is an array of discrete angular frequencies defined by the Fourier transform and of length N_{FFT}; $A_j(\omega)$ and $B_j(\omega)$ are arrays of amplitudes of the up-going and down-going harmonic shear waves of layer *j*; i is the complex number; and k_j^* is the complex wave number of the layer ($\omega\sqrt{\rho/G^*}$). Similarly, the frequency-domain shear stress in a layer is given by:

$$\tau_j(z_j, \omega) = ik_j^* G_j^*\{A_j(\omega)\exp(ik_j^* z_j) - B_j(\omega)\exp(-ik_j^* z_j)\} \quad (2)$$

where G_j^* is the complex shear modulus of the layer. Regarding Eq. 2, some software may use G, the shear modulus of the layer, instead of G^*, the complex shear modulus. When this is done, the stress-strain curve (in time-domain) becomes a straight line with a slope of G instead of a series of hysteresis loops. Using G in lieu of G^* is inconsequential for computing acceleration time histories and response spectra, but use of G^* is essential for computing dissipated energy, which is was the motivation for this study.

Comparison of Existing Software

SHAKE91

The original SHAKE program, written by Per B. Schnabel et al. 1972 [13], was the first well-known equivalent-linear code. It has undergone several updates, and SHAKE91 [5] is a direct descendant of the original. SHAKE91 is the de facto standard equivalent-linear code, and several subsequent codes either directly use its code or are based on it. SHAKEVT [14] added the option to output dissipated energies for each layer and increased the maximum length of the input motion, among other modifications. SHAKE2000 is a pre- and post-processor for SHAKE91 [15]. WESHAKE is a modification and expansion of the SHAKE program [16]. Although the EduSHAKE and ProSHAKE programs are a rewrite of the SHAKE code, they are based on SHAKE, as well [17].

SHAKE91 continues to be a standard because it is such a simple program. Its input and output are simple text files and its executable is a slight 500 kB. Where EduSHAKE and ProSHAKE require an XP emulator to run in Windows 7 (or newer versions), SHAKE91's

Fortran executable still runs on most platforms without problems. However, SHAKE91 lacks a graphical user interface, a batch mode, and a catalog of degradation curves. It also has a few limitations that additionally hamper its utility. For example, the recommended limit to the number of points in the input motion is 3800. This was fine when most motions were discretized at intervals of 0.01 seconds. However, many motions are now discretized at 0.005 second intervals, and thus, SHAKE91 will not run with these motions. Additionally, a maximum of 13 distinct shear modulus and damping degradation curves can be used, limiting the discretization of the soil profile to some extent.

Additionally, SHAKE91 has an implementation bug: if the input motion has an odd number of columns, the motion is interpreted incorrectly by the program [18]. For example, if the input motion has 5 columns (the standard PEER database format), every sixth data point of the ground motion is interpreted as a zero by SHAKE91. This causes a frequency shift in the motion and cuts off the tail of the motion. This is caused by the way SHAKE91 stores the input motion (line 171, MAIN.FOR). In an apparent effort to save memory, SHAKE91 splits the real input motion and stores it in the real and imaginary parts of a variable. For odd-columned input motion formats, the last data point of the column goes into the real part and zero is assumed for the imaginary part.

DEEPSOIL

DEEPSOIL was first developed in 1998 under the direction of Youssef M.A. Hashash at the University of Illinois at Urbana-Champaign [12]. The program has several features in addition to those of SHAKE91: it has a graphical user interface, it contains a catalog of shear modulus and damping degradation curves, it can output data and plots to Microsoft Excel format, and it has a batch mode for handling multiple input motions. Additionally, both time domain non-linear and frequency domain equivalent-linear analyses are supported; only the frequency domain equivalent-linear analyses were evaluated herein. One minor issue was encountered: the output shear stress time history is calculated using shear modulus (i.e., G) instead of the complex shear modulus (i.e., G^*); thus, it cannot be used to calculate dissipated energy without additional post-processing.

Strata

Strata was written by Albert Kottke and Ellen M. Rathje at the University of Texas at Austin [10]. It has a graphical user interface, an extensive and customizable catalog of shear modulus and damping degradation curves, an option to use Random Vibration Theory (RVT), and options to randomize the profile properties. It also has an extensive set of output options (including dissipated energy), outputs to universal comma-separated value (.csv) format, and multiple input motion batch support. The source code of Strata is completely open and can be compiled on virtual any desktop operating system. (Windows binaries are available precompiled.) In contrast with other implementations of the equivalent-linear algorithm, Strata has an option to auto-discretize the layers of a soil profile, splitting each soil type into smaller sub-layers for analysis. For the comparisons in this paper, auto-discretization of the layers was turned off. Strata is currently under development, but the most recent version (399) is stable enough for most analyses.

ShakeVT2

ShakeVT2 was written in the Python language. Python is an object-oriented, interpreted language that excels at readability and portability of the code [19]. It was chosen for use in this research because it is easy to learn, free to use and distribute, and a large number of scientific libraries are readily available to extend the base language. At the time of this writing, a graphical user interface is not available for ShakeVT2. However, ShakeVT2 well performs the functions for which it was designed: calculating dissipated energy and the number of equivalent cycles for multiple profiles and motions using the low cycle fatigue implementation of the Palmgren-Miner hypothesis outlined in Green and Terri (2005) [20]. Because ShakeVT2 is written in a high-level language (compared to Strata's C++ or SHAKE91's Fortran), the core equivalent-linear algorithm of ShakeVT2 is much slower than those of Strata, DEEPSOIL, and SHAKE91. In most cases, however, the difference in the speed of the core algorithm is not an issue.

Comparison of Computation Results

In order to adequately compare the performance of the above-mentioned implementations of the equivalent-linear algorithm, a comparison of their results is presented herein. The Treasure Island E-W ground motion (TRI090) from the 1989 Loma Prieta earthquake is compared with the Yerba Buena Island E-W motion (YBI090) applied at the base of the Treasure Island soil profile (after [21]). Table 1 summarizes the profile.

Figure 2 shows the Fourier amplitude spectrum of the accelerations at the top of the profile after convergence of the equivalent-linear algorithms. The various algorithms show agreement within the accuracy of floating-point numbers for this profile and motion. Although not shown in this figure, discrepancies between SHAKE91/VT and the other software exist at amplitudes below about 10e-4 g/s. This is due to SHAKE's lack of precision.

Table 1. Treasure Island soil profile used in the analysis (after [21]).

Material	Thickness (m)	Unit Weight (kN/m^3)	Shear Wave Velocity (m/s)
Sand (Fill)	2.9	19.7	270
Sand (Fill)	5.5	19.7	142.4
Sand (Fill)	5	19.7	175
Young Bay Mud	8.1	16.6	179
Young Bay Mud	8.2	16.6	179
Sand	5.2	20.4	320
Sand	6.6	20.4	320
Old Bay Mud	17.6	19.5	270
Old Bay Mud	17.6	19.5	270
Old Bay Mud	12.5	19.5	390
Weathered Rock	11.9	21	660
Bedrock	-	22	1072

In this and the subsequent plots, SHAKE91 and SHAKEVT are plotted as the same line

since they gave identical results for motions with less than 4096 data points. And, because the Yerba Buena Island motion had more than 4096 points, SHAKE91 could not be used and it is assumed that SHAKE91 and SHAKEVT would yield the same results. There are several different normalization formats commonly used for Fourier amplitude spectrum (FAS). The FAS format used by SHAKE91/VT is different from that used by Strata and DEEPSOIL. As a result, to compare the FAS computed by the different codes, the SHAKE91/VT FAS was normalized using Eq. 3:

$$FAS_{Strata} = FAS_{DEEPSOIL} = 0.5 \cdot N_{FFT} \cdot \Delta t \cdot FAS_{SHAKE91} \tag{3}$$

where N_{FFT} is the number of points used in the FFT, and Δt is the time step of the input motion.

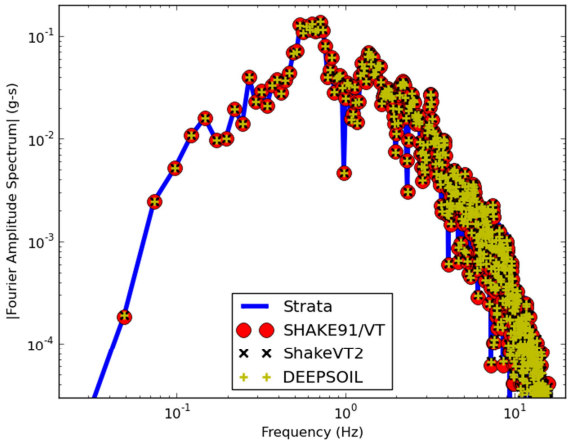


Figure 2. Comparison of the calculated Fourier amplitude spectra. Note that Strata smooths the FAS but that functionality was removed to aid in comparison.

The maximum shear strain profiles compute by the various codes differed (see Figure 3). By requesting the maximum shear strain at 1 meter depth intervals, ShakeVT2 shows that the peak value of shear strain occurs at approximately 15 meters. In comparison, Strata also outputs the maximum shear strains at relatively small depth increments but the output values are not as high as those of ShakeVT2. However, this disparity relates to how Strata outputs its maximum shear strain profile and is not due to differences in the algorithms used to compute the maximum shear strains at a given depth. Strata calculates the maximum shear strains at the center of each layer and interpolates between these depths in plotting the shear strain profile. However, the shear strain time histories computed by ShakeVT2 and Strata at a given depth are identical. SHAKE91/VT and DEEPSOIL also output the maximum shear strains at only at one depth per layer, at the top and center of each layer, respectively. As a result, the soil profile would need to be discretized by more/thinner layers in order to improve the accuracy of the computed strain profiles.

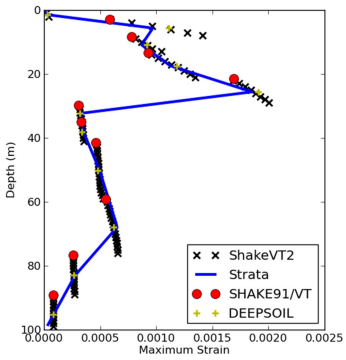


Figure 3. Maximum shear strain profile for the YBI090 motion.

Comparing the Effects of the Method to Calculate Response Spectrum

Several methods are available to calculate the response spectra from an acceleration time history. DEEPSOIL [10] allows the user to select from one of three methods to compute response spectra: 'Frequency Domain', 'Duhamel Integral', or 'Newmark'. These respectively refer to a method that uses the FAS to calculate the response spectra, Nigam and Jennings' piecewise-exact method [22], and the Newmark β method. Using the same soil profile and motion, the 5% damping response spectra at the surface of the profile was computed for each option. Fig. 4 shows that for this profile and ground motion, the choice of method to calculate response spectra makes little difference.

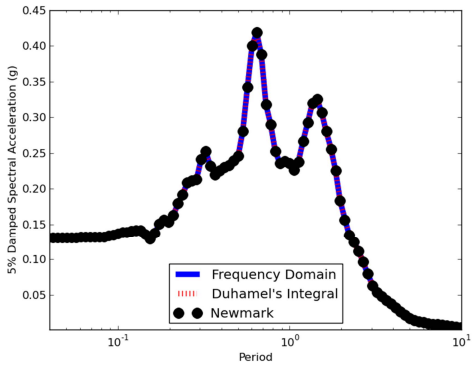


Figure 4. Comparison of methods to compute the response spectrum.

Comparing the Effects of the Complex Shear Modulus

Historically, several different equations have been used to calculate the complex shear modulus, G^*. SHAKE91/VT (lines 421-422 of B1.FOR) calculates the complex shear modulus of a layer using Eq. 4:

$$G^* = G\left(1 - 2\xi^2 + i2\xi\sqrt{1-\xi^2}\right) \quad (4)$$

where G is the shear modulus (τ/γ) of the layer and ξ is the viscous damping ratio of the layer (as a decimal). Strata [9] uses the complex shear modulus calculated from Eq. 5. Both DEEPSOIL and ShakeVT2 allow the user to choose Eqs. 4, 5, or 6, but default to Eq. 6. All equations give similar results when damping is small.

$$G^* = G(1 - \xi^2 + i2\xi) \quad (5)$$

$$G^* = G(1 + i2\xi) \quad (6)$$

Fig. 5 illustrates how the choice of complex shear modulus affects the FAS at the surface and the maximum acceleration profile. The FASs are very similar below a frequency of about 0.3 Hz, but the values diverge above that frequency. This translates into a difference in maximum acceleration at several depths in this soil profile.

Comparison of Effective Strain Ratios

All of the software mentioned in this paper allow the user to choose a ratio of effective shear strain to the maximum shear strain. The default value in Strata, DEEPSOIL, and ShakeVT2 is 0.65. The SHAKE91 manual [5] suggests using values between 0.4 and 0.75 "depending on the input motion and which magnitude earthquake it is intended to represent." Using ShakeVT2 and values of 0.5, 0.65, and 0.8 for the effective strain ratio, the response of the soil profile was calculated. Fig. 6 shows the resulting Fourier amplitude spectra and maximum acceleration profiles. Once again, the Fourier amplitude spectra diverge most at high frequencies and the maximum acceleration profiles are significantly different at several depths. This is not a surprise because effective strain ratio directly affects the stiffness and damping of a profile, and, thus, its capacity to amplify the incoming ground motion.

Conclusions

The equivalent-linear procedure, since its introduction with SHAKE in 1972, continues to be a useful method to estimate site response. Five implementations of the equivalent-linear algorithm have been compared for a single profile and ground motion. It has been shown that these implementations give similar results when supplied the same inputs. However, the choice of complex shear modulus and effective strain ratio have an impact on the results, especially for motions that contain a lot of energy at high frequencies. Additionally, it has been shown that when using DEEPSOIL, SHAKEVT, or SHAKE91 the discretization of the profile may obscure peaks when plotting the maximum shear strain with depth. Finally, SHAKE91 and SHAKEVT suffer from an implementation bug that gives incorrect results when the input motion file has an odd number of columns.

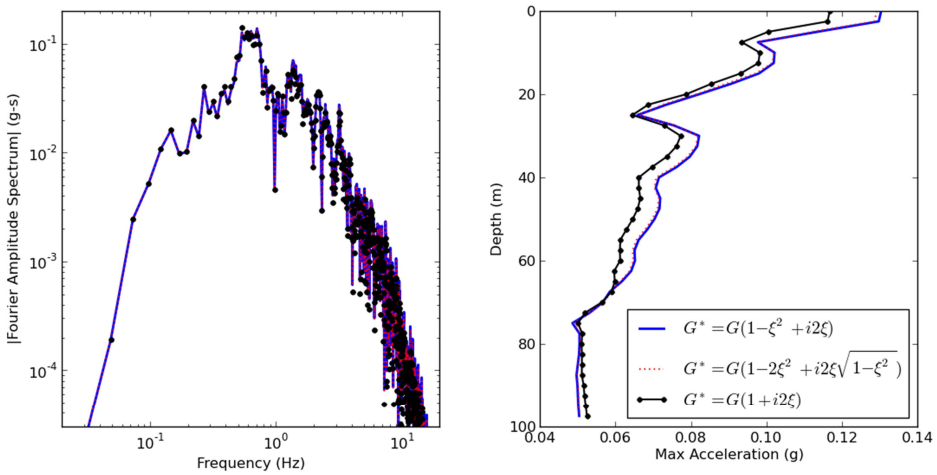


Figure 5. Differences in the Fourier amplitude spectrum and the max acceleration profile caused by using different equations for shear modulus.

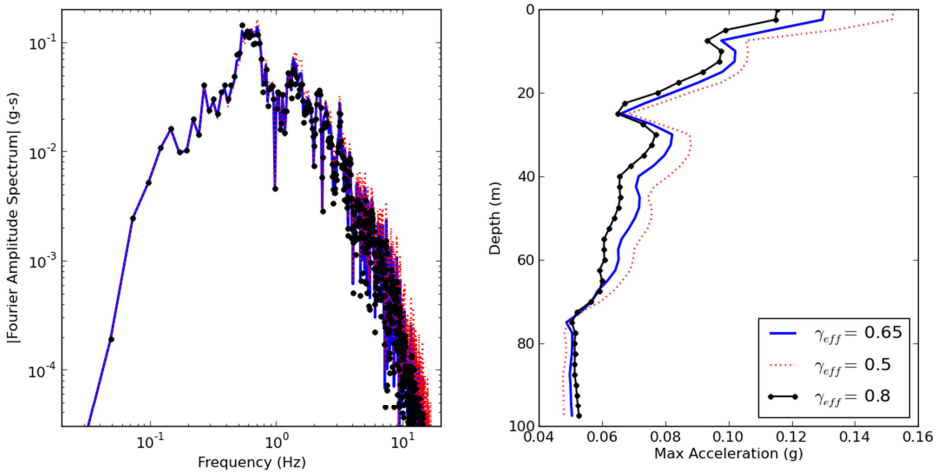


Figure 6. Fourier amplitude spectra and maximum acceleration profiles as affected by the effective shear strain ratio.

Acknowledgments

This research is partially funded by National Science Foundation (NSF) grants CMMI-1030564 and CMMI-1306261. This support is gratefully acknowledged. However, any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do

not necessarily reflect the views of the National Science Foundation.

References

1. Dickenson SE, Seed RB, Lysmer J, Mok CM. Response of Soft Soils During the 1989 Loma Prieta Earthquake and Implications for Seismic Design Criteria. *Proceedings of the Pacific Conference on Earthquake Engineering,* November 20-23, 1991, Auckland, New Zealand.
2. Minimum Design Loads for Buildings and Other Structures. The American Society of Civil Engineers 2005, ASCE/SEI 7-05.
3. Schnabel P, Seed HB, Lysmer J. Modification of Seismograph Records for Effects of Local Soil Conditions. Bulletin of the Seismological Society of America 1972; 62 (6): 1649-1664.
4. Schnabel P, Seed HB, Lysmer J. Erratum: Modification of Seismograph Records for Effects of Local Soil Conditions. *Bulletin of the Seismological Society of America* 1973; **63** (2): 750.
5. Idriss IM, Sun JI. User's Manual for SHAKE91. Center for Geotechnical Modeling, Dept. of Civil & Environmental Engineering, Univ. of California, Davis, California 1992.
6. Darendeli MB, Stokoe KH. Development of a New Family of Normalized Modulus and Material Damping Curves. Geotechnical Engineering Report GD01-1. University of Texas at Austin 2001.
7. Kramer SL. *Geotechnical Earthquake Engineering.* 1996. Prentice Hall, Upper Saddle River, New Jersey.
8. Bardet JP, Ichii K, Lin CH. EERA: A Computer Program for Equivalent-linear Earthquake Site Response Analyses of Layered Soil Deposits. Dept. of Civil Engineering, Univ. of Southern California 2000.
9. Nie J, Xu J, Costantino C. P-CARES: Probabilistic Computer Analysis for Rapid Evaluation of Structures. Brookhaven National Laboratory, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission 2007. NUREG/CR-6922, BNL-NUREG-77338-2006.
10. Kottke AR, Rathje EM. Technical Manual for Strata. PEER Report 2008/10. University of California, Berkeley, California.
11. Rathje EM, Kottke A. "Strata." https://nees.org/resources/strata 2013.
12. Hashash YMA. DEEPSOIL: User Manual and Tutorial. Version 5. Dept. of Civil and Environmental Engineering, Univ. of Illinois at Urbana-Champaign. 2011.
13. Schnabel PB, Lysmer J, Seed HB. SHAKE: A Computer Program for Earthquake Response Analysis of Horizontally Layered Sites. College of Engineering, University of California, Berkeley, California, 1972. Report No. EERC 72-12.
14. Green, RA. Energy-Based Evaluation and Remediation of Liquefiable Soils, Ph.D. Dissertation, Department of Civil and Environmental Engineering, Virginia Polytechnic Institute and State University (Virginia Tech), Blacksburg, VA, 397pp
15. Ordóñez, GA. SHAKE2000: A Computer Program for the 1-D Analysis of Geotechnical Earthquake Engineering Problems, User's Manual. July 2011 Revision.
16. Yule DE, Wahl RE, Wallace DC. WESHAKE6 Manual. http://www.itk.ilstu.edu/faculty/dcwalla/manual6.htm. Accessed November 13, 2013.
17. ProShake: Ground Response Analysis Program, User's Manual. Version 1.1. EduPro Civil Systems, Inc. Redmond, Washington.
18. Ordóñez, GA. *Personal communication.* March 27, 2013.
19. About Python. http://www.python.org/about/. Accessed November 13, 2013.
20. Green, R.A. and Terri, G.A. (2005). "Number of Equivalent Cycles Concept for Liquefaction Evaluations - Revisited", *Journal of Geotechnical and Geoenvironmental Engineering*, ASCE, 131(4), 477-488.
21. Gibbs JF, Fumal TE, Boore DM, Joyner WB. Seismic Velocities and Geologic Logs from Borehole Measurements at Seven Strong Motion Stations that Recorded the Loma Prieta Earthquake. US Geological Survey 1992. OFR 92-287.
22. Nigam NC, Jennings PC. Calculation of Response Spectra from Strong-motion Earthquake Records. *Bulletin of the Seismological Society of America* 1969; **59** (2) 909-922.

Appendix J

Code

J.1 License

J.2 Equivalent-Linear: ShakeVT2

The file `EquivLin.py` performs equivalent-linear site response analysis and calculates the number of cycles based on dissipated energy, n_{eq}.

```
'
try:
    import ipdb as pdb
except:
    import pdb
import csv
import numpy as np
#import numexpr as ne
from scipy.interpolate import interp1d
#import guiqwt.pyplot as plt
import matplotlib.pyplot as plt
import degrcurv as dc

#import wxfiledialog as fd
#import qtfiledialog as fd
#import cPickle

try:
    xrange
except NameError:
    xrange = range

class SoilProfile(object):
    '''
    This object acts as a container for the Soil Profile and subsequent
        calculations.

    Example:

    SoilProfileObject = SoilProfile(profile_name, Unit_Wt, Vs, t, **kwargs)

    The functions 'inputread' and 'input_soil_profile' automatically call this
    object. Alternately, the required values can be given manually.

        Inputs:
            profile_name = a name for the profile
            Unit_wt = an array of unit weights of the layers of the profile.
            Vs = an array of shear wave velocities of the layers of the profile.
            t = tan array of the thicknesses of each layer of the profile.

        Keyword Arguments:        Definition/Default:
            filename       filename of the input profile

            ko             The ratio of vertical over horizontal shear
                              stresses (array). This is used to calculate
                              the degradation curves.

            phi            The friction angle (array). Also used to calculate
                              degradation curves. Used if the ko array
                              is zero.
```

```
50
51            N160           Corrected SPT blowcounts (array). Only used if ko
52                             and phi are zero. If an N160 value is not
53                             given, ko is assumed to be 0.5.
54
55            gwt            The depth of the ground water table. If not given,
56                             it is assumed to be at the surface.
57
58            g              Acceleration due to gravity.
59                             Assumed to be 9.81 m/s^2.
60
61            layerno        An array of integers numbering the layers.
62                             Assumed to be counted from the surface
63                             starting at 1.
64
65          layername        An array of names describing the layers.
66          scalefactor      A factor by which ground motions can be amplified
67                             or deamplified. Assumed to be 1.
68
69          strainratio      Ratio of equivalent uniform strain divided by the
70                             maximum strain. Default is 0.65. This value is
71                             the default when calling the
72                             Dynamics.iterations method.
73
74          loc_mot          Layer at which the motion is input into the
75                             profile, given as an integer. When using this
76                             value, the Dynamics._calcWaves method counts the
77                             layers from the surface, starting at 1,
78                             irrespective of the layerno array values.
79                             Default is the bottom layer (bedrock).
80
81          outcrop_mot      0 signifies that the input motion is an outcrop
82                             motion, 1 otherwise. This can be overridden by
83                             the Dynamics object call if the 'outcrop_mot'
84                             keyword is used.
85          OCR              An array giving the Over Consolidation Ratio for
86                             each layer. This is used only for the
87                             Darendeli and Stokoe (2001) Degradation Curves.
88
89          DS_soil          Soil number for Darendeli and Stokoe Degrad Curves
90                             The soil type indicator:
91                                 0 for a general set of curves
92                                   (Table 8.12 from Darendeli's Dissertation)
93                                1 for clean sand
94                                 2 for sands with high fines contents
95                                 3 for silts
96                             4 for clays
97
98          DS_N             Number of cycles for the Darendeli and
99                             Stokoe (2001) Degradation Curves.
100                            Default is 10 for all layers.
101
102         DS_freq          Frequency for the Darendeli and
103                            Stokoe (2001) Degradation Curves.
104                            Default is 1 Hz for all layers.
105   ====================== NOT IMPLEMENTED ===================================
106         ACCTHo6          an array indicating if the acceleration time history should be
```

```
107                             output for the center of each layer. 1 will output, 0 will
                                    suppress output.
108           AccTHoutcrop6   Specifies if the motion from the above is an outcrop motion (0)
                  or within (1)[default]
109           AccTHotype6     Specifies if the above timehistories should be the entire time
                  history (1),
110                             or the max value only (0)[default]
111           TauTH7a         An array specifying if the stress time histories of the
                  corresponding layers should be output.
112                             1 will output, 0 will suppress output.
113           GamTH7b         An array specifying if the stress time histories of the
                  corresponding layers should be output.
114                             1 will output, 0 will suppress output.
115           RespSpec9       An array specifying if the response spectrum should be calculated
                  and output:
116                             0 for no [default], 1 for outcrop, 2 for within
117
118
119           There should be additional options that I need to
                  code.<<<+=============================================================
120
121
122       The following 'data attributes' or variables are created by __init__:
123           self.t
124           self.unit_wt
125           self.gwt
126           self.profile_name
127           self.ko
128           self.phi
129           self.PI
130           self.N160
131           self.g
132           self.layerno
133           self.layername
134           self.scalefactor
135           self.strainratio
136           self.loc_mot
137           self.outcrop_mot
138           self.filename
139           self.OCR
140
141           Some of these may equal 'None'.
142       '''
143       def __init__(self, profile_name, unit_wt, Vs, t, **kwargs):
144           self.profile_name = profile_name
145           self.unit_wt = unit_wt
146           self.Vs = Vs
147           self.t = t
148           self.ko = kwargs.get('ko', np.zeros(len(t),))
149           self.phi = kwargs.get('phi', np.zeros(len(t),))
150           self.PI = kwargs.get('PI', np.zeros(len(t),))
151           self.N160 = kwargs.get('N160', np.zeros(len(t),))
152           self.gwt = kwargs.get('gwt', 0)
153           self.g = kwargs.get('g', 9.81)
154           self.layerno = kwargs.get('layerno', np.arange(len(t),)+1)
155           self.layername = kwargs.get('layername')
156           self.scalefactor = kwargs.get('scalefactor',1.)
```

```
157         self.strainratio = kwargs.get('strainratio',0.65)
158         self.loc_mot = kwargs.get('loc_mot',0)
159         self.outcrop_mot = kwargs.get('outcrop_mot',0)
160 #        self.AccTHo6 = kwargs.get('AccTHo6',np.zeros(len(t),))
161 #        self.AccTHoutcrop6 = kwargs.get('AccTHoutcrop6',np.ones(len(t),))
162 #        self.AccTHotype6 = kwargs.get('AccTHotype6',np.zeros(len(t),))
163 #        self.TauTH7a = kwargs.get('TauTH7a',np.zeros(len(t),))
164 #        self.GamTH7b = kwargs.get('GamTH7b',np.zeros(len(t),))
165 #        self.RespSpec9 = kwargs.get('RespSpec9',np.zeros(len(t),))
166         self.OCR = kwargs.get('OCR',None)
167         self.DS_soil = kwargs.get('DS_soil', None)
168         self.DS_N = kwargs.get('DS_N', None)
169         self.DS_freq = kwargs.get('DS_freq', None)
170         self.filename = kwargs.get('filename')
171 
172         self._calcInsituStress()
173 
174 
175     def _calcInsituStress(self):
176         '''
177         Calculates the mid-depth of each layer, the vertical stress,
178         the effective stresses, and the mean effective stresses.
179 
180         This method is automatically called by the soil profile
181         object __init__. For an external call, use SoilProfile.calcInsituStress.
182 
183         Notice that if ko is not given, it is calculated from phi. If
184         phi is not given, it is calculated from N160. If none of ko, phi, or
185         N160 are given, the ko value is assumed to be 0.5. This is important
186         because ko is used to calculate the mean effective stress which is
187         used, in some cases, to calculate the degradation curves.
188 
189         Each of the inputs should be an array with the [0] value being the
190         value of the layer at the surface:
191 
192             t = thickness of the layer
193 
194             unit_wt = the unit weight of the layer
195 
196             gwt = depth of the ground water table from the surface (positive value-
197                    not an array)
198 
199             ko =  at rest earth pressure coefficient
200 
201             phi = effective friction angle. Used to calc the ko if ko is given as
202                    zero.
203 
204             N160 = the overburden- and energy-corrected SPT blowcount. Used to
205                    calc the ko it ko and phi are zero.
206                    If N160, phi and ko are all zero for a layer, Ko is assumed to
207                    be 0.5
208 
209         The following 'data attributes' or variables are created:
210         self.t_mid
211         self.sigv
212         self.sigveff
213         self.sigmeff
```

```
        '''
        t = self.t
        unit_wt = self.unit_wt
        gwt = self.gwt
        ko, phi, N160 = self.ko, self.phi, self.N160
        t_mid = np.zeros(len(t))
        sigv = np.zeros(len(t))
        sigveff = np.zeros(len(t))
        for i in xrange(len(t_mid)):
            if i == 0:
                t_mid[i] = (t[i] / 2)
                sigv[i] = (t_mid[i] * unit_wt[i])
            else:
                t_mid[i] = (t_mid[i - 1] + t[i - 1] / 2 + t[i] / 2)
                sigv[i] = (sigv[i - 1] + t[i - 1] / 2 * unit_wt[i - 1] +
                           t[i] / 2 * unit_wt[i])
            if t_mid[i] > gwt:
                sigveff[i] = (sigv[i] - 9.81 * (t_mid[i] - gwt))
            else:
                sigveff[i] = (sigv[i])
            if ko[i] == 0:
                if phi[i] != 0:
                    ko[i] = 1 - np.sin(phi[i] * np.pi / 180.) # Using phi to calc ko
                elif N160[i] != 0:
                    phi[i] = (20 * N160[i]) ** 0.5 + 20 # Jaky, 1944, using N160 to calc phi
                         and ko
                    ko[i] = 1 - np.sin(phi[i] * np.pi / 180.)
                    # Hatanae and Uchida, 1996
                else:
                    ko[i] = 0.5 # making an assumption
        sigmeff = sigveff * (1. + 2. * ko[:len(sigveff)]) / 3.
        self.t_mid, self.sigv = t_mid, sigv
        self.sigveff, self.sigmeff = sigveff, sigmeff


    def calcInsituStress(self, *args, **kwargs):
        '''
        This small function calculates the stresses at new depths
        Positional Arguments:
        Depths at which the insitu stresses should be calculated.
        If specified, the keyword argument "depths" is ignored.
        If neither the positional or keyword arguments are used,
        the mid-depth of each layer is used.

        Keyword Arguments:
        out = 'total': the total vertical stress is output,
        Example:
            sigv = self.calcInsituStress([0.5,5.2], out='total')

        out = 'all':  the total, effective and mean effective stresses are output,
        Example:
             sigv, sigveff, sigmeff = self.calcInsituStress([0.5,5.2],
                                      out='all')

        out = 'eff':  the effective vertical stress is output.
        Example:
            sigveff = self.calcInsituStress([0.5,5.2], out='eff')
```

```
270
271         out = 'meff':  the mean effective vertical stress is output.
272         Example:
273             sigmeff = self.calcInsituStress([0.5,5.2], out='meff')
274         '''
275         out = kwargs.get('out', 'all')
276         gam_water =kwargs.get('gam_water', 9.81)
277         if len(args) > 0:
278             depths = np.array(args)
279         else:
280             depths = kwargs.get('depths',self.SoilObj.t_mid)
281         gwt = self.gwt
282         unit_wt = self.unit_wt
283         if np.isscalar(depths):
284             depths = np.array([depths], dtype=float)
285         try:
286             depths.ravel()
287         except AttributeError:
288             depths = np.array(depths, dtype=float)
289             depths.ravel()
290
291         h = np.append(self.t,1000)
292         hsum = np.cumsum(h, dtype=float)
293         sigv = np.zeros(len(depths))
294         buoy = np.zeros(len(depths))
295         ko = np.zeros(len(depths))
296
297         # Find the layer for each depth and calc the total stresses
298         j = 0
299         for i in xrange(len(depths)):
300             while depths[i] >= round(hsum[j],3):
301                 j += 1
302             if depths[i] < h[0]:
303                 d = depths[i]
304             else:
305                 d = depths[i] - hsum[j-1]
306
307             if depths[i] >= gwt:
308                 buoy[i] = (depths[i] - gwt) * gam_water
309             else:
310                 buoy[i] = 0
311
312             if j == 0:
313                 sigv[i] = unit_wt[j] * d
314             else:
315                 sigv[i] = np.sum(h[:j] * unit_wt[:j])+ unit_wt[j] * d
316             ko[i] = self.ko[j]
317
318         if out == 'total':
319             return sigv
320         elif out == 'all':
321             sigveff = sigv - buoy
322             sigmeff = sigveff * (1. + 2. * ko) / 3.
323             return sigv, sigveff, sigmeff
324         elif out == 'meff':
325             sigveff = sigv - buoy
326             sigmeff = sigveff * (1. + 2. * ko) / 3.
```

```
            return sigmeff
        else:
            sigveff = sigv - buoy
            return sigveff


    def calcVs30(self,**kwargs):
        '''
        Calculates the Vs30 of the profile according to eqn 20.4-1 of
        ASCE 7-10.


        Optional Keyword Argument:
            SiteClass = True or False [default].
                        If true, the site class according to ASCE 7-10 will
                        also be output with the Vs30.
                        If  False, the thickness of the entire soil profile
                        is returned instead.
            returns = True or False [default]. Whether or not anything is
                        returned, the following 'data attributes' or variables
                        are created:

                        self.Vs30
                        self.SiteClass

        Example:

            Vs30 = SoilObject.calcVs30(Vs_array, t_array, returns = True)
            Vs30,SiteClass = SoilObject.calcVs30(Vs_array,
                                                 t_array,
                                                 SiteClass=True,
                                                 returns = True)
        '''
        t = np.zeros(self.Vs.shape, dtype=float)
        t[:-1] = self.t
        Vsprofile = self.Vs
        returns = kwargs.get('returns',False)
        SiteClass = kwargs.get('SiteClass',False)

        n=-1
        if np.sum(t) <= 30:
            t[-1] = 30 - np.sum(t[:-1])
        else:

            t2 = np.zeros(t.shape,dtype=float)
            while np.sum(t2) < 30:
                n += 1
                t2[n] = t[n]

            if np.sum(t2) > 30:
                t2[n] = t2[n] - (np.sum(t2)-30)

            t = t2

        self.Vs30 = np.sum(t) / np.sum(t/Vsprofile)

        if SiteClass == True:
```

```
            if self.Vs30 > 1524:
                self.SiteClass = 'A'
            elif self.Vs30 >= 762:
                self.SiteClass = 'B'
            elif self.Vs30 >= 365.76:
                self.SiteClass = 'C'
            elif self.Vs30 >= 182.88:
                self.SiteClass = 'D'
            else:
                self.SiteClass = 'E'

            if returns == True:
                return self.Vs30, self.SiteClass
        else:
            if returns == True:
                return self.Vs30, np.sum(t)


    def calcAvgVs(self, depth):
        '''
        Calculates the average shear wave velocity of the top
        'depth' of the profile according to eqn 20.4-1 of
        ASCE 7-10.


        '''
        t = np.zeros(self.Vs.shape, dtype=float)
        t[:-1] = self.t
        Vsprofile = self.Vs
        n=-1
        if np.sum(t) <= depth:
            t[-1] = depth - np.sum(t[:-1])
        else:

            t2 = np.zeros(t.shape,dtype=float)
            while np.sum(t2) < depth:
                n += 1
                t2[n] = t[n]

            if np.sum(t2) > depth:
                t2[n] = t2[n] - (np.sum(t2)-depth)

            t = t2

        return np.sum(t) / np.sum(t/Vsprofile)


    def plot(self, *args, **kwargs):
        '''
        Outputs standard plots that illustrate the soil profile.

        Positional Arguments:

            List the plots you want to see.
            Possible plots:
                Vs
                unit_wt
```

```
            N160
            sigv
            sigmeff
            sigveff
            phi
            ko
            all

        Or, you can leave it blank and all will be output.

    Keywords Arguments:

        save = False (default) or True
            For now, if save = True, then the figures will also be closed.
            (Haven't figured it out yet.)

        extension = '.png' (default) or any other that matplotlib handles

        layerlabels = array of labels for each layer

        If layerlabels is specified, the following kwargs may be specified:
        labeldepths = depth at which the label should be placed
        labelxvals = x value at which the label is placed.

        Note that I have only implemented the labeling for the Vs plot.

    Example:

        self.plot('Vs','unit_wt', save=True, extension='.png')

        No data attributes are created with this method.
    '''
    if len(args) == 0:
        args = ('Vs','unit_wt','N160','sigv','sigmeff',
                    'sigveff','phi','ko')
    if 'all' in args:
        args = ('Vs','unit_wt','N160','sigv','sigmeff',
                    'sigveff','phi','ko')
    save = kwargs.get('save',False)
    ext = kwargs.get('extension','.png')
    lw = kwargs.get('lw',2)
    mark = kwargs.get('mark','-k')
    layerlabels = kwargs.get('layerlabels', False)
    if layerlabels:
        layername = kwargs.get('layername',self.layername)
        labeldepths = kwargs.get('labeldepths')
        labelxvals = kwargs.get('labelxvals')



    def profilecreator(array, **kwargs):
        '''
        Makes points at the top and bottom of each layer to make plotting
        better.
        '''
        depth = kwargs.get('depth',False)
        if depth == True:
```

```
            output = [0]
            for i in xrange(len(array)):
                if output[i] == 0:
                    output.append(array[i])
                else:
                    output.append(array[i-1])
                    output.append(array[i])
            output.append(array[-1])
            output.append(array[-1]*1.05)
        else:
            output = []
            for i in xrange(len(array)):
                output.append(array[i])
                output.append(array[i])
        return output

    def labeler(depths, labels, **kwargs):
        '''
        Adds labels to the plots.
        '''
        [xmin, xmax, ymin, ymax] = plt.axis()
        xvals = kwargs.get('xvals',xmin)

        for i,z in enumerate(depths):
            plt.annotate(labels[i], xy=(xvals[i],z))


    depths = profilecreator(np.cumsum(self.t),depth=True)
    t_mid = self.t_mid

    if 'unit_wt' in args:
        unit_wt = profilecreator(self.unit_wt)
        plt.figure('unit_wt')
        plt.plot(unit_wt,depths,'-b')
        plt.xlabel('Unit Weight, $\gamma$ $(kN/m^3)$')
        plt.ylabel('Depth ($m$)')
        plt.title(self.filename)
        [xmin, xmax, ymin, ymax] = plt.axis()
        plt.axis([xmin, xmax, ymax, ymin])

    if 'sigv' in args:
        sigv = self.sigv
        plt.figure('stress')
        plt.plot(sigv,t_mid,'-b',label='$\sigma_v$')
        plt.xlabel('Stress (kPa)')
        plt.ylabel('Depth ($m$)')
        plt.legend()
        plt.title(self.filename)
        [xmin, xmax, ymin, ymax] = plt.axis()
        plt.axis([xmin, xmax, ymax, ymin])

    if 'sigmeff' in args:
        sigmeff = self.sigmeff
        plt.figure('stress')
        plt.plot(sigmeff,t_mid,'-r',label='$\sigma \'_m$')
        plt.xlabel('Stress (kPa)')
        plt.ylabel('Depth ($m$)')
```

```
555             plt.legend()
556             plt.title(self.filename)
557             [xmin, xmax, ymin, ymax] = plt.axis()
558             plt.axis([xmin, xmax, ymax, ymin])
559 
560         if 'sigveff' in args:
561             sigveff = self.sigveff
562             plt.figure('stress')
563             plt.plot(sigveff,t_mid,'-g',label='$\sigma \'_v$')
564             plt.xlabel('Stress(kPa)')
565             plt.ylabel('Depth ($m$)')
566             plt.legend()
567             plt.title(self.filename)
568             [xmin, xmax, ymin, ymax] = plt.axis()
569             plt.axis([xmin, xmax, ymax, ymin])
570 
571         while 'phi' in args:
572             phi = profilecreator(self.phi)
573             if max(phi) == 0:
574                 break
575             plt.figure('phi')
576             plt.plot(phi,depths[:-2],'-b')
577             plt.xlabel('Friction Angle, $\phi$')
578             plt.ylabel('Depth ($m$)')
579             plt.title(self.filename)
580             [xmin, xmax, ymin, ymax] = plt.axis()
581             plt.axis([xmin, xmax, ymax, ymin])
582             break
583 
584         if 'Vs' in args:
585             Vs = profilecreator(self.Vs)
586             plt.figure('Vs')
587             plt.plot(Vs,depths,mark,lw=lw)
588             plt.xlabel('Shear Wave Velocity, $V_s (m/s)$')
589             plt.ylabel('Depth ($m$)')
590             plt.title(self.filename)
591             [xmin, xmax, ymin, ymax] = plt.axis()
592             plt.axis([xmin, xmax, ymax, ymin])
593 
594             if layerlabels:
595                 labeler(labeldepths, layername, xvals=labelxvals)
596 
597 
598         while 'N160' in args:
599             N160 = profilecreator(self.N160)
600             if max(N160) == 0:
601                 break
602             plt.figure('N160')
603             plt.plot(N160,depths[:-2],'-b')
604             plt.xlabel('Corrected SPT Blowcounts, $N_{1,60}$')
605             plt.ylabel('Depth ($m$)')
606             plt.title(self.filename)
607             [xmin, xmax, ymin, ymax] = plt.axis()
608             plt.axis([xmin, xmax, ymax, ymin])
609             break
610 
611         while 'ko' in args:
```

```
612             ko = profilecreator(self.ko)
613             if max(ko) == 0:
614                 break
615             plt.figure('ko')
616             plt.plot(ko,depths[:len(ko)],'-b')
617             plt.xlabel('Lateral Earth Pressure Coefficient, $K_0$')
618             plt.ylabel('Depth ($m$)')
619             plt.title(self.filename)
620             [xmin, xmax, ymin, ymax] = plt.axis()
621             plt.axis([xmin, xmax, ymax, ymin])
622             break
623         if save == True:
624             import matplotlib
625             figures=[manager.canvas.figure for manager in
626                         matplotlib._pylab_helpers.Gcf.get_all_fig_managers()]
627             for i, figure in enumerate(figures):
628                 figure.savefig(self.profile_name +'-'+figure._label+ext)
629 #                figure.clf()
630 
631 
632     def calcDepth(self, stress, **kwargs):
633         '''
634         This method will return the depth at which a stress is felt.
635 
636         Keyword arguments:    Meaning/Options:
637         stresstype               'total', 'eff'[default], 'mean'
638                                  corresponding to
639                                  the total vertical stress, the vert
640                                      effective stress, and the mean
641                                      effective stress, respectively.
642         Output:
643         depths = the depths at which the corresponding stresses are felt.
644                   No data attributes are created with this method.
645 
646         Example:
647 
648             depths = self.calcDepth([50., 101.], stresstype='eff')
649         '''
650         unit_wt = self.unit_wt
651         t = self.t
652         t_mid = self.t_mid
653         ko = self.ko
654         gwt = self.gwt
655         stresstype = kwargs.get('stresstype','eff')
656 
657         if np.isscalar(stress):
658             stress = np.array([stress], dtype=float)
659         try:
660             stress.ravel()
661         except AttributeError:
662             stress = np.array(stress, dtype=float)
663             stress.ravel()
664 
665         outdepths = np.zeros(stress.shape,dtype=float)
666 
667         if stresstype == 'total':
668             stressprof = self.sigv
```

```
            unit_wt_eff = unit_wt[:-1]
        elif stresstype == 'eff':
            stressprof = self.sigveff
            unit_wt_eff = np.where(np.cumsum(t) <= gwt, unit_wt[:-1],
                                   unit_wt[:-1]-9.81)
        elif stresstype == 'mean':
            stressprof = self.sigmeff
            unit_wt_eff = np.where(np.cumsum(t) <= gwt, unit_wt[:-1],
                                   unit_wt[:-1]-9.81) * (1 + 2 * ko[:-1]) / 3
        else:
            print('I didn\'t understand the stresstype')
            return np.ones(stress.shape,dtype=float) * -1000.

        for i,sig in enumerate(stress):
            try:
                no = np.where(sig < stressprof)[0][0]
            except IndexError:
                print('Stress of {} kPa not reached in this profile.'.format(
                        sig))
                outdepths[i] = -1000
                continue


            stressinterface = stressprof[no-1] + (unit_wt_eff[no-1] *
                        t[no-1] / 2)
            if stressinterface >= sig:
                outdepths[i] = t_mid[no-1] + ((sig - stressprof[no-1]) /
                        unit_wt_eff[no-1])
            else:
                outdepths[i] = np.sum(t[:no]) + ((sig -
                             stressinterface) / unit_wt_eff[no])

        return outdepths



    def getLayer(self, depth):
        '''
        Return the layer number for a given depth.
        Example:

            layernums = self.getLayer(10.25)

            layernums = self.getLayer([10.25, 13.7])

            No data attributes are created with this method.
        '''
        tsum = np.cumsum(self.t)
        if np.isscalar(depth):
            depth = np.array([depth], dtype=float)
        try:
            depth.ravel()
        except AttributeError:
            depth = np.array(depth, dtype=float)
            depth.ravel()
        layernumbers = np.ones(depth.shape, dtype=int) * -1
```

```
726
727         for i,z in enumerate(depth):
728             layernumbers[i] = np.where(z <= tsum)[0][0]
729
730         if len(layernumbers) == 1:
731             return layernumbers[0]
732         else:
733             return layernumbers
734
735
736     def to_csv(self, **kwargs):
737         '''
738         Writes out a .csv file from this profile.
739         '''
740         filename = kwargs.get('filename', self.profile_name)
741
742         f = open(filename, 'w')
743         f.write('Name of Profile, {}\n'.format(self.profile_name))
744         f.write('Units, 0\n')
745         f.write('GWT Depth, {}\n'.format(self.gwt))
746         f.write('Location of Motion, {}\n'.format(self.loc_mot))
747         f.write('Outcrop Motion,{},'.format(self.outcrop_mot)+
748             ',,,,,,,,|==== Darendeli and Stokoe Parameters ====|\n')
749         f.write('Layer number,name,Unit Weight,Vs,Thickness,PI,ko,phi,'+
750                 'N160,OCR,Soil Type,N,Frequency\n')
751         num = len(self.t)
752         strout1 = '{},'*13 + '\n'
753         strout2 = '{},' * 4 + '\n'
754
755         for i,(no, name, utwt, vs, t, pi, ko, phi, n160) in enumerate(zip(
756                 self.layerno[:-1], self.layername[:-1],
757                 self.unit_wt[:-1], self.Vs[:-1], self.t,
758                 self.PI, self.ko, self.phi, self.N160)):
759             ocr = self.OCR[i] if np.any(self.OCR) else ''
760             soiltype = self.DS_soil[i] if np.any(self.DS_soil) else ''
761             freq = self.DS_freq[i] if np.any(self.DS_freq) else ''
762             n = self.DS_N[i] if np.any(self.DS_N) else ''
763
764             f.write(strout1.format(no, name, utwt, vs, t, pi, ko, phi, n160,
765                 ocr, soiltype, n, freq))
766
767         f.write(strout2.format(self.layerno[-1], self.layername[-1],
768             self.unit_wt[-1], self.Vs[-1]))
769         f.close()
770
771 class GroundMotion(object):
772     '''
773     This object acts as a container for all ground motion related variables.
774
775     Keyword Arguments:       Definition/defaults:
776         name          The name of the ground motion/ground motion file.
777
778         filename      The filename of ground motion file. If not specified,
779                           name needs to be a valid filename in the path.
780
781         scalefactor   Factor by which the motion time history is scaled.
782                           This scaling occurs when the motion is read from
```

```
                        the file. If the motion is specified as a keyword
                        argument, no scaling occurs.

     Butrwrth      If True, a Butterworth filter is applied when the motion
                    is called from the file. If the motion is specified
                    as a keyword argument, the Butterworth filter is not
                    applied. Default is False

     b_order       The order of the Butterworth filter. Default is 4.

     cutofffreq    Cutoff frequency (in Hz) for the Butterworth filter.
                    Default is 100 Hz.

     g             Acceleration due to gravity. Default is 9.81. This is
                    used to convert the motion from units of g to m/s/s.
                    Apply the appropiate value for the units of the motion
                     and the soil profile. This code assumes everything
                     is in SI units unless specified.

==== The following are only needed if a valid filename is not given. ====
   motion         The input acceleration time history. If this is not
                      specified, a valid filename must be given.

   NFFT           The number of points used in the FFT. Only required if
                      the motion keyword argument is specified.

   FA             The Fourier Amplitude spectrum of the input motion.
                      Only required if the motion keyword argument is
                      specified.

   f               An array of frequencies corresponding to the FA in Hz.
                      Only required if the motion keyword argument is
                      specified.

   w              An array of angular frequencies corresponding to the FA
                      in rad/sec. Only required if the motion keyword
                      argument is specified.

   NPTS           Number of points in the input time history. Only
                      required if the motion keyword argument is specified.

   dt             The sample spacing of the time history input motion in
                      seconds. Only required if the motion keyword
                      argument is specified.



   dummy           Let dummy=True if you just want to calc the transfer function of the
        profile.


   'Data attributes' or variables created by the __init__ method:
   self.name
   self.filename
   self.scalefactor
   self.Butrwrth
```

```
        self.b_order (if self.Butrwrth)
        self.cutofffreq (if self.Butrwrth)
        self.g
        self.motion
        self.NFFT
        self.FA
        self.f
        self.w
        self.NPTS
        self.dt
        self.dummy
        self.time

    '''
    def __init__(self,**kwargs):
        self.name = kwargs.get('name')
        self.filename = kwargs.get('filename')
        formatin = kwargs.get('formatin','PEER')
        self.scalefactor = kwargs.get('scalefactor',1.)
        self.Butrwrth = kwargs.get('Butrwrth',False)
        if self.Butrwrth:
            self.b_order = kwargs.get('b_order',4.)
            self.cutofffreq = kwargs.get('cutofffreq',100)
        self.g = kwargs.get('g',9.81)
        self.motion = kwargs.get('motion')
        self.NFFT = kwargs.get('NFFT')
        self.FA = kwargs.get('FA')
        self.f = kwargs.get('f')
        self.w = kwargs.get('w')
        self.NPTS = kwargs.get('NPTS')
        self.dt = kwargs.get('dt')
        self.dummy = kwargs.get('dummy',False)
        if self.NPTS and self.dt:
            self.time = np.linspace(0,(self.NPTS-1) * self.dt,self.NPTS)
        if self.dummy == True:
            self.name = 'dummy'
            self.motion = np.ones([8192,],dtype=float) * -1.
            self.NPTS = 8192
            if self.NFFT == None:
                self.NFFT = 8192
            if self.dt == None:
                self.dt = 0.01
            self.f = np.linspace(0, 1/self.dt/2, self.NFFT//2+1)
            self.w = self.f[:self.NFFT/2+1] * 2 * np.pi
            self.FA = np.ones(self.w.shape, dtype='complex128')
        if np.any(self.motion) == None:
            if self.filename == None:
                self.getInMotion(self.name, **kwargs)
            else:
                self.getInMotion(self.filename, **kwargs)

        if np.any(self.FA) == None:
            self.getMotInfo(self.motion, self.dt, **kwargs)



    def getInMotion(self, motion_name, **kwargs):
```

```
        '''
        Reads in the input motion from a PEER-formatted file. Also converts it
        to the frequency domain and applies scaling. Also, applies a Butterworth
        lowpass filter, if specified.

        Keyword Arguments:

        formatin = 'PEER' [default], 'PEER-OLD', or 'PEER-scaled'

        directout = False [default] or True. If true, the __init__ data
                    attributes are not updated.

        The following 'data attributes' or variables may be created:
        self.Ar =  a normalized rms acceleration
        self.predfreq = an estimation of the dominant frequency of the motion
        self.Ar2      self.Ar / self.predfreq
        '''
        Butrwrth = kwargs.get('Butrwrth', False)
        if self.Butrwrth:
            self.b_order = kwargs.get('b_order',4.)
            self.cutofffreq = kwargs.get('cutofffreq',100)
        g = self.g
        formatin = kwargs.get('formatin', 'PEER')
#        baseline = kwargs.get('baseline', False)
        extras = kwargs.get('extras', True)
        directout = kwargs.get('directout', False)
        # Import the ground motion
        count = 0
        eqmot = []
        f = open(motion_name, 'rb')
        if formatin == 'PEER-scaled':
           for line in f:
                count += 1
                if count == 7:
                    NPTS = int(line.split()[1][:-1])
                    dt = float(line.split()[3][:-4])
                if count > 7:
                    for num in line.split():
                        eqmot.append(float(num))
        elif formatin == 'PEER-OLD':
            for line in f:
                count += 1
                if count == 4:
                    NPTS = int(line.split()[1].replace(',',''))
                    dt = float(line.split()[3])
                if count > 4:
                    for num in line.split():
                        eqmot.append(float(num))
        else:
            for line in f:
                count += 1
                if count == 4:
                    NPTS = int(line.split()[0])
                    dt = float(line.split()[1])
                if count > 4:
                    for num in line.split():
                        eqmot.append(float(num))
```

```
953         motion = np.array(eqmot) * self.scalefactor
954         if max(motion.shape) != NPTS :
955             print('NPTS discrepancy! {} {}'.format(max(motion.shape), NPTS))
956             print(motion_name)
957             NPTS = max(motion.shape)
958 
959         motion = motion.reshape([NPTS])
960 #        if baseline == True:
961 #            motion = motion - np.average(motion)
962         f.close()
963         del eqmot, count, num
964 
965         self.getMotInfo(motion, dt, **kwargs)
966 
967 
968     def getMotInfo(self, scaledmotion, dt, **kwargs):
969         '''
970         This will get the transform and other info of a motion if only the
971         acceleration time history is input.
972         '''
973         extras = kwargs.get('extras',True)
974         Butrwrth = kwargs.get('Butrwrth', False)
975         if self.Butrwrth:
976             self.b_order = kwargs.get('b_order',4.)
977             self.cutofffreq = kwargs.get('cutofffreq',100)
978         g = self.g
979         directout = kwargs.get('directout', False)
980         g = self.g
981         NPTS = len(scaledmotion)
982         NFFT = 2 ** nextpow2(NPTS)
983         f = np.linspace(0, 1/dt/2, NFFT//2+1)
984         w = f[:NFFT/2+1] * 2 * np.pi
985         if Butrwrth == True:
986             w_cutoff = self.cutofffreq * 2 * np.pi
987             butter = (1 / (1 + (np.abs(w) / w_cutoff) ** (2 * self.b_order))
988                                                     ) ** 0.5
989         else:
990             butter = np.ones(w.shape)
991 
992         FA = np.fft.rfft(scaledmotion * g, NFFT) * butter
993 
994 
995         if directout == True:
996             return scaledmotion, NFFT, FA, f, w, NPTS, dt
997             pass
998         else:
999             self.motion = np.fft.irfft(FA / g)[:NPTS]
1000             self.NFFT = NFFT
1001             self.FA = FA
1002             self.f = f
1003             self.w = w
1004             self.NPTS = NPTS
1005             self.dt = dt
1006             self.time = np.linspace(0,(NPTS-1) * dt,NPTS)
1007             if extras:
1008                 sortedfreq = np.array(sorted(zip(np.abs(FA),f)))
1009                 predfreq = (np.sum(sortedfreq[-NFFT/200:,0] *
```

```
                    sortedfreq[-NFFT/200:,1]) /
                    np.sum(sortedfreq[-NFFT/200:,0]))
                Ar = calcAriasRatio(scaledmotion, dt)
                Ar2 = Ar / predfreq
                self.Ar = Ar
                self.Ar2 = Ar2
                self.predfreq = predfreq

class Dynamics(object):
    '''
    This class combines the soil profile and the ground motion objects to
    produce the dynamic Equivalent Linear Procedure.

    Input:
        SoilObject  =   This is the object created by instantiating the
                            SoilProfile class

        MotionObject =  This is the object created by instantiating the
                            GroundMotion class.

            You can specify these two objects so that you can perform different
            analyses on the same profile with the same motion


        Keyword Arguments:        Definition and default values:
            modtype    =    specifies complex shear modulus calculations.
                               The options are:
                               'FreqInd': Frequency Independent Complex Shear
                                          Modulus (Kramer, 1996) [default]
                               'FreqDep': Frequency Dependent Complex Shear
                                          Modulus (Udaka, 1975)
                                          (used by SHAKE?)
                               'Simplified': Simplified complex shear modulus
                               (Kramer, 1996)
                The calculations can be found under the _calcWaves function.

            InitGratio      The initial Gratio estimate. Should be either a
                               scalar or the length of the layers (not
                               counting bedrock). By default it is equal to
                               Gmax.

            BaseGratio      The initial Gratio for the bedrock. Since
                               there are no iteration or degradation
                               curves for the bedrock, this should be 1. [Default]

            InitDamping     The initial estimate of damping for each layer,
                               default is 0.05 (5%)

            BaseDamping    The damping constant of the bedrock, as a decimal.
                               0.01 is default. I may try to add the option of
                               adding degradation curves later

            DegradCurves   Specifies the method used to calculate the shear
                               modulus and damping degradation curves.
                               The only options currently supported are
                                   'IZ' Ishibashi and Zhang (1993) and
                                   'DS' for Darendeli and Stokoe (2001)
```

```
                              It shouldn't be difficult to add more.
                              If the 'DS' option is used, the DSVariables
                              object is instantiated and used.

          iters           Max number of iterations, default is 20

          Error           The threshold error (decimal, as a ratio of Gmax)
                             at which iterations will stop, default is 0.02.

  ===========> Iterations continue until either the number of iterations is
                  reached or the maximum computed error of the iteration is
                  less than 'Error,' the threshold error.

          outcrop_mot     0 if the input motion is an outcrop, 1 otherwise.
                             This overrides the SoilProfile object data
                             attribute, if specified, otherwise, the
                             SoilObject value is used.
          loc_mot          Overrides the location of motion given in
                             the SoilProfile object data
                             attribute, if specified, otherwise, the
                             SoilObject value is used.

          strainratio     Ratio of equivalent uniform strain divided by the
                             maximum strain. This should be a single value;
                             the default is given by the SoilObject.
                             The default given by the SoilObject is 0.65.
                              If anything other than a numerical value is
                              given, a strain ratio is calculated using
                              dissipated energy to give a weighted average
                              of the strain time history.
                              This is experimental; see self.calcDissEn to
                              see how it works.

          start_iters     If False, code will wait until the
                                DynamicsObject.iterate() method is called.
                                Default is True.

          run_all         If True, the number of equivalent cycles and
                              dissipated energy will be calculated for all
                              mid-layer depths.

         verbose          True or False [default]. If True, details from
                               each iteration are given.

  ======== The following options allow the user to finely specify degradation
            curves. They are all completely optional. They are useful when
            comparing outputs from different codes. ========================

          DCO_sigmeff     Overrides the calculated mean effective stresses
                             for use in the degradation curve equations.
                             Default are the SoilObject values.

          DCO_PI          Overrides soil profile PI values for use
                             in the degradation curve equations.
                             Default are the SoilObject values.

          DS_OCR          Provides OCR values for the Darendeli and
```

```
1124                            Stokoe (2001) Degradation Curves.
1125                            Default is 1 for all layers.
1126
1127            DS_soil         Soil number for Darendeli and Stokoe Degrad Curves
1128                            The soil type indicator:
1129                                0 for a general set of curves
1130                                  (Table 8.12 from Darendeli's Dissertation)
1131                               1 for clean sand
1132                                2 for sands with high fines contents
1133                                3 for silts
1134                            4 for clays
1135
1136            DS_N            Number of cycles for the Darendeli and
1137                            Stokoe (2001) Degradation Curves.
1138                            Default is 10 for all layers.
1139
1140            DS_freq         Frequency for the Darendeli and
1141                            Stokoe (2001) Degradation Curves.
1142                            Default is 1 Hz for all layers.
1143
1144            DS_Dmin          An alternative way to calc Dmin for the Darendeli
1145                            and Stokoe Degradation curves.
1146                            1 for 'Default', 0 for 'Green'
1147                            !!! Not Implemented !!!
1148
1149            DCO_curve       If True, user must input their own degradation
1150                            curves, specified with the following keywords:
1151                DCO_gam = shear strains as decimals (an array n long)
1152                DCO_Gratio = G/Gmax values (decimal) (an array m x n, where
1153                             m is the number of layers above the baserock)
1154                DCO_damping = damping (decimal) in the same shape as DCO_Gratio
1155
1156        The following data attributes are created by the __init__ method.
1157            Since the init method calls other methods, the list may not be
1158            complete.
1159
1160            self.g
1161            self.Gmax
1162            self.G
1163            self.D
1164            self.DegradCurves
1165            self.modtype
1166            self.iters
1167            self.ComIter    =  Flag to show when iterations have completed.
1168            self.Error
1169            self.gam
1170            self.Gratio
1171            self.damping
1172            self.loc_mot
1173            self.SoilObj
1174            self.MotObj
1175            self.outcrop_mot
1176            self.dissEn
1177            self.strainratio
1178            self.verbose
1179            self.Ar =  a normalized rms acceleration.
1180            self.Ar_ratio = self.Ar / Ar of the input motion.
```

```
1181
1182
1183        '''
1184        def __init__(self, SoilObject, MotionObject, **kwargs):
1185            self.g = SoilObject.g
1186            self.Gmax = (SoilObject.unit_wt / self.g) * SoilObject.Vs ** 2
1187            self.G = np.copy(self.Gmax)
1188            self.G[:-1] = self.G[:-1] * kwargs.get('InitGratio', 1.)
1189            self.G[-1] = self.G[-1] * kwargs.get('BaseGratio',1.)
1190            self.InitG = np.copy(self.G)
1191            self.ComIter = False
1192            self.D = np.ones(len(self.Gmax)) * 0.05
1193            self.D[:-1] = kwargs.get('InitDamping', 0.05) # <------Initial estimate of damping;
                    may want to change
1194            self.D[-1] = kwargs.get('BaseDamping',0.01) # <<<--------------------------
1195            self.InitD = np.copy(self.D)
1196            self.DegradCurves = kwargs.get('DegradCurves','IZ')
1197            self.modtype = kwargs.get('modtype', 'FreqInd')
1198            self.iters = kwargs.get('iters',30)
1199            self.Error = kwargs.get('Error',0.02)
1200
1201            # Get the degradation curves
1202            gam = kwargs.get('gam')
1203            if kwargs.get('DCO_curve',False) == True:
1204                    self.gam = kwargs.get('DS_gam')
1205                    self.Gratio = kwargs.get('DCO_Gratio')
1206                    self.damping = kwargs.get('DCO_damping')
1207            if self.DegradCurves == 'DS':
1208                self.DSvars = DSvariables(kwargs,SoilObject)
1209
1210                self.gam, self.Gratio, self.damping = self._getDegrad(
1211                                            self.DegradCurves,
1212                                            self.DSvars.PI,
1213                                            self.DSvars.sigmeff,
1214                                            OCR = self.DSvars.OCR,
1215                                            soil = self.DSvars.soiltype,
1216                                            N = self.DSvars.N,
1217                                            freq = self.DSvars.freq,
1218                                            gam = gam)
1219            elif self.DegradCurves == 'IZ':
1220                self.gam,self.Gratio, self.damping = self._getDegrad(
1221                                                    self.DegradCurves,
1222                                                    SoilObject.PI,
1223                                                    SoilObject.sigmeff,
1224                                                    gam=gam)
1225            else:
1226                line=('Degradation Method input not understood\n' +
1227                        'or a custom degradation curve has been entered.')
1228                print(line)
1229
1230
1231            self.SoilObj = SoilObject
1232            self.loc_mot = kwargs.get('loc_mot', self.SoilObj.loc_mot)
1233
1234            self.MotObj = MotionObject
1235            self.outcrop_mot = kwargs.get('outcrop_mot',self.SoilObj.outcrop_mot)
1236            self.gammf = None
```

```
1237            self.dissEn = None
1238
1239            # Start Iterations
1240            start_iters = kwargs.get('start_iters',True)
1241            if start_iters:
1242                self.strainratio = kwargs.get('strainratio',self.SoilObj.strainratio)
1243                self.verbose = kwargs.get('verbose',False)
1244                self.iterate(iters=self.iters,
1245                             Error=self.Error,
1246                             strainratio=self.strainratio,
1247                             verbose=self.verbose)
1248
1249            # Run_all
1250            run_all = kwargs.get('run_all',False)
1251            if run_all:
1252                self.calcNeq()
1253                self.Ar,self.PeakTimeRatio = calcAriasRatio(self.calcAcc(domain='time',
1254                                            MotType='within',
1255                                            MaxOnly=False,
1256                                            ), self.MotObj.dt, returnall=True)
1257                Ar_input = calcAriasRatio(self.MotObj.motion,self.MotObj.dt)
1258                self.Ar_ratio = self.Ar / Ar_input
1259
1260        def reset(self):
1261            '''
1262            Resets G and D to initial values
1263            '''
1264            self.G = np.copy(self.InitG)
1265            self.D = np.copy(self.InitD)
1266            self.ComIter = False
1267
1268
1269        def iterate(self,**kwargs):
1270            '''
1271            Performs the equivalent linear iterations until either the iteration
1272            limit is reached or the error goes below the predefined threshold.
1273
1274            Keywords:           Definition/Default:
1275              iters              number of iterations, default is that defined
1276                                    by the self.__init__ function
1277
1278              Error              the error threshold, default is that defined
1279                                    by the self.__init__ function
1280
1281              strainratio        Ratio of equivalent uniform strain divided by
1282                                    the maximum strain. This should be a single
1283                                    value; the default is given
1284                                    by the SoilObject. The default given by the
1285                                    SoilObject is 0.65.
1286                                    If anything other than a numerical value is
1287                                    given, a strain ratio is calculated using
1288                                    dissipated energy to give a weighted average
1289                                    of the strain time history.
1290
1291              verbose             True or False [default]. If True, details from
1292                                    each iteration are given.
1293
```

```
        Data attributes created by this method:
        self.gamavgarray = average shear strain of all layers
        self.Garray = array of current degraded shear moduli for every layer
                      for every iteration
        self.Darray = array of current degraded damping for every layer
                      for every iteration
        self.Errarray = array of errors for every layer
                      for every iteration
        self.Gratiocurv = an object to interpolate new values of modulus
        self.Dcurv = an object to interpolate new values of damping
        self.count = current or final count of iterations
        self.ratio = strain ratios used for the iteration
        self.ReportError = 'y' if an error has occurred, 'n' if not.

        Updated or created data attributes:
        self.G
        self.D
        self.A
        self.B
        self.ComIter
        self.gammf = strain for mid point or each layer in freq domain
        self.taumf = stress for mid point or each layer in freq domain
        '''
        self.gamavgarray = np.zeros((self.Gratio.shape[0],max(self.iters,1)),
                                    dtype=float)
        self.Garray = np.zeros((self.Gratio.shape[0],max(self.iters,1)),
                               dtype=float) #G/Gmax
        self.Darray = np.zeros((self.Gratio.shape[0],max(self.iters,1)),
                               dtype=float) #D
        self.Errarray = np.zeros((max(self.iters,1), len(self.G)-1),
                dtype=float)
#        self.Gratiocurv = interp1d(self.gam, self.Gratio, kind='linear', axis = 1)
#        self.Dcurv = interp1d(self.gam, self.damping, kind='linear', axis = 1)
        self.Gratiocurv = interp1d(np.log10(self.gam), self.Gratio,
                                   kind='linear', axis = 1)
        self.Dcurv = interp1d(np.log10(self.gam), self.damping,
                              kind='linear', axis = 1)
        self.iters = kwargs.get('iters', self.iters)
        Error = kwargs.get('Error', self.Error)
        self.strainratio = kwargs.get('strainratio',self.SoilObj.strainratio)
        self.verbose = kwargs.get('verbose',False)
        iterate = True
        self.count = 0
        exceedcount = 0
        while iterate == True: # Begin Iterations
            self.count += 1
            self._calcWaves() # Get the new upgoing and down going waves
            try:
                # Won't work if the strain ratio is a string, go to except
                self.ratio = (float(self.strainratio) *
                              np.ones(self.G[:-1].shape))
                # Get the max stress and strain values
                taumax,gammax = self._calcTauGam(MaxOnly=True,domain='time',
                                                 returns=True)

            except ValueError:
                # Get the stress and strain for each layer in the time domain
```

```
1351                taut, gamt = self._calcTauGam(MaxOnly=False,domain='time',
1352                                              returns=True)
1353                # Calc the strain ratio using the dissipated energy method
1354                self.ratio = self.calcStrainRatio_DE(taut, gamt)
1355                gammax = np.max(np.abs(gamt),axis=1)
1356
1357            self.gamavgarray[:,self.count-1] = self.ratio * gammax
1358            self.gammax = gammax
1359            # For really stiff profiles or soft motions, self.ratio * gammax
1360            # may fall below the gam range. Here we make sure we can interpolate.
1361            if np.max(self.ratio * gammax) > self.gam[-1]:
1362                exceedcount += 1
1363
1364            gammain = np.min((
1365                          np.max((
1366                              self.ratio*gammax,
1367                              self.gam[0] * np.ones(
1368                              gammax.shape)), axis=0),
1369                              self.gam[-1]*np.ones(gammax.shape)
1370                            ), axis=0
1371                          )
1372            # Get the new G and D from interpolated curves
1373            try:
1374                Gnewmat = self.Gratiocurv(np.log10(gammain))
1375    #             Gnewmat = self.Gratiocurv(self.ratio * gammax)
1376                Gnew = self.Gmax[:-1] * np.diag(Gnewmat)
1377                Dnewmat = self.Dcurv(np.log10(gammain))
1378    #             Dnewmat = self.Dcurv(self.ratio * gammax)
1379                Dnew = np.diag(Dnewmat)
1380            except ValueError:
1381                print('ValueError!')
1382                print('Soil Profile: {}'.format(self.SoilObj.profile_name))
1383                print('Ground Motion: {} {}'.format(self.MotObj.name,
1384                           self.MotObj.filename))
1385                print('Strain Ratio: {}'.format(self.ratio))
1386                print('Gamma Max: {}'.format(gammax))
1387                print('G: {}'.format(self.G))
1388                print('D: {}'.format(self.D))
1389                self.ReportError = 'y'
1390                pdb.set_trace()
1391                raise Exception("ValueError in the interpolation of the" +
1392                             'degradation curves' )
1393                return
1394
1395            Err = (self.G[:-1] - Gnew)/self.Gmax[:-1] # calc the error
1396            self.Errarray[self.count - 1,:] = Err
1397            # Check if we can duck out of iterations
1398            if np.max(abs(Err)) <= Error:
1399                iterate = False
1400                self.ReportError = 'n'
1401                self.ComIter = True
1402                self.MaxError = np.max(abs(Err))
1403
1404                #if exceedcount >= 2:
1405                    #iterate = True
1406                #if exceedcount >= 3:
1407                    #pdb.set_trace()
```

```
            if self.count >= self.iters:
                iterate = False
                if np.max(abs(Err)) <= Error:
                    pass
                    self.ReportError = 'n'
                    self.ComIter = True
                else:
                    print('Reached {:d} iterations without converging!'.format(
                                        self.iters))
                    self.ReportError = 'y'
                    self.ComIter = False
                self.MaxError = np.max(abs(Err))
            # print out data about each iteration/layer
            if self.verbose == True:
                print('Iteration: {}\n'.format(self.count))
#                print('Gamma Max: {}'.format(gammax))
#                print('G:'.format(self.G))
#                print('D:'.format(self.D))
#                print(np.max(self.A))
                print('{:^11} {:^9} {:^9} {:^9} {:^9}'.format(
                          'Gamma_max:',
                          'Gold',
                          'Gnew',
                          'Dold',
                          'Dnew',
                          'Error'))
                for i in xrange(len(Gnew)):
                    print('{:^10.6e} {:^9.0f} {:^9.0f}'.format(
                          gammax[i],
                          self.G[i],
                          Gnew[i]) +
                         '{:^9.4f} {:^9.4f} {:^9.4}'.format(
                          self.D[i],
                          Dnew[i],
                          Err[i]))
            # Get ready for the new iteration.
            if iterate == True:
                self.G[:-1] = Gnew
                self.D[:-1] = Dnew
                self.Garray[:,self.count-1] = self.G[:-1]/self.Gmax[:-1]
                self.Darray[:,self.count-1] = self.D[:-1]


    def _calcWaves(self):
        '''
        Calculates the up-going and down-going wave amplitudes.

            This method is called without input arguments to calculate the
            amplitudes of the upgoing and downgoing waves (A & B). The input
            are already defined as part of the object.

            The following data attributes are created:
            self.Gstr = complex shear modulus
            self.Vsstr = complex shear wave velocity
            self.ks = wave number?
            self.alps = alpha, the layer impedence ratio
```

```
1465            self.A = amplitude of upgoing waves
1466            self.B = amplitude of downgoing waves
1467            self.Updated = flag for something.
1468        '''
1469        g = self.g
1470        G = self.G
1471        D = self.D
1472        h = self.SoilObj.t
1473        unit_wt = self.SoilObj.unit_wt
1474        try:
1475            w = self.MotObj.w
1476            FA = self.MotObj.FA
1477        except AttributeError:
1478            print(self.MotObj.filename)
1479            print(self.MotObj.dt)
1480        # From the Deepsoil Manual:
1481        if self.modtype == 'FreqDep': # Frequency Dependent Complex Shear Modulus (Udaka,
                1975) (Shake?)
1482            self.Gstr = G * (1 - 2 * D ** 2 + 1j * 2 * D * (1 - D ** 2) ** (0.5 + 0j))
1483        elif self.modtype == 'FreqInd': # Frequency Independent Complex Shear Modulus
                (Kramer, 1996)
1484            self.Gstr = G * (1 + 1j * 2 * D)
1485        elif self.modtype == 'Simplified': # Simplified complex shear modulus (Kramer, 1996)
1486            self.Gstr = G * (1 - D ** 2 + 1j * 2 * D)
1487        Gstr = self.Gstr
1488        self.Vsstr = (Gstr * g / unit_wt) ** 0.5
1489 #       self.Vsstr = ne.evaluate('(Gstr * g / unit_wt) ** 0.5')
1490        Vsstr = self.Vsstr
1491        self.ks = w / Vsstr[:,np.newaxis]
1492        ks = self.ks
1493
1494        alps = ((unit_wt[:-1] * Vsstr[:-1]) /
1495                (unit_wt[1:] * Vsstr[1:]))
1496        # Initialize the up and down going waves.
1497        A=np.ones(ks.shape, dtype='complex128')
1498        B=np.ones(ks.shape, dtype='complex128')
1499        # Calculate the waves, this is somewhat computationally intensive. C code could be
                useful here in the future.
1500        for i in xrange(len(h)):
1501            A[i+1,:] = (0.5 * A[i,:] * (1 + alps[i]) *
1502                    np.exp(1j * ks[i,:] * h[i]) +
1503                    0.5 * B[i,:] * (1 - alps[i]) *
1504                    np.exp(-1j * ks[i,:] * h[i]))
1505            B[i+1,:] = (0.5 * A[i,:] * (1 - alps[i]) *
1506                    np.exp(1j * ks[i,:] * h[i]) +
1507                    0.5 * B[i,:] * (1 + alps[i]) *
1508                    np.exp(-1j * ks[i,:] * h[i]))
1509        # Get the input motion
1510        FD = np.zeros(FA.shape, dtype=complex)
1511        FD[1:] = FA[1:] / (1j*w[1:]) ** 2
1512        FD[0] = FA[0] #* (self.MotObj.dt * (self.MotObj.NPTS - 1)) ** 2 #
                <<<<==================== This may not be strictly legitimate
1513        if self.loc_mot == 0:
1514            index = -1
1515        else:
1516            index = self.loc_mot - 1 # Subtracting one here to convert to python's 0
                    counting method
```

```
1517            if index > len(A):
1518                index = -1
1519            if self.outcrop_mot == 0:
1520                A1new = FD / (2 * A[index, :])
1521            else:
1522                A1new = FD / (A[index,:] + B[index, :])
1523
1524    #       Aratio = A1new / A[index,:] # <++++ Double checking my calcs: 3 Sept 2013,
          Everything is okay.
1525    #       A2 = A * Aratio[np.newaxis,:]
1526    #       Bratio = A2[0,:] / A[0,:]
1527    #       B2 = B * Bratio[np.newaxis,:]
1528            # Getting the corrected waves from the input motion
1529            self.A = A * A1new[np.newaxis, :]
1530            self.B = B * A1new[np.newaxis, :]
1531    #       plt.figure(1)
1532    #       plt.plot(abs(self.A[0,:]))
1533    #       plt.plot(abs(A2[0,:]),'o')
1534    #       plt.figure(2)
1535    #       plt.plot(abs(self.B[0,:]))
1536    #       plt.plot(abs(B2[0,:]),'o')
1537    #       plt.show()
1538    #       pdb.set_trace()
1539            self.alps = alps
1540            self.Updated = True
1541
1542
1543        def _getDegrad(self,Degrad, PI, sigmeff, **kwargs):
1544            '''
1545            Gets the shear modulus and damping degradation curves.
1546
1547            Outputs a 1D array (gam = shear strain (decimal) and 2- 2D arrays
1548            (Gratio = G/Gmax and damping (decimal)) that represent the shear
1549            modulus and damping degradation curves. It depends on the module
1550            degrcurv, so use that if you want to customize it. Or, create your own
1551            degradation curves.
1552            Right now, Degrad must equal 'IZ' for the Ishibashi and Zhang (1993)
1553            curves or 'DS' for Darendeli and Stokoe (2001).
1554            sigmeff must be in kPa
1555
1556            Keyword Arguments:
1557                (See the __init__ for these definitions, only for 'DS')
1558            OCR
1559            soil
1560            N
1561            freq
1562            Dmintype !!! Not Implemented !!!
1563
1564            Everything but Degrad should be an array, or you will get errors.
1565            '''
1566            #straindef = np.array([0.0000001, 0.000001, 0.000003, 0.00001,
1567                                  #0.00003, 0.0001, 0.0003, 0.001, 0.003,
1568                                  #0.01, 0.99])
1569            straindef = np.logspace(-6, -2, num=20, endpoint=True)
1570            if Degrad == 'DS':
1571                OCR = kwargs.get('OCR')
1572                soil = kwargs.get('soil')
```

```
            N = kwargs.get('N')
            freq = kwargs.get('freq')
#           Dmintype = kwargs.get('DS_Dmin')
            if np.isscalar(OCR):
                OCR = np.ones(len(PI),dtype=float) * OCR
            if np.isscalar(soil):
                soil = np.ones(len(PI),dtype=int) * soil
            if np.isscalar(N):
                N = np.ones(len(PI),dtype=float) * N
            if np.isscalar(freq):
                freq = np.ones(len(PI),dtype=float) * freq
        gam = kwargs.get('gam', straindef)
        if gam == None:
            gam = straindef
        damping = np.zeros([len(PI),len(gam)],dtype=float)
        Gratio = np.zeros([len(PI),len(gam)],dtype=float)
        for i in xrange(len(PI)):
            if Degrad == 'IZ':
                gam, Gratio[i], damping[i] = dc.IZ_1993(gam = gam,
                                                        PI=PI[i],
                                                        sigm=sigmeff[i])
            elif Degrad == 'DS':
                gam,Gratio[i],damping[i] = dc.DS_2001(gam = gam,
                                                      PI=PI[i],
                                                      sigm=sigmeff[i],
                                                      OCR=OCR[i],
                                                      soil=soil[i],
                                                      N=N[i],
                                                      frq=freq[i])

        return gam, Gratio, damping


    def _calcTauGam(self, **kwargs):
        '''
        This calculate thes stress and strains for arbitrary conditions.

        Keyword Arguments:        Definitions/Default values:
          depths             array or single value of depth at which the
                                 stresses and strains should be calculated.
                                 Default is the mid-depth of each layer.

          domain            'time' [default] or 'freq'

          MaxOnly            True [default] or False. When True, only the
                                 absolute maximum value of each layer is
                                 returned. This will force the domain to be
                                 'time' and the returns to be True.

          returns            True or False [default]. When True, the stresses
                                 and strains are returned. Otherwise, they
                                 are data attributes of the object
                                 (self.taumf and self.gammf, but only when the
                                 depths are equal to the mid-depths of the
                                 layers).

          BaseCorr           True or False [default]. Applies a baseline
```

```
                            correction to the strain in time domain by
                            subtracting the average value.

        Example:
            tau, gam = [DynamicsObject]._calcTauGam(depths = [0.1,0.4],
                                                    domain = 'time'
                                                    returns = True)

        Data attributes possibly created by this method:
        self.taumf
        self.gammf
        '''
        ComIter = self.ComIter
        t_mid = self.SoilObj.t_mid
        A = self.A
        B = self.B
        ks = self.ks
        t = self.SoilObj.t
        Gstr = self.Gstr

        depths = kwargs.get('depths',t_mid)
        domain = kwargs.get('domain','time')
        MaxOnly = kwargs.get('MaxOnly',True)
        returns = kwargs.get('returns',False)
        BaseCorr = kwargs.get('BaseCorr',False)
        if MaxOnly == True:
            domain = 'time'
            returns = True

        def hardway(t):
            '''
            Gets the requested values when things are trickier.
            returns tau, gam
            '''
            returns = True
            t = np.append(t, 1000)
            gam = np.zeros([len(depths), np.shape(A)[1]], dtype='complex128')
            tau = np.zeros([len(depths), np.shape(A)[1]], dtype='complex128')
            tsum = np.cumsum(t, dtype=float)

            j = 0
            for i in xrange(len(depths)):
                while depths[i] >= round(tsum[j], 3):
                    j += 1
                if depths[i] < t[0]:
                    d = depths[i]
                else:
                    d = depths[i] - tsum[j-1]
                gam[i] = 1j * ks[j] * (A[j, :] * np.exp(1j *
                           ks[j] * d) - B[j, :] *
                           np.exp(-1j * ks[j] * d))
                tau[i] = gam[i] * Gstr[j]
            return tau,gam


        # Check if I have to do it the hard way.
        if len(np.ravel(depths)) != len(np.ravel(t_mid)):
```

```
1687            tau,gam = hardway(t)
1688        else:
1689            if np.equal(depths, t_mid).all():
1690                if ComIter == True: # If the iterations are finished, I can call existing
                        taumf and gammf
1691                    gam = self.gammf
1692                    tau = self.taumf
1693                else:
1694                    gam = (1j * ks[:-1] *
1695                          (A[:-1,:] * np.exp(1j * ks[:-1] *
1696                           t[:,np.newaxis] /2) -
1697                           B[:-1,:] * np.exp(-1j * ks[:-1] *
1698                           t[:,np.newaxis] /2)))
1699                    tau = gam * Gstr[:-1,np.newaxis]
1700                    self.gammf = gam
1701                    self.taumf = tau
1702            else:
1703                tau,gam = hardway(t)
1704
1705        # Do I need to convert my motions to the time domain?
1706        if domain == 'time':
1707            NPTS = self.MotObj.NPTS
1708            tau = np.fft.irfft(tau, axis=1)
1709            gam = np.fft.irfft(gam, axis=1)
1710            tau = tau[:,:NPTS]
1711            gam = gam[:,:NPTS]
1712            if BaseCorr == True: # This correction is pretty elementary; I don't like it.
1713                basecorrect = np.mean(gam, axis=1)
1714                gam = gam - basecorrect[:,np.newaxis] # Baseline correction
1715
1716        if MaxOnly == True:
1717            tau = abs(tau).max(1)
1718            gam = abs(gam).max(1)
1719        if returns == True:
1720            return tau, gam
1721
1722
1723    def _AccVelDisp(self,**kwargs):
1724        '''
1725        Calculates acceleration, velocity, or displacement.
1726
1727        Keyword argument:   Options: (Default listed first)
1728            which        =   'Acc' , 'Vel'  or 'Disp'
1729
1730            domain       =   'time'  or   'freq'
1731
1732            MotType      =   'within', 'outcrop' or 'incoming'
1733
1734            MaxOnly      =   False or True
1735
1736            depths       =   mid-depths of each layer or list of desired
1737
1738        It may be easier to use the self.calcAcc, self.calcDisp, or
1739            self.calcVel methods. This is meant for internal use only.
1740        '''
1741        Which = kwargs.get('which','Acc')
1742        domain = kwargs.get('domain','time')
```

```
        MotType = kwargs.get('MotType','within')
        MaxOnly = kwargs.get('MaxOnly',False)
        depths = kwargs.get('depths',self.SoilObj.t_mid)
        smooth = False
        if domain == 'freq':
            smooth = kwargs.get('smooth',False)
            window = kwargs.get('window',20)
        A = self.A
        B = self.B
        h = self.SoilObj.t
        h = np.append(h,1000)
        w = self.MotObj.w
        NPTS = self.MotObj.NPTS
        g = self.g
        ks = self.ks
        disp = np.zeros([len(depths),np.shape(A)[1]], dtype='complex128')
        hsum = np.cumsum(h, dtype=float)

        # Figure out which layer we are in, and calc the displacement
        j = 0
        for i in xrange(len(depths)):
            while depths[i] > round(hsum[j],3):
                j += 1
            if depths[i] < h[0]:
                d = depths[i]
            else:
                d = depths[i] - hsum[j-1]
            if MotType == 'incoming':
                disp[i] = A[j,:] * np.exp(1j * ks[j] * d)
            elif MotType == 'within':
                disp[i] = (A[j,:] * np.exp(1j * ks[j] * d) +
                       B[j,:] * np.exp(-1j * ks[j] * d))
            else:
                disp[i] = 2 * A[j,:] * np.exp(1j * ks[j] * d)

        # Convert to Acc or Vel, if needed.
        if Which == 'Acc':
            out = disp * (1j * w[np.newaxis,:]) ** 2 / g
        elif Which == 'Vel':
            out = disp * (1j * w[np.newaxis,:])
        else:
            out = disp


        if domain == 'time':
            out = np.fft.irfft(out, axis=1)
            out = np.delete(out, np.s_[NPTS:], axis=1)

        # Not Fully implemented.
        if smooth == True:
            del i, j

            ind = np.arange(out.shape[1],dtype=int)
            indlow = ind - window / 2
            indhigh = ind + window / 2
            np.putmask(indlow,indlow<0,0)
            np.putmask(indhigh, indhigh>ind[-1],ind[-1])
```

```
            smoothout = np.zeros(out.shape,dtype=complex)
            for j in ind:
                smoothout[:,j] = np.mean(out[:,indlow[j]:indhigh[j]]) * (window)
            out = smoothout

        if MaxOnly == True:
            out = abs(out).max(1)

        return out


    def calcStress(self, *args, **kwargs):
        '''
        Use this method to obtain the stress at one or more depths.

        Keyword Argumentss:        Definitions/Default values:
          depths            array or single value of depth at which the
                                stresses should be calculated.
                                Default is the mid-depth of each layer.

          domain           'time' [default] or 'freq'

          MaxOnly           True or False [default] . When True, only
                                the absolute maximum value of each layer is
                                returned. This will force the domain to be
                                'time' and the returns to be True.

         ratio             True or False [default].
                               Calculates the cyclic stress ratio or CSR.

         ratiotype        Choose 'verteff' or 'meaneff' {not implemented yet}

         time               True or False[default]. Returns an array of the
                               time if in the time domain. Else it returns
                               the frequency array if the domain is frequency.

        This method calls the self._calcTauGam method. It does not set any
        data attributes.
        '''
        domain = kwargs.get('domain','time')
        MaxOnly = kwargs.get('MaxOnly',False)
        ratio = kwargs.get('ratio',False)
        ratiotype = kwargs.get('ratiotype')
        time = kwargs.get('time',False)
        if ratiotype:
            ratio = True
        if len(args) > 0:
            depths = np.array(args)
        else:
            depths = kwargs.get('depths',self.SoilObj.t_mid)

        out, strain = self._calcTauGam(depths=depths,
                                domain=domain,
                                MaxOnly=MaxOnly,
                                returns=True)
        if ratio == True:
            if ratiotype == 'meaneff':
```

```
                pass
            else:
                effstr = self.SoilObj.calcInsituStress(depths, out='eff')

            if len(depths) > 1:
                out = out / effstr[:,np.newaxis]
            else:
                out = out / effstr

        if len(depths) == 1:
            out = np.ravel(out)
        if time == True:
            if domain == 'time':
                xarr = self.MotObj.time
            else:
                xarr = self.MotObj.f

            return xarr, out
        else:
            return out


    def calcStrain(self, *args, **kwargs):
        '''
        Use this method to obtain the strain at one or more depths.

        Keyword Argumentss:        Definitions/Default values:
          depths            array or single value of depth at which the
                                strains should be calculated.
                                Default is the mid-depth of each layer.

          domain           'time' [default] or 'freq'

          MaxOnly           True [default] or False. When True, only
                                the absolute maximum value of each layer is
                                returned. This will force the domain to be
                                'time' and the returns to be True.

         time               True or False[default]. Returns an array of the
                               time if in the time domain. Else it returns
                               the frequency array if the domain is frequency.

        This method calls the self._calcTauGam method. It does not set any
        data attributes.
        '''
        domain = kwargs.get('domain','time')
        MaxOnly = kwargs.get('MaxOnly',False)
        time = kwargs.get('time',False)
        if len(args) > 0:
            depths = np.array(args)
        else:
            depths = kwargs.get('depths',self.SoilObj.t_mid)

        stress, out = self._calcTauGam(depths=depths,
                                domain=domain,
                                MaxOnly=MaxOnly,
                                returns=True)
```

```
1914
1915            if len(depths) == 1:
1916                out = np.ravel(out)
1917            if time == True:
1918                if domain == 'time':
1919                    xarr = self.MotObj.time
1920                else:
1921                    xarr = self.MotObj.f
1922
1923                return xarr, out
1924            else:
1925                return out
1926
1927
1928        def calcNeq(self,*args,**kwargs):
1929            '''
1930            Calculates the equivalent number of cycles using dissipated energy.
1931
1932             Positional Arguments:
1933                depths at which equivalent number of cycles are calculated.
1934                ** Before refactoring, I allowed the args to be the strain ratio.
1935                Hopefully I have caught and changed all the old ones.
1936
1937            Keyword Arguments       Definition/Default
1938             depths   =     array of depths, default is mid-depth of each layer
1939                              If the depths are specified as a keyword argument,
1940                              these depths are not used.
1941
1942             set_ratio =   ratio of uniform sinusoidal amplitude and max strain
1943                               Note: this sets the ratio_Neq attribute and
1944                               defaults to the ratio used in the iteration
1945                               method self.ratio. Allows me to look at how
1946                               the strain ratio affects the Neq in this calc.
1947
1948             returns  = False {default} or True. By default, this function does
1949                           not return any values. Instead, the data attributes
1950                           of N_eq, tau_avgNeq, Neq_ratio, and dissEn are created
1951                           or updated.
1952
1953            Data attributes created/updated by this method if he mid-layer depths
1954            are used. Note that these values will change for every set_ratio value.
1955            self.N_eq
1956            self.tau_avgNeq = the amplitude of the equiv. sinusoidal stress motions
1957            self.Neq_ratio = the strain ratio corresponding to the self.N_eq
1958            self.dissEn -> only created if the mid-layer depths are used.
1959
1960            This method calls the self.calcDissEn method.
1961            '''
1962            if len(args) > 0:
1963                depths = np.array(args)
1964            else:
1965                depths = kwargs.get('depths',self.SoilObj.t_mid)
1966            ratio = kwargs.get('set_ratio',self.ratio)
1967            returns = kwargs.get('returns',False)
1968            t_mid = self.SoilObj.t_mid
1969            t = self.SoilObj.t
1970
```

```
        def hardway(t):
            '''
            Gets the stiffness and damping if the mid-layer depths aren't
            specified.
            '''
            t = np.append(t, 1000)
            G = np.zeros([len(depths),], dtype=float)
            D = np.zeros([len(depths),], dtype=float)
            tsum = np.cumsum(t, dtype=float)

            j = 0
            for i in xrange(len(depths)):
                while depths[i] >= round(tsum[j], 3):
                    j += 1
                G[i] = self.G[j]
                D[i] = self.D[j]
            return G,D

        # Check if I have to use the hard way to get G, D.
        if len(np.ravel(depths)) != len(np.ravel(t_mid)):
            same = False
            G,D = hardway(t)

        else:
            if np.equal(depths, t_mid).all():
                same = True
                G = self.G[:-1]
                D = self.D[:-1]
            else:
                same = False
                G,D = hardway(t)

        tau, gam = self._calcTauGam(domain='time',
                                    MaxOnly=False,
                                    depths=depths,
                                    returns = True)
        DE = self.calcDissEn(tau, gam, output='normal')


        # get max stresses
        taumax = np.max(np.abs(tau),axis=1)
        tau_avgNeq = ratio * taumax
        #tau_avgNeq = 0.65 * taumax
        dissEn_one = 2 * np.pi * D * tau_avgNeq ** 2 / G
        N_eq = DE / dissEn_one
        if same == True:
            self.tau_avgNeq = tau_avgNeq
            self.N_eq = N_eq
            self.Neq_ratio = ratio
            self.dissEn = DE

        if returns == True:
            return N_eq, tau_avgNeq, ratio, DE


    def calcDissEn(self, stresses, strains, **kwargs):
        '''
```

```
        Calculates the dissipated energy for a given set of stress and strain
        time histories.


        Input:
            stresses  m x n array of stress time histories with each layer
                           in a different row. Make sure to use
                           stress = ifft(G* * strain(omega))
                           where G* is the complex shear modulus and
                           strain(omega) is the shear strain as a decimal in
                           the frequency domain. m is the number of layers and
                           n is the number of pts in the time history
                           (self.MotObj.NPTS)

            strains   m x n array of strain time histories (decimal),
                           in the same format as the stresses.
        Keyword Arguments:

            FinalValOnly = True | False


        Output:
            DissE   The dissipated energy in units of stress per unit volume
                       if output='normal'. Else, an array of strain ratios.
                       To obtain the normalized dissipated energy like that
                       given by SHAKEVT, divide the dissipated energy
                       by the vertical effective stress.
        '''
        if kwargs.get('FinalValOnly', True):
            dissEn = np.zeros(np.shape(stresses)[0], dtype=float)
            dissEn = np.sum((stresses[:,1:] + stresses[:,:-1]) *
                           (strains[:,1:] - strains[:,:-1]), axis = 1) * 0.5
            return dissEn
        else:
            dissEn = np.zeros(stresses.shape, dtype=float)
            dissEn = np.cumsum((stresses[:,1:] + stresses[:,:-1]) *
                               (strains[:,1:] - strains[:,:-1]) * 0.5,
                               axis = 1)
            return dissEn



    def calcStrainRatio_DE(self, stresses, strains, **kwargs):
        '''
        Calculates a strain ratio using the weighted average
                       strain from the dissipated energy.

        Input:
            stresses  m x n array of stress time histories with each layer
                           in a different row. Make sure to use
                           stress = ifft(G* * strain(omega))
                           where G* is the complex shear modulus and
                           strain(omega) is the shear strain as a decimal in
                           the frequency domain. m is the number of layers and
                           n is the number of pts in the time history
                           (self.MotObj.NPTS)

```

```
            strains   m x n array of strain time histories (decimal),
                            in the same format as the stresses.

        Output: An array of strain ratios of length m.

        '''
        plot = kwargs.get('plot', False)
        dissEn = np.zeros(np.shape(stresses)[0], dtype=float)
        dissEn = ((stresses[:,1:] + stresses[:,:-1]) *
                  (strains[:,1:] - strains[:,:-1])) * 0.5
        x = stresses/strains
        dx = np.diff(x)
        avgstrain = np.zeros(strains.shape[0],)

        for i in xrange(np.shape(x)[0]):
            ind = np.where((dx[i,:-2] < 0) & (dx[i,1:-1] > 0) & (dx[i,2:]<0))
            if ind[0][0] == 0:
                ind = ind[0] + 1
            else:
                ind = np.insert(ind[0],0,0)
            dDE = np.zeros((ind.shape[0]),)
            dstrain = np.zeros((ind.shape[0]),)
#           dstress = np.zeros((ind.shape[0]),)

            for k in xrange(len(ind)-2):
                if k % 2 == 1:
                    continue
                else:
                    dstrain[k] = (np.max(strains[i,ind[k]:ind[k+2]]) -
                                  np.min(strains[i,ind[k]:ind[k+2]]))/2
#                    dstress[k] = (np.max(stresses[i,ind[k]:ind[k+2]]) -
#                                  np.min(stresses[i,ind[k]:ind[k+2]]))/2
                    dDE[k] = dissEn[i,ind[k+2]] - dissEn[i,ind[k]]

                    if plot == True:
                        plt.plot(strains[i,ind[k]:ind[k+2]],
                                 stresses[i,ind[k]:ind[k+2]],'-b')
                        plt.xlabel('Strain')
                        plt.ylabel('Stress (kPa)')
                        plt.show()
            avgstrain[i] = np.average(dstrain,weights=dDE)
#           G = dstress / dstrain
#           print(self.G[i])
#           pdb.set_trace()
            del dDE, ind, dstrain#, dstress
        strainratio = avgstrain / np.max(strains,axis=1)
        np.putmask(strainratio,strainratio<=0,0.1)
        np.putmask(strainratio,strainratio>1.,1.)
        return strainratio


    def calcDisp(self,*args,**kwargs):
        '''
        Calculates displacements using the _AccVelDisp method.

        Positional Arguments, if used, are the depths.

```

```
        Keyword argument:   Options: (Default listed first)

            domain        =   'time'  or   'freq'

            MotType       =   'within', 'outcrop' or 'incoming'

            MaxOnly       =   False or True

            depths        =   mid-depths of each layer or list of desired
                               These depths can also be listed as regular,
                               non-keyword arguments; see the example.

         Returns an array of velocities in units of m for time
                histories, m-s for frequency domain. Array is 2D if more
                than one depth is specified.

         Example:
             disp = DynamicObject.calcDisp(2.0,4.5,
                                    domain='time',
                                    MaxOnly=False,
                                    MotType = 'within')
        '''
        domain = kwargs.get('domain','time')
        MotType = kwargs.get('MotType','within')
        MaxOnly = kwargs.get('MaxOnly',False)

        if len(args) > 0:
            depths = np.array(args)
        else:
            depths = kwargs.get('depths',self.SoilObj.t_mid)

        out = self._AccVelDisp(which='Disp',depths=depths,domain=domain,
                               MotType=MotType, MaxOnly=MaxOnly)
        if len(depths) == 1:
            out = np.ravel(out)
        return out


    def calcVel(self,*args,**kwargs):
        '''
        Calculates velocity using the _AccVelDisp method.

        Keyword argument:   Options: (Default listed first)

            domain        =   'time'  or   'freq'

            MotType       =   'within', 'outcrop' or 'incoming'

            MaxOnly       =   False or True

            depths        =   mid-depths of each layer or list of desired
                               These depths can also be listed as regular,
                               non-keyword arguments; see the example.

             Returns an array of velocities in units of m/s for time
                histories, m/s-s for frequency domain. Array is 2D if more
                than one depth is specified.
```

```
2199
2200
2201            Example:
2202                vel = DynamicObject.calcVel(2.0,4.5,
2203                                            domain='time',
2204                                            MaxOnly=False,
2205                                            MotType = 'within')
2206
2207
2208        '''
2209        domain = kwargs.get('domain','time')
2210        MotType = kwargs.get('MotType','within')
2211        MaxOnly = kwargs.get('MaxOnly',False)
2212        if len(args) > 0:
2213            depths = np.array(args)
2214        else:
2215            depths = kwargs.get('depths',self.SoilObj.t_mid)
2216
2217        out = self._AccVelDisp(which='Vel',depths=depths,domain=domain,
2218                               MotType=MotType, MaxOnly=MaxOnly)
2219        if len(depths) == 1:
2220            out = np.ravel(out)
2221        return out
2222
2223
2224    def calcAcc(self,*args,**kwargs):
2225        '''
2226        Calculates acceleration using the _AccVelDisp method.
2227
2228        Keyword argument:   Options: (Default listed first)
2229
2230            domain        =   'time'  or   'freq'
2231
2232            MotType       =   'within', 'outcrop' or 'incoming'
2233
2234            MaxOnly       =   False or True
2235
2236            depths        =   mid-depths of each layer or list of desired
2237                                These depths can also be listed as regular,
2238                                non-keyword arguments; see the example.
2239
2240            Returns an array of accelerations.
2241
2242            Example:
2243                acc = DynamicObject.calcAcc(2.0,4.5,
2244                                            domain='time',
2245                                            MaxOnly=False,
2246                                            MotType = 'within')
2247
2248            Returns an array of velocities in units of m/s/s for time
2249                histories, m/s/s-s for frequency domain. Array is 2D if more
2250                than one depth is specified.
2251        '''
2252        domain = kwargs.get('domain','time')
2253        MotType = kwargs.get('MotType','within')
2254        MaxOnly = kwargs.get('MaxOnly',False)
2255        smooth = kwargs.get('smooth',False)
```

```
        window = kwargs.get('window',20.)
        if len(args) > 0:
            depths = np.array(args)
        else:
            depths = kwargs.get('depths',self.SoilObj.t_mid)

        out = self._AccVelDisp(which='Acc',depths=depths,domain=domain,
                               MotType=MotType, MaxOnly=MaxOnly, smooth=smooth,
                               window=window)
        if len(depths) == 1:
            out = np.ravel(out)
        return out


    def TF(self,*args, **kwargs):
        '''
        Calculates the transfer function between two layers in the frequency
            domain.

        Positional Arguments:
            List of two layers. A 1 refers to the top layer.
                                  0 refers to bedrock.
        Keyword Arguments:
            MotType = 'outcrop'[default] or 'within',
                      'rocktosurf','rocktobase'
        Output
            tf  = the transfer function.
        '''
        args = list(args)
        layer1 = args[0]
        layer2 = args[1]
        MotType = kwargs.get('MotType','outcrop')
        A = self.A
        B = self.B
        layer1 -= 1
        layer2 -= 1

        if MotType == 'rocktosurf':
            tf = np.abs((A[0,:] + B[0,:])/(2 * A[-1,:]))
        elif MotType == 'rocktobase':
            tf = np.abs((A[-1,:] + B[-1,:])/(2 * A[-1,:]))
        elif MotType == 'outcrop':
            tf = np.abs(A[layer1,:]/A[layer2,:])
        else:
            tf = np.abs((A[layer1,:]+B[layer1,:])/
                        (A[layer2,:]+B[layer2,:]))
        return tf



    def calcTF(self,*args, **kwargs):
        '''
        Calculates the transfer function between two layers in the frequency
            domain.

        Positional Arguments:
            List of two layers. A 1 refers to the top layer.
```

```
                                0 refers to bedrock.
        Keyword Arguments:
            MotType = Tuple of size two with values of
                          'outcrop' or 'within', [default]

        Output
            tf  = the transfer function.

        Example:
            tf = self.calcTF(1,0,MotType=('within', 'outcrop'))
        '''
        args = list(args)
        layer1 = args[0]
        layer2 = args[1]
        MotType = kwargs.get('MotType','outcrop')
        A = self.A
        B = self.B
        layer1 -= 1
        layer2 -= 1

        if MotType[0] == 'outcrop':
            num = 2 * A[layer1,:]
        else:
            num = A[layer1,:] + B[layer1,:]

        if MotType[1] == 'outcrop':
            den = 2 * A[layer2,:]
        else:
            den = A[layer2,:] + B[layer2,:]

        return np.abs(num/den)



    def _PiecewiseExact(self,acc_g,w,dt,zeta,**kwargs):
        '''
        Determines the spectral response of an SDOF structure.

        Uses Duhamel's integral. *** This is currently really slow!!!***

        Positional Arguments:
            acc_g = ground acceleration in units of g

            w = natural frequency of the SDOF structure.
                To return an acceleration response vector,
                input a vector of frequencies here.

            dt = sampling interval or time interval of the input acceleration

            zeta = the damping ratio of the structure = c/c_crit (decimal)

        Keyword Arguments:

            returnextra    If True, returns displacement, velocity, and
                               total acceleration. Only useful for single
                               value of w. Else, returns Spectral
                               Acceleration. Default is False.
```

```
        Code from rsc.c at
        <http://pubs.usgs.gov/of/2006/1369/ResponseSpectra.zip>
        This would be a good function to move to C.
        '''
        returnextra = kwargs.get('returnextra', False)
        a = np.float64(np.ravel(acc_g) * 9.80665) # m/s/s
        w = np.float64(np.ravel(w))
        dt = np.float64(dt)
        z = np.float64(zeta)


        root = np.sqrt(1. - z ** 2)
        wd = w * root
        sine = np.sin(wd * dt)
        cosn = np.cos(wd * dt)
        expo = np.exp(-z * w * dt)

        zratio = z / root
        blue = 2. * z / (w ** 3 * dt)
        yell = (2 * z ** 2 - 1) / (w ** 2 * dt)

        a11 = expo * (zratio * sine + cosn)
        a12 = expo * sine / wd
        a21 = -w / root * expo * sine
        a22 = expo * (cosn - zratio * sine)

        b11 = (expo * ((yell + z / w) * (sine / wd)
            + (blue + 1. / w ** 2) * cosn) - blue)
        b12 = (-expo * (yell * sine / wd + blue * cosn)
            - (1. / w ** 2) + blue)
        b21 = (expo * ((yell + z / w) * (cosn - zratio * sine)
            -(blue + 1. / w ** 2) * (wd * sine + z * w * cosn)) +
            1. / (w ** 2 * dt))
        b22 = (-expo * (yell * (cosn - zratio * sine) -
            blue * (wd * sine + z * w * cosn)) -
            1. / (w ** 2 * dt))
        if np.isscalar(w):
            u = np.zeros([1,len(a)], dtype=np.float64)
            v = np.zeros([1,len(a)], dtype=np.float64)
            for i in xrange(len(a) - 1):
                u[:,i+1] = (a11 * u[:,i] + a12 * v[:,i] + b11 * a[i] +
                            b12 * a[i+1])
                v[:,i+1] = (a21 * u[:,i] + a22 * v[:,i] + b21 * a[i] +
                            b22 * a[i+1])
            TA = - (2 * z * w * v + w ** 2 * u) / 9.80665
        else:
            u = np.zeros([len(w),len(a)], dtype=np.float64)
            v = np.zeros([len(w),len(a)], dtype=np.float64)
            for i in xrange(len(a) - 1):
                u[:,i+1] = (a11 * u[:,i] + a12 * v[:,i] + b11 * a[i] +
                            b12 * a[i+1])
                v[:,i+1] = (a21 * u[:,i] + a22 * v[:,i] + b21 * a[i] +
                            b22 * a[i+1])
            TA = - (2 * z * w[:,np.newaxis] * v +
                    w[:,np.newaxis] ** 2 * u) / 9.80665
```

```
2427            if returnextra:
2428                return np.ravel(u), np.ravel(v), np.ravel(TA)
2429            else:
2430                Sa = np.max(np.abs(TA),axis=1)
2431                return Sa
2432
2433
2434        def StressRedCoef(self, *args, **kwargs):
2435            '''
2436            Deprecated name. Use calc_r_d instead.
2437            '''
2438            print('Function name deprecated.' +
2439                  '   Use "calc_r_d" instead of "StressRedCoef"')
2440            if kwargs.get('returns', False):
2441                rd = self.calc_r_d(*args, **kwargs)
2442                return rd
2443            else:
2444                self.calc_r_d(*args, **kwargs)
2445
2446
2447        def calc_r_d(self, *args, **kwargs):
2448            '''
2449            Calculates the stress reduction coefficient for the profile and
2450            ground motion.
2451
2452            Positional Arguments:
2453            Depths at which the r_d coefficient should be calculated.
2454
2455            Keyword Arguments:       Definition/Default
2456            depths                   Default is the mid-depth of each layer.
2457                                         Disregarded if positional arguments
2458                                         are used.
2459
2460            returns                  False [default] or True.
2461
2462            If returns == True, stress reduction coefficient (r_d) is returned.
2463            This will return the calculated values.
2464
2465            If the mid-layer depths are used, the self.rd data attribute is created
2466            by this method.
2467            '''
2468            if len(args) > 0:
2469                depths = np.array(args)
2470            else:
2471                depths = kwargs.get('depths',self.SoilObj.t_mid)
2472            returns = kwargs.get('returns',False)
2473            if np.equal(depths, self.SoilObj.t_mid).all():
2474                same = True
2475                sigv = self.SoilObj.sigv
2476            else:
2477                pdb.set_trace()
2478                same = False
2479                sigv = self.SoilObj.calcInsituStress(*depths,out='total')
2480
2481            taumax,gammax = self._calcTauGam(depths=depths,domain='time',
2482                                MaxOnly=True,returns=True)
2483            amax = self.calcAcc(depths=[0.],MaxOnly=True,MotType='outcrop',
```

```
                    domain='time')

        rd = taumax / (amax * sigv)
        if same == True:
            self.rd = rd
        if returns == True:
            return rd


    def RespSpec(self, *args, **kwargs):
        '''
        Calculates the response spectra for a given depth(s).

        Positional Arguments:
            Depths at which RS is to be calculated. If none are given,
                it will calculate the response spectra for
                all mid-depths (very slow).

        Keyword Arguments:     Values:
        depths              an array of depths. Disregarded if the
                                 positional arguments are used.

        damping             any damping value as decimal, default is 0.05

        w                   an array of frequencies. Computationally
                              intensive if this array is large.
        method             'piecewiseexact' is the only one implemented now.

        MotType       =   'within', 'outcrop' or 'incoming'

        Returns Sa, w

        This method calls the self._PiecewiseExact method.
        '''
        if len(args) > 0:
            depths = np.array(args)
        else:
            depths = kwargs.get('depths',self.SoilObj.t_mid)
        damping = kwargs.get('damping',0.05)
        w = kwargs.get('w',np.arange(0.001,600,1.2))
        method = kwargs.get('method', 'piecewiseexact')
        MotType = kwargs.get('MotType','within')

        if method == 'piecewiseexact':
            acc = self.calcAcc(depths=depths,
                           domain='time',
                           MaxOnly=False,
                           MotType = MotType)
        else:
            print('Did not understand the method. Please try again.')
            return
        SA = np.zeros([len(depths),len(w)],dtype=float)
        for i in xrange(len(depths)):
            if len(depths) == 1:
                SA = self._PiecewiseExact(acc, w, self.MotObj.dt,damping)
            else:
                SA[i,:] = self._PiecewiseExact(acc[i,:], w,
```

```
                                        self.MotObj.dt,damping)
        if SA.shape[0] == 1:
            SA = np.ravel(SA)
        return SA,w


    def ASCE_RS_ratios(self,**kwargs):
        '''
        Calculates the ratio between the 5% damped surface
        response spectra of surface ground motions to input base motions.

        See ASCE 7-05, sec. 21.1.3.

        Keyword Arguments:
            T       array of periods

        Returns the surface to base response ratio and the corresponding
            array of periods.

        This method calls the self.RespSpec method and assumes 5% damping,
            'within' surface motion, and 'outcrop' base motion.

        Example:

            RS_ratio, T = self.ASCE_RS_ratios(T=np.logspace(-3,1,num=1000))
        '''
        T = kwargs.get('T',np.logspace(-3,1,num=1000))

        # Obtain the surface response spectrum
        surf,w = self.RespSpec(0.,damping=0.05,w=2.*np.pi/T,MotType='within')

        # Obtain the input base motion response spectrum
        basedepth = np.sum(self.SoilObj.t) + 0.001
        base,w = self.RespSpec(basedepth,damping=0.05,w=w,MotType='outcrop')

        ratio = surf / base

        return ratio,T


    def plot(self, *args, **kwargs):
        '''
        Outputs standard plots that illustrate the equivalent linear response.

        Positional Arguments:
            List the plots you want to see.
            Possible plots:
                mod         modulus profile
                damp        damping profile
                maxacc      max acceleration profile
                csr         max stress ratio profile
                maxstrain   max strain profile
                dissen      dissipated energy profile
                                (calc'd only at mid points)
                strainratio strain ratio profile used to obtain average strain
                                from max
                cycles      number of equivalent cycles profile
```

```
            rd         stress reduction coefficient
            allprofile or all  outputs all


        For keywords arguments:
            dz =   For profiles: depth increments (in meters)

            acc =  an array containing the following:
                       [depth,'time' or 'freq','outcrop' or 'within']

            vel =  an array containing the following:
                       [depth,'time' or 'freq','outcrop' or 'within']

            disp = an array containing the following:
                       [depth,'time' or 'freq','outcrop' or 'within']

            tf   =  an array containing the following:
                       [toplayer, bottomlayer, 'outcrop' or 'within']
                       Use the layer numbers. If only one value is given,
                       the bottom layer is assumed to be bedrock.

            respspec = an array containing the following:
                       [depth,damping,'outcrop' or 'within']

            plottype = 'loglog' [default], 'semilogx',
                           'linear' (only for freq domain)

            period = False [default] or True. Specifies if the horizontal axis
                    should show period or frequency.

            save = False (default) or True. For now, if save = True, then the
                   figures will also be closed. (Haven't figured it out yet.)

            extension = '.png' (default) or any other that matplotlib handles

            syms1  =  matplotlib line symbols. Default is '-b'

            syms2  =  matplotlib line symbols. Default is 'ob'

            linelabel =  label for plot. Default is ''


        Example:

            DynamicsObj.plot('mod','maxacc', save=True, dz=1)
        '''
        syms1 = kwargs.get('syms1','-b')
        syms2 = kwargs.get('syms2','ob')
        linelabel = kwargs.get('linelabel','')
        t = self.SoilObj.t
        t_mid = self.SoilObj.t_mid
        basedepth = np.sum(t,dtype=float)

        if 'all' in args:
            args = ('mod','damp','maxacc','csr','maxstrain',
                       'dissen','strainratio','cycles', 'rd')
```

```
        if 'allprofile' in args:
            args = ('mod','damp','maxacc','csr','maxstrain',
                        'dissen','strainratio','cycles', 'rd')

        dz = kwargs.get('dz')
        acc = kwargs.get('acc')
        vel = kwargs.get('vel')
        disp = kwargs.get('disp')
        tf = kwargs.get('tf')
        respspec = kwargs.get('respspec')
        period = kwargs.get('period',False)
        plottype = kwargs.get('plottype','loglog')
        save = kwargs.get('save',False)
        ext = kwargs.get('extension','.png')
        return_fig = kwargs.get('fig')
        if return_fig != None:
            plt.figure(return_fig.number)

        if dz:
            depths = np.arange(0.,basedepth+dz,dz,dtype=float)
        else:
            depths = t_mid


        def profilecreator(array, **kwargs):
            '''
            Makes points at the top and bottom of each layer to make plotting
            better.
            '''
            depth = kwargs.get('depth',False)
            if depth == True:
                output = [0]
                for i in xrange(len(array)):
                    if output[i] == 0:
                        output.append(array[i])
                    else:
                        output.append(array[i-1])
                        output.append(array[i])
                output.append(array[-1])
                output.append(array[-1]*1.05)
            else:
                output = []
                for i in xrange(len(array)):
                    output.append(array[i])
                    output.append(array[i])
            return np.array(output)


        def profinterp(depth2,values2,newdepths):
            newvalues = []
            for i in xrange(len(newdepths)):
                for j in xrange(1,len(depth2),2):
                    if (newdepths[i] <= depth2[j])&(newdepths[i] >= depth2[j-1]):
                        newvalues.append(values2[j])
                        break
            return newvalues
```

```
2712
2713        depth2 = profilecreator(np.cumsum(t),depth=True)
2714
2715        if 'mod' in args:
2716            mod = profilecreator(self.G)
2717            plt.figure('Shear Modulus Profile')
2718            plt.plot(mod,depth2,syms1)
2719            plt.plot(self.G[:-1],t_mid, syms2)
2720            plt.xlabel(r'Degraded Shear Modulus, $G$ $(kPa)$')
2721            plt.ylabel(r'Depth ($m$)')
2722            plt.title(self.SoilObj.filename+'-'+self.MotObj.filename)
2723            [xmin, xmax, ymin, ymax] = plt.axis()
2724            plt.axis([xmin, np.max(self.G[:-1])*1.1, ymax, ymin])
2725            fig = plt.gcf()
2726
2727        if 'damp' in args:
2728            damp = profilecreator(self.D)
2729            plt.figure('Damping Profile')
2730            plt.plot(damp*100,depth2, syms1)
2731            plt.plot(self.D[:-1]*100,t_mid,'or')
2732            plt.xlabel(r'Degraded Damping, $D$ ($\%$)')
2733            plt.ylabel(r'Depth ($m$)')
2734            plt.title(self.SoilObj.filename+'-'+self.MotObj.filename)
2735            [xmin, xmax, ymin, ymax] = plt.axis()
2736            plt.axis([xmin, xmax, ymax, ymin])
2737            fig = plt.gcf()
2738
2739        if 'dissen' in args:
2740            if self.dissEn != None:
2741                dissen = self.dissEn
2742            else:
2743                self.calcNeq()
2744                dissen = self.dissEn
2745            dissen2 = profilecreator(dissen)
2746            plt.figure('DissEn')
2747            plt.plot(dissen2,depth2[:-2], syms1)
2748            plt.plot(self.dissEn,t_mid,'or')
2749            plt.xlabel(r'Dissipated Energy $(kJ/m^3)$')
2750            plt.ylabel(r'Depth ($m$)')
2751            plt.title(self.SoilObj.filename+'-'+self.MotObj.filename)
2752            [xmin, xmax, ymin, ymax] = plt.axis()
2753            plt.axis([xmin, xmax, ymax, ymin])
2754            fig = plt.gcf()
2755
2756        if 'cycles' in args:
2757            try:
2758                cyc = self.N_eq
2759            except AttributeError:
2760                self.calcNeq()
2761                cyc = self.N_eq
2762            cyc2 = profilecreator(cyc)
2763            plt.figure('NumCyc')
2764            plt.plot(cyc2,depth2[:-2], syms1)
2765            plt.plot(self.N_eq,t_mid,'or')
2766            plt.xlabel(
2767                'Number of Equivalent Cycles Based on Dissipated Energy')
2768            plt.ylabel(r'Depth ($m$)')
```

```
2769            plt.title(self.SoilObj.filename+'-'+self.MotObj.filename)
2770            [xmin, xmax, ymin, ymax] = plt.axis()
2771            plt.axis([xmin, xmax, ymax, ymin])
2772            fig = plt.gcf()
2773
2774        if 'maxacc' in args:
2775            accel = self.calcAcc(domain='time',
2776                           MotType='within',
2777                           MaxOnly=True,
2778                           depths=depths)
2779            plt.figure('AccProfile')
2780            plt.plot(accel,depths, syms1)
2781            plt.xlabel(r'Max Acceleration ($g$)')
2782            plt.ylabel(r'Depth ($m$)')
2783            plt.title(self.SoilObj.filename+'-'+self.MotObj.filename)
2784            [xmin, xmax, ymin, ymax] = plt.axis()
2785            plt.axis([xmin, xmax, ymax, ymin])
2786            fig = plt.gcf()
2787
2788        if ('maxstrain' in args) or ('csr' in args):
2789            tau,gam = self._calcTauGam(domain='time',
2790                           returns=True,
2791                           MaxOnly=True,
2792                           depths=depths)
2793
2794        if 'maxstrain' in args:
2795            plt.figure('MaxStrain')
2796            plt.plot(gam*100,depths, syms1)
2797            plt.xlabel(r'Max Shear Strain, $\gamma$, ($\%$)')
2798            plt.ylabel(r'Depth ($m$)')
2799            plt.title(self.SoilObj.filename+'-'+self.MotObj.filename)
2800            [xmin, xmax, ymin, ymax] = plt.axis()
2801            plt.axis([xmin, xmax, ymax, ymin])
2802            fig = plt.gcf()
2803
2804        if 'csr' in args:
2805            Gnew = np.array(profinterp(depth2,mod,depths))
2806            sigveff = self.SoilObj.calcInsituStress(depths,out='eff')
2807            CSR = gam * Gnew / sigveff
2808            plt.figure('CSR')
2809            plt.plot(CSR,depths, syms1)
2810            plt.xlabel(r'Max Cyclic Stress Ratio, $CSR$')
2811            plt.ylabel(r'Depth ($m$)')
2812            plt.title(self.SoilObj.filename+'-'+self.MotObj.filename)
2813            [xmin, xmax, ymin, ymax] = plt.axis()
2814            plt.axis([xmin, xmax, ymax, ymin])
2815            fig = plt.gcf()
2816
2817        if 'strainratio' in args:
2818            ratio1 = np.ones(t.shape)*np.array(self.ratio)
2819            ratio = profilecreator(ratio1)
2820            plt.figure('StrainRatio')
2821            plt.plot(ratio,depth2[:-2], syms1)
2822    #          plt.plot(ratio1,t_mid,'or')
2823            plt.xlabel('Strain Ratio')
2824            plt.ylabel(r'Depth ($m$)')
2825            plt.title(self.SoilObj.filename+'-'+self.MotObj.filename)
```

```
            [xmin, xmax, ymin, ymax] = plt.axis()
            plt.axis([xmin, xmax, ymax, ymin])
            fig = plt.gcf()


        # Specific Depth plots
        def plothelpers(x,y,domain,plottype,period,figure,dt=True,label=None):
            '''
             x should be frequency or time, not angular frequency
             If dt is True, the response is multiplied by dt.
            '''
            plt.figure(figure.number)
            if domain == 'freq':
                if dt:
                    y = np.abs(y) * self.MotObj.dt

                if period:
                    x = 1./x
                    plt.xlabel('Period (s)')
                    if plottype == 'loglog':
                        plt.loglog(x,y,label=label)

                    elif plottype == 'semilogx':
                        plt.semilogx(x,y,label=label)
                    else:
                        plt.plot(x,y,label=label)
                else:
                    plt.xlabel(r'Frequency ($Hz$)')
                    if plottype == 'loglog':
                        plt.loglog(x,y,label=label)
                    elif plottype == 'semilogx':
                        plt.semilogx(x,y,label=label)
                    else:
                        plt.plot(x,y,label=label)
            else:
                plt.plot(x,y,label=label)
                plt.xlabel('Time (s)')
            curr_fig = plt.gcf()
            return curr_fig

        if acc:
            if acc[2] == 'outcrop':
                mot = 'outcrop'
            else:
                mot = 'within'
            Accel = np.ravel(self.calcAcc(acc[0],domain=acc[1],MotType=mot))
            if acc[1] == 'time':
                x = self.MotObj.time
            else:
                x = self.MotObj.f

            if return_fig:
                plt.figure(return_fig.number)
            else:
                plt.figure(r'Acceleration at z = %s'%str(acc[0]))
            if acc[1] == 'time':
                plt.ylabel(r'Acceleration ($g$)')
```

```
            else:
                plt.ylabel(r'Acceleration ($\frac{m}{s^2}$ $-s$)'
                           %str(acc[0]) )
            plt.title(self.SoilObj.filename+' - '+self.MotObj.filename)
            fig = plt.gcf()
            label = '%s m'%str(acc[0])
            fig = plothelpers(x,Accel,acc[1],plottype,period,fig,label=label)

        if vel:
            if vel[2] == 'outcrop':
                mot = 'outcrop'
            else:
                mot = 'within'
            Veloc = np.ravel(self.calcVel(vel[0],domain=vel[1],MotType=mot))
            if vel[1] == 'time':
                x = self.MotObj.time
            else:
                x = self.MotObj.f

            if return_fig:
                plt.figure(return_fig.number)
            else:
                plt.figure(r'Velocity at z = %s'%str(vel[0]))
            if vel[1] == 'time':
                plt.ylabel(r'Velocity ($\frac{m}{s}$)')
            else:
                plt.ylabel(r'Velocity ($\frac{m}{s}$ $-s$)')
            plt.title(self.SoilObj.filename+' - '+self.MotObj.filename)
            fig = plt.gcf()
            label = '%s m'%str(vel[0])
            fig = plothelpers(x,Veloc,vel[1],plottype,period,fig,label=label)

        if disp:
            if disp[2] == 'outcrop':
                mot = 'outcrop'
            else:
                mot = 'within'
            Disp = np.ravel(self.calcDisp(disp[0],domain=disp[1],MotType=mot))
            if disp[1] == 'time':
                x = self.MotObj.time
            else:
                x = self.MotObj.f

            if return_fig:
                plt.figure(return_fig.number)
            else:
                plt.figure(r'Displacement at z = %s'%str(disp[0]))
            if disp[1] == 'time':
                plt.ylabel(r'Displacement ($m$)')
            else:
                plt.ylabel(r'Displacement ($m$ $-s$)'
                           %str(disp[0]) )
            plt.title(self.SoilObj.filename+' - '+self.MotObj.filename)
            fig = plt.gcf()
            label = '%s m'%str(disp[0])
            fig = plothelpers(x,Disp,disp[1],plottype,period,fig,label=label)

```

```
2940
2941        if tf:
2942            x = self.MotObj.f
2943            if len(tf) == 1:
2944                try:
2945                    layer1 = int(tf[0])
2946                    layer2 = 'Bedrock'
2947                except:
2948                    print("I didn't understand the transfer function input")
2949 #                  break
2950                y = self.TF(layer1,0)
2951            elif len(tf) == 2:
2952                try:
2953                    layer1 = int(tf[0])
2954                    layer2 = int(tf[1])
2955                    y = self.TF(layer1,layer2)
2956                except:
2957                    print ("Assuming the second value of the TF " +
2958                           "specifies motion type")
2959                    mot = tf[1]
2960                    layer2 = 'Bedrock'
2961                    y = self.TF(layer1,0,MotType=mot)
2962            else:
2963                layer1 = int(tf[0])
2964                layer2 = int(tf[1])
2965                mot = tf[2]
2966                y = self.TF(layer1,layer2,MotType=mot)
2967            plt.figure('Transfer Function' )
2968            plt.ylabel(r'Transfer Function Between Layers %s and %s'
2969                       %(str(layer1),str(layer2)))
2970            plt.title(self.SoilObj.filename+' - '+self.MotObj.filename)
2971            fig = plt.gcf()
2972            label = 'Layer #%s'%str(layer1)
2973            fig = plothelpers(x,y,'freq',plottype,period,fig,dt=False,
2974                              label=label)
2975
2976
2977        if respspec:
2978            plt.figure('Response Spectrum' )
2979            plt.ylabel(r'Response Spectrum at a Depth of %s m'
2980                       %str(respspec[0]))
2981            plt.title(self.SoilObj.filename+' - '+self.MotObj.filename)
2982            y,w = self.RespSpec(respspec[0],
2983                                damping=respspec[1],
2984                                MotType=respspec[2],
2985                                w=self.MotObj.w[1:])
2986            x = self.MotObj.f[1:]
2987            y = np.ravel(y)
2988            fig = plt.gcf()
2989            label = 'Depth: %s m, Damping: %s'%(str(respspec[0]),str(respspec[1]))
2990            fig = plothelpers(x,y,'freq',plottype,period,fig,dt=False,
2991                              label=label)
2992
2993        if save == True:
2994            import matplotlib
2995            fig=[manager.canvas.figure for manager in
2996                       matplotlib._pylab_helpers.Gcf.get_all_fig_managers()]
```

```
            for i, figure in enumerate(fig):
                figure.savefig(self.SoilObj.profile_name[:5] + '_' +
                               self.MotObj.filename[:5]+'-'+
                               figure._label+ext)
        return fig

    def calcPMNeq(self, *args, **kwargs):
        '''
        Calculates Neq using the traditional (Seed et al. 1975) Palmgren-Miner
        Implementation.
        '''
        VectorSum = kwargs.get('VectorSum', False)
        #m = 0.35 # I've assigned m all the way up at line 24
        #cutoff = 0.3
        m = kwargs.get('m', 0.34)
        self.pm_m = m
        cutoff = kwargs.get('cutoff', 0.3)
        self.pm_cutoff = cutoff
        self.pm_Neq = []
        self.pm_Discards = []

        #returns = kwargs.get('returns',True)

        amax = self.calcAcc(depths=[0.],MaxOnly=False,MotType='outcrop',
                            domain='time')
        mm = m
        mymax = np.max(np.abs(amax))
        amaxn = amax / mymax
        # Now I need to get the peak values for the positive and negatives parts.
        ind = np.where(np.sign(amaxn[:-1]) != np.sign(amaxn[1:]))[0] # Index of each zero
             crossing
        myN = 0.
        discard = 0.
        Ncyc = 0.5 # I'm assuming the result of the mean crossing is one half cycle, change
             if needed.
        for i, val in enumerate(ind):
            try:
                if i == 0:
                    peak = np.max(np.abs(amaxn[0:val]))
                    #sign = np.sign(np.mean(amaxn[0:val]))
                else:
                    peak = np.max(np.abs(amaxn[ind[i-1]:val]))
                    #sign = np.sign(np.mean(amaxn[ind[i-1]:val]))
            except ValueError:
                continue
            if peak < cutoff:
                discard += Ncyc * np.exp(np.log(peak / 0.65) / mm)
                continue
            else:
                myN += Ncyc * np.exp(np.log(peak / 0.65) / mm)
        self.pm_Neq.append(myN) # Just summing here since Ncyc is 0.5
        self.pm_Discards.append(discard)
        if VectorSum: # This is a la Liu et al. 2001
            P2P = False
            VS = np.sqrt(self.amax_surf[0] ** 2 + self.amax_surf[1] ** 2)
            VS = VS / np.max(VS)
            alt = False
```

```
3052            if alt:
3053                import CycleCounting as cc
3054                CC = cc.Load(VS)
3055                Peaks, Cycles = CC.calcPeak2Peak(AmpType='ASTM')
3056                self.PMNeqVS = self.PMFloor(Peaks, Cycles)
3057                self.PMNeqVSdis = 0. # Not a good way to do this using the alt method.
3058            else:
3059                if P2P:
3060                    import CycleCounting as cc
3061                    CC = cc.Load(VS)
3062                    Peaks, Cycles = CC.calcPeak2Peak(AmpType='ASTM')
3063                    ind1 = np.where(Peaks >= cutoff)
3064                    ind2 = np.where(Peaks < cutoff)
3065                    self.PMNeqVS = np.sum(Cycles[ind1] * np.exp(np.log(
3066                                        Peaks[ind1] / 0.65)/m))
3067                    self.PMNeqVSdis = np.sum(Cycles[ind2] * np.exp(np.log(
3068                                        Peaks[ind2] / 0.65)/m))
3069                else:
3070                    # Find the location of the peaks:
3071                    dVS = np.diff(VS)
3072                    ind0 = np.where(np.sign(dVS[1:]) != np.sign(dVS[:-1]))[0] + 1
3073                    Peaks = VS[ind0]
3074                    #import matplotlib.pyplot as plt
3075                    #plt.plot(VS)
3076                    #plt.plot(ind0, Peaks, 'or')
3077                    #plt.show()
3078                    #pdb.set_trace()
3079                    ind1 = np.where(Peaks >= cutoff)
3080                    ind2 = np.where(Peaks < cutoff)
3081                    self.PMNeqVS = np.sum(0.5 * np.exp(np.log(
3082                                        Peaks[ind1] / 0.65)/m))
3083                    self.PMNeqVSdis = np.sum(0.5 * np.exp(np.log(
3084                                        Peaks[ind2] / 0.65)/m))
3085                    #print(self.PMNeqVS, self.PMNeqVSdis)
3086        return myN, discard
3087
3088
3089 class DSvariables(object):
3090     '''
3091     A simple object to hold data for Darendeli and Stokoe's degradation
3092     curve options.
3093     '''
3094     def __init__(self,kwargs,SoilObj):
3095         # Degradation Curve Overrides and Special Values:
3096         self.sigmeff = kwargs.get('DCO_sigmeff',SoilObj.sigmeff)
3097         self.PI = kwargs.get('DCO_PI',SoilObj.PI)
3098         PI = self.PI
3099         if (SoilObj.OCR is None) or (len(SoilObj.OCR) == 0):
3100             self.OCR = kwargs.get('DS_OCR',
3101                                   np.ones(len(PI),dtype=float)) #Non-over consolidated as
                                          default
3102         else:
3103             self.OCR = SoilObj.OCR
3104         self.soiltype = kwargs.get('DS_soil', SoilObj.DS_soil)
3105         if np.any(self.soiltype):# Not sure if this is right. Before: if self.soiltype ==
               None:
3106             pass
```

```
3107            else:
3108                self.soiltype = 0. * np.ones(len(PI),dtype=float) #General default
3109            self.N = kwargs.get('DS_N', SoilObj.DS_N)
3110            if np.any(self.N):
3111                pass
3112            else:
3113                self.N = 10. * np.ones(len(PI),dtype=float) # 10 cycles default
3114            self.freq = kwargs.get('DS_freq', SoilObj.DS_freq)
3115            if np.any(self.freq):
3116                pass
3117            else:
3118                self.freq = 1. * np.ones(len(PI),dtype=float) # 1 Hz default
3119
3120    def nextpow2(i):
3121        '''
3122        Returns the the log-base-2 of the next power of 2 greater than the input
3123        value.
3124
3125        Example:
3126        >>> y = nextpow2(5)
3127        >>> print y
3128        3
3129        >>> 2**nextpow2(5)
3130        8
3131        '''
3132        n = 2
3133        while n < i:
3134            n = n * 2
3135        return int(round(np.log2(n)))
3136
3137    def calcAriasRatio(motion, dt, **kwargs):
3138        '''
3139        Calculates the following:
3140        a_rms / (max(abs(motion)) )
3141
3142        It should be unitless.
3143
3144        Although it uses the rms acceleration, any type of motion can be used.
3145
3146        Positional Arguments:
3147            motion     can be a 2D array of motions (each motion is a row)
3148                           or a 1D array of motions
3149
3150            dt         the time step of the motion(s) in seconds, can be an array
3151
3152        Keyword Arguments:
3153            returnall  If True, returns Ar and PeakTimeRatio, else, returns only
3154                           Ar.  Default is False.
3155            duration   '5to95' [Default], '5to75', or 'total'. If '5to95' uses from 5% to
3156                           95% of the motion squared (Husid?) plot.
3157                           This keyword is passed directly to the calcRmsAccel
3158                           function.
3159        '''
3160        returnall = kwargs.get('returnall',False)
3161
3162        if 1 in motion.shape or len(motion.shape) == 1:
3163            NumMotions = 1
```

```
    else:
        NumMotions = np.min(motion.shape)

    if np.isscalar(dt) & NumMotions!=1:
        dt = np.ones([NumMotions], dtype=float) * dt
    if NumMotions == 1:
        if np.isscalar(dt):
            pass
        else:
            pdb.set_trace()
            dt = dt[0]

    Ar = np.zeros([NumMotions,], dtype=float)
    PTR = np.zeros([NumMotions,], dtype=float)
    if returnall:
        if NumMotions == 1:
            RMS, PTR = calcRmsAcc(motion, dt, **kwargs)
            Ar = (RMS / np.max(np.abs(motion))) # Assume data is in rows
        else:
            for i in xrange(NumMotions):
#               pdb.set_trace()

                RMS, PTR[i] = calcRmsAcc(motion[i,:], dt, **kwargs)
                Ar[i] = (RMS / np.max(np.abs(motion[i,:]))) # Assume data is in rows
        return Ar, PTR
    else:
        if NumMotions == 1:
            Ar = (calcRmsAcc(motion, dt, **kwargs) /
                    np.max(np.abs(motion))) # Assume data is in rows
        else:
            for i in xrange(NumMotions):
                pdb.set_trace()
                Ar[i] = (calcRmsAcc(motion[i,:], dt[i], **kwargs)[0] /
                    np.max(np.abs(motion[i,:]))) # Assume data is in rows
        return Ar


def calcRmsAcc(motion, dt, **kwargs):
    '''
    Calculates the rms acceleration.

    Positional Arguments:
        motion     a 1D array of a motion, for now.

        dt         the time step of the motion(s) in seconds, a scalar

    Keyword Arguments:
        duration       '5to95' [default] Uses the duration from 5-95% of
                          the area under the motion squared
                       '5to75'
                      'total'      Uses the entire time of the input motion

        returnall  If True, returns Ar and PeakTimeRatio, else, returns only
                       Ar.  Default is False.

    '''
    returnall = kwargs.get('returnall',False)
```

```
    motion = np.ravel(motion)
    durationtype = kwargs.get('duration','5to95')
    NPTS = len(motion)
    if np.isscalar(dt):
        pass
    else:
        dt = dt[0]
    time = np.linspace(0,(NPTS-1) * dt,NPTS)
    asqr = np.cumsum(motion ** 2) * dt
    indPeak = np.where(np.abs(motion) == np.max(np.abs(motion)))

    if durationtype == '5to95':
        int1 = interp1d( asqr, time, kind='linear')
        T = int1(0.95 * asqr[-1]) - int1(0.05 * asqr[-1])
        PeakTimeRatio = (time[indPeak] - int1(0.05 * asqr[-1])) / T
    elif durationtype == '5to75':
        int1 = interp1d( asqr, time, kind='linear')
        T = int1(0.75 * asqr[-1]) - int1(0.05 * asqr[-1])
        PeakTimeRatio = (time[indPeak] - int1(0.05 * asqr[-1])) / T
    else:
        T = time[-1]
        PeakTimeRatio = time[indPeak] / T

    if returnall:
        return np.sqrt(asqr[-1] / T), PeakTimeRatio
    else:
        return np.sqrt(asqr[-1] / T)

def input_soil_profile(csvprofile):
    '''
    Creates a soil profile object using Lake's format of the csv.
    The new format is simpler and doesn't contain any options for output.

    Positional Argument:
        Name or path of a profile .csv file.
    '''
    layerno = []
    layername = []
    unit_wt = []
    t = []
    ko = []
    phi = []
    Vs = []
    PI = []
    N160 = []
    OCR = []
    DS_soil = []
    DS_N = []
    DS_freq = []

    count = 0
    filein = open(csvprofile, "rU")
    reader = csv.reader(filein)
    for line in reader:
        count += 1
        if count == 1:
            Profile_name = str(line[1])
```

```
        elif count == 2:
            units = int(line[1])
        elif count == 3:
            gwt = float(line[1])
        elif count == 4:
            loc_mot = int(line[1])
        elif count == 5:
            outcrop_mot = int(line[1])
        elif count == 6:
            pass
        else:
            layerno.append(int(line[0]))
            layername.append(line[1])
            unit_wt.append(float(line[2]))

            Vs.append(float(line[3]))
            try:
                t.append(float(line[4]))
            except (ValueError, IndexError):
                break # Notice I kick out if the thicknesses run out.
            try:
                PI.append(float(line[5]))
            except (ValueError,IndexError):
                PI.append(0.) # If there is no number, I assume 0.
            try:
               ko.append(float(line[6]))
            except (ValueError,IndexError):
               ko.append(0.) # If there is no number, I assume 0.
            try:
                phi.append(float(line[7]))
            except (ValueError,IndexError):
                phi.append(0.) # If there is no number, I assume 0.
            try:
                N160.append(float(line[8]))
            except (ValueError,IndexError):
                N160.append(0.) # If there is no number, I assume 0.
            try:
                OCR.append(float(line[9]))
            except (IndexError,ValueError):
                OCR.append(1.) # If there is no number, I assume 1.
            try:
                DS_soil.append(int(line[10]))
            except (IndexError,ValueError):
                DS_soil.append(0) # If there is no number, I assume 0, general soil.
            try:
                DS_N.append(int(line[11]))
            except (IndexError,ValueError):
                DS_N.append(10) # If there is no number, I assume 10.
            try:
                DS_freq.append(float(line[12]))
            except (IndexError,ValueError):
                DS_freq.append(1) # If there is no number, I assume 1.

    filein.close()

    # Convert to numpy arrays
    layerno = np.array(layerno)
```

```
    unit_wt = np.array(unit_wt)
    ko = np.array(ko)
    Vs = np.array(Vs)
    t = np.array(t)
    phi = np.array(phi)
    PI = np.array(PI)
    N160 = np.array(N160)
    OCR = np.array(OCR)
    DS_soil = np.array(DS_soil)
    DS_N = np.array(DS_N)
    DS_freq = np.array(DS_freq)

    # Convert all values to metric
    if units == 1:
        t = t * 0.3048 # Convert from ft to m
        Vs = Vs * 0.3048 # ft/s to m/s
        unit_wt = unit_wt * 9.81 / 62.4 # pcf to kN/m^3

    if Profile_name == '':
        Profile_name = os.path.basename(csvprofile)[:-4]

    # Create the SoilProfile object.
    sprofile = SoilProfile(Profile_name, unit_wt, Vs, t,
                        ko=ko, phi=phi, PI=PI, N160=N160,
                        layerno=layerno, layername=layername,
                        gwt=gwt, loc_mot=loc_mot,
                        outcrop_mot=outcrop_mot,
                         filename=csvprofile, OCR=OCR,
                         DS_soil=DS_soil, DS_N=DS_N, DS_freq=DS_freq)
    return sprofile
```

J.3 Shear Modulus Reduction and Damping Curves

The file `degrcurv.py` returns the shear modulus and damping degradation curves from Darendeli and Stokoe (2001) or Ishibashi and Zhang (1993).

```
,
import numpy as np
#straindef=np.logspace(-6,-0.004364805,num=10)
straindef = np.array([0.000001, 0.000003, 0.00001, 0.00003, 0.0001, 0.0003,
                    0.001, 0.003, 0.01, 0.99, 5])


def IZ_1993(gam=straindef, PI=0., sigm=100):
    ''' Returns the Ishibashi and Zhang (1993) values for G/Gmax
    and damping (decimal) for a given values of strain (as a decimal),
    Plasticity index, and initial mean effective confining stress (kPa).
    '''
    PI = float(PI)
```

```
    sigm = float(sigm)

    if PI == 0.:
        n = 0.0
    elif PI <= 15:
        n = 3.37e-6 * PI ** 1.404
    elif PI <= 70:
        n = 7e-7 * PI ** 1.976
    else:
        n = 2.7e-5 * PI ** 1.115


    K = 0.5 * (1 + np.tanh(np.log(((0.000102 + n) / gam) ** 0.492)))
    m = (0.272 * (1 - np.tanh(np.log((0.000556 / gam) ** 0.4))) *
                np.exp(-0.0145 * PI ** 1.3))
    Gratio = K * sigm ** m
    np.putmask(Gratio, Gratio>1, 1)
    #use np.copyto when updates to numpy 1.7 or use Gratio.clip(max=1)
    D = 0.1665 * (1 + np.exp(-0.0145 * PI ** 1.3)) * (0.586 * Gratio ** 2. -
                1.547 * Gratio + 1.)

    return gam, Gratio, D


def DS_2001(gam=straindef, PI=0., sigm=100., OCR=1., soil=0, N=15., frq=1.):
    '''
    Darendeli and Stokoe (2001) shear modulus reduction and damping curves.
    gam= shear strain (decimal)
    PI= Plasticity Index
    sigm = the initial mean effective confining stress (kPa)
    OCR = the overconsolidation ratio
    soil = the soil type indicator:
        0 for general
        1 for clean sand
        2 for sands with high fines contents
        3 for silts
        4 for clays
    N = the number of cycles
    frq = the frequency of loading
    Dmintype = 1 for 'Default', 0 for 'Green' (Suggest using Green's for now!!!!!)
                !!!Not Implemented !!!
    The output of the function is:
    gam = the same input shear strain as a decimal
    Gratio = G/Gmax = the shear modulus normalized by the small strain
        shear modulus.
    D = damping (decimal)
    '''
    PI = float(PI)
    sigm = float(sigm)
    OCR = float(OCR)
    N = float(N)
    frq = float(frq)
    soil = int(soil)
    phi1 = np.array([ 3.52E-02, 4.74E-02, 3.34E-02, 4.16E-02, 2.58E-02])
    phi2 = np.array([ 1.01E-03,-2.34E-03,-5.79E-05, 6.89E-04, 1.95E-03])
    phi3 = np.array([ 3.246E-01, 2.50E-01, 2.49E-01, 3.21E-01, 9.92E-02])
    phi4 = np.array([ 3.483E-01, 2.34E-01, 4.82E-01, 2.80E-01, 2.26E-01])
```

```
    phi5 = np.array([ 9.19E-01, 8.95E-01, 8.45E-01, 1.00E+00, 9.75E-01])
    phi6 = np.array([ 8.005E-01, 6.88E-01, 8.89E-01, 7.12E-01, 9.58E-01])
    phi7 = np.array([ 1.29E-02, 1.22E-02, 2.02E-02, 3.03E-03, 5.65E-03])
    phi8 = np.array([-1.069E-01, -1.00E-01,-1.00E-01,-1.00E-01,-1.00E-01])
    phi9 = np.array([-2.889E-01, -1.27E-01,-3.72E-01,-1.89E-01,-1.96E-01])
    phi10 = np.array([ 2.919E-01, 2.88E-01, 2.33E-01, 2.34E-01, 3.68E-01])
    phi11 = np.array([ 6.329E-01, 7.67E-01, 7.76E-01, 5.92E-01, 4.66E-01])
    phi12 = np.array([-5.66E-03, -2.83E-02,-2.94E-02,-7.67E-04, 2.23E-02])

    # Calculate the reference shear strain:
    gamr = ((phi1[soil] + phi2[soil] * PI * OCR ** phi3[soil])
            * (sigm / 101.325) ** phi4[soil]) # percent
    gam = gam * 100 # convert to percent

    # Calculate assorted variables:
    a = phi5[soil]
    b = phi11[soil] + phi12[soil] * np.log(N)
    C1 = -1.1143 * a ** 2 + 1.8618 * a + 0.2523
    C2 = 0.0805 * a ** 2 - 0.0710 * a - 0.0095
    C3 = -0.0005 * a ** 2 + 0.0002 * a + 0.0003
    # Calculate G/Gmax
    Gratio = 1. / (1. + (gam / gamr) ** a)


    Dmasinga1 = 100 / np.pi * (4 * (gam - gamr * np.log((gam + gamr)
               / gamr)) / (gam ** 2 / (gam + gamr)) - 2 )

    Dmasing = C1 * Dmasinga1 + C2 * Dmasinga1 ** 2 + C3 * Dmasinga1 ** 3

    Dmin = ((phi6[soil] + phi7[soil] * PI * OCR ** phi8[soil])
           * (sigm / 101.3) ** phi9[soil] * (1 + phi10[soil] * np.log(frq)))

    D = (b * (Gratio ** 0.1) * Dmasing + Dmin) / 100. #Return damping to a decimal


    return gam / 100, Gratio, D



if __name__ == '__main__':
    import matplotlib.pyplot as plt
    gam = np.logspace(-4, 0, 100) /100. # into a decimal
    _, G_IZ, D_IZ = IZ_1993(gam=gam, PI=0, sigm=25)
    _, G_DS, D_DS = DS_2001(gam=gam, PI=0, sigm=25, OCR=1)

    fig, (axG, axD) = plt.subplots(nrows=2, ncols=1)
    axG.plot(gam * 100, G_IZ, 'b')
    axG.plot(gam * 100, G_DS, 'r')
    axG.set_xscale('log')
    axD.plot(gam * 100, D_IZ*100, 'b')
    axD.plot(gam * 100, D_DS*100, 'r')
    axD.set_xscale('log')
    plt.show()
```

J.4 Batch Site Response Analyses

The file `Run_Master.py` runs site response analyses for either earthquake dataset and all combinations of motions and profiles.

```
,
import os
#import copy
from time import gmtime, strftime
#import numpypy as np
import numpy as np
import tables as pt
try:
    import ipdb as pdb
except:
    import pdb
#import pandas as pd
import sys
sys.path.append('/media/Storage/Documents/Python/Modules')
import EquivLin as el

from scipy.interpolate import interp1d

m = np.linspace(0.2, 0.4, 11)

fullprofilenames = {
        'Profile000':'Balboa Boulevard-Northridge Earthquake (1994)',
        'Profile001':'Malden Street-Northridge Earthquake (1994)',
        'Profile002':'Wynne Avenue-Northridge Earthquake (1994)',
        'Profile003':'MBARI NO:3 EB-1-Loma Prieta Earthquake (1989)',
        'Profile004':'MBARI NO:3 EB-5-Loma Prieta Earthquake (1989)',
        'Profile005':'MBARI NO:3 UC-B-10-Loma Prieta Earthquake (1989)',
        'Profile006':'Treasure Island-Loma Prieta Earthquake (1989)',
        'Profile007':'SFOBB-1-Loma Prieta Earthquake (1989)',
        'Profile008':'SFOBB-2-Loma Prieta Earthquake (1989)',
        'Profile009':'POO7-2-Loma Prieta Earthquake (1989)',
        'Profile010':'POO7-3-Loma Prieta Earthquake (1989)',
        'Profile011':'Woodward Marine UC-B4-Loma Prieta Earthquake (1989)',
        'Profile012':'Moss Landing UC-B1-Loma Prieta Earthquake (1989)',
        'Profile013':'Moss Landing UC-B2-Loma Prieta Earthquake (1989)',
        'Profile014':'Marine Lab. B1-Loma Prieta Earthquake (1989)',
        'Profile015':'Marine Lab. B2-Loma Prieta Earthquake (1989)',
        'Profile016':'Miller Farm CMF 3-Loma Prieta Earthquake (1989)',
        'Profile017':'Miller Farm CMF 5-Loma Prieta Earthquake (1989)',
        'Profile018':'Miller Farm CMF 8-Loma Prieta Earthquake (1989)',
        'Profile019':'Miller Farm CMF 10-Loma Prieta Earthquake (1989)',
        'Profile020':'Richmond City Hall-Loma Prieta Earthquake (1989)',
        'Profile021':'Port of Richmond POR-2-Loma Prieta Earthquake (1989)',
        'Profile022':'Port of Richmond POR-3-Loma Prieta Earthquake (1989)',
        'Profile023':'Port of Richmond POR-4-Loma Prieta Earthquake (1989)',
        'Profile024':'Miller Farm-Loma Prieta Earthquake (1989)',
        'Profile025':'Farris Farm-Loma Prieta Earthquake (1989)',
        'Profile026':'Wildlife Site-Elmore Ranch (1987)',
        'Profile027':'Wildlife Site-Superstition Hills (1987)',
        'Profile028':'Radio Tower B1-Superstition Hills (1987)',
```

```
        'Profile029':'Radio Tower B2-Superstition Hills (1987)',
        'Profile030':'McKim Ranch A-Superstition Hills (1987)',
        'Profile031':'Kornbloom-Superstition Hills (1987)',
        'Profile032':'River Park A & C-Superstition Hills (1987)',
        'Profile033':'Heber Road A1-Superstition Hills (1987)',
        'Profile034':'Heber Road A2-Superstition Hills (1987)',
        'Profile035':'Heber Road A3-Superstition Hills (1987)',
        'Profile036':'Kornbloom-Westmorland (1981)',
        'Profile037':'McKim Ranch-Westmorland (1981)',
        'Profile038':'Radio Tower B1-Westmorland (1981)',
        'Profile039':'Radio Tower B2-Westmorland (1981)',
        'Profile040':'River Park A & C-Westmorland (1981)',
        'Profile041':'Wildlife-Westmorland (1981)',
        'Profile042':'Heber Road A1-Imperial Valley (1979)',
        'Profile043':'Heber Road A2-Imperial Valley (1979)',
        'Profile044':'Heber Road A3-Imperial Valley (1979)',
        'Profile045':'Kornbloom-Imperial Valley (1979)',
        'Profile046':'Radio Tower B1-Imperial Valley (1979)',
        'Profile047':'Radio Tower B2-Imperial Valley (1979)',
        'Profile048':'River Park A & C-Imperial Valley (1979)',
        'Profile049':'Wildlife Site-Imperial Valley (1979)'}

class WUSComb(pt.IsDescription):
    #Motion Name
    RecSeq = pt.Int16Col(pos=1)
    Motion0 = pt.StringCol(itemsize=32,pos=2)
    Motion1 = pt.StringCol(itemsize=32,pos=3)

    #Profile Name
    Profile = pt.StringCol(itemsize=32,pos=4)
    profiletype = pt.Int32Col(pos=5)

    # Calc'd Info
    DissEn0 = pt.Float32Col(pos=6)
    DissEn1 = pt.Float32Col(pos=7)
    N_EQ0 = pt.Float32Col(pos=8)
    N_EQ1 = pt.Float32Col(pos=9)
    TauAvg0 = pt.Float32Col(pos=10)
    TauAvg1 = pt.Float32Col(pos=11)

    DE_comb = pt.Float32Col(pos=12)
    Neq_comb = pt.Float32Col(pos=13)
    RepErr0 = pt.StringCol(itemsize=1, pos=16)
    iters_0 = pt.Int32Col(pos=17)
    maxerror_0 = pt.Float32Col(pos=18)
    RepErr1 = pt.StringCol(itemsize=1, pos=19)
    iters_1 = pt.Int32Col(pos=20)
    maxerror_1 = pt.Float32Col(pos=21)
    #Degrad = pt.Float32Col(itemsize=2, pos=21)

    # Profile Info
    Depth = pt.Float32Col(pos=22)
    Sigv = pt.Float32Col(pos=23)
    Sigveff = pt.Float32Col(pos=24)
    Sigmeff = pt.Float32Col(pos=25)
    Mod = pt.Float32Col(pos=26)
    Damp = pt.Float32Col(pos=27)
```

```
    Mod1 = pt.Float32Col(pos=26)
    Damp1 = pt.Float32Col(pos=27)

    Vs = pt.Float32Col(pos=28)
    UnitWt = pt.Float32Col(pos=29)
    LayerName = pt.StringCol(itemsize=32, pos=30)
    t = pt.Float32Col(pos=31)
    PI = pt.Float32Col(pos=32)
    z_bed = pt.Float32Col(pos=33)
    Vsbed = pt.Float32Col(pos=33)
    #Vcontrast1 = pt.Float32Col(pos=34)
    #Vcontrast2 = pt.Float32Col(pos=35)
    Vs30 = pt.Float32Col(pos=34)
    Vs12 = pt.Float32Col(pos=35)
    #sa_vs30 = pt.Float32Col(pos=37)
    NEHRP = pt.StringCol(itemsize=1,pos=38)
    Vs30_pref = pt.Float32Col(pos=39)

    # EQ Info
    EpicD = pt.Float32Col(pos=40)
    Eq_Mag = pt.Float32Col(pos=41)
    Mag_Type = pt.StringCol(itemsize=2,pos=42)
    Hypd = pt.Float32Col(pos=43)
    J_BD = pt.Float32Col(pos=44)
    RMSD = pt.Float32Col(pos=45)
    CLSTD = pt.Float32Col(pos=46)
    #D_ = pt.Float32Col(pos=47)

    EQEvent = pt.Int32Col(pos=48)
    PGA = pt.Float32Col(pos=49)
    PGV = pt.Float32Col(pos=50)
    PGD = pt.Float32Col(pos=51)
    ThetaD = pt.Float32Col(pos=52)
    X = pt.Float32Col(pos=53)
    PhiD = pt.Float32Col(pos=54)
    Y = pt.Float32Col(pos=55)
    Rake = pt.Float32Col(pos=56)
    EQ_Mech = pt.Float32Col(pos=57)
    Dip = pt.Float32Col(pos=58)

    # More motion info
    T_d01_0 = pt.Float32Col(pos=59)
    T_d01_1 = pt.Float32Col(pos=60)
    T_d05_0 = pt.Float32Col(pos=61)
    T_d05_1 = pt.Float32Col(pos=62)
    #a_rms01_0 = pt.Float32Col(pos=63)
    #a_rms01_1 = pt.Float32Col(pos=64)
    T_tb75_0 = pt.Float32Col(pos=63)
    T_tb75_1 = pt.Float32Col(pos=64)
    T_tb95_0 = pt.Float32Col(pos=65)
    T_tb95_1 = pt.Float32Col(pos=66)
    I_a_0 = pt.Float32Col(pos=67)
    I_a_1 = pt.Float32Col(pos=68)

    amax_surf_0 = pt.Float32Col(pos=69)
    amax_surf_1 = pt.Float32Col(pos=70)
    amax_surf_max = pt.Float32Col(pos=71)
```

```
164
165         # Other calc'd info
166     rd_0 = pt.Float32Col(pos=72)
167     rd_1 = pt.Float32Col(pos=73)
168     Ar_0 = pt.Float32Col(pos=74)
169     Ar_1 = pt.Float32Col(pos=75)
170     Artype = pt.StringCol(itemsize=8, pos=75)
171     #Ar_ratio_0 = pt.Float32Col(pos=76)
172     #Ar_ratio_1 = pt.Float32Col(pos=77)
173     #DE_ratio_0 = pt.Float32Col(pos=78)
174     #DE_ratio_1 = pt.Float32Col(pos=79)
175     #Neq_ratio_0 = pt.Float32Col(pos=80)
176     #Neq_ratio_1 = pt.Float32Col(pos=81)
177     #PeakTimeRatio0 = pt.Float32Col(pos=82)
178     #PeakTimeRatio1 = pt.Float32Col(pos=83)
179     MaxError = pt.Float32Col(pos=76)
180     Discard0 = pt.Float32Col(pos=77)
181     Discard1 = pt.Float32Col(pos=77)
182     m = pt.Float32Col(pos=77)
183     cutoff = pt.Float32Col(pos=77)
184     PMNeq0 = pt.Float32Col(pos=77)
185     PMNeq1 = pt.Float32Col(pos=77)
186     PMNeqVS = pt.Float32Col(pos=77)
187     PMNeqVSdis = pt.Float32Col(pos=77)
188
189 class WUSSingle(pt.IsDescription):
190     #Motion Name
191     RecSeq = pt.Int16Col(pos=1)
192     Motion = pt.StringCol(itemsize=32,pos=2)
193
194     #Profile Name
195     Profile = pt.StringCol(itemsize=32,pos=3)
196     profiletype = pt.Int32Col(pos=4)
197
198     # Calc'd Info
199     DissEn = pt.Float32Col(pos=5)
200     N_EQ = pt.Float32Col(pos=6)
201     TauAvg = pt.Float32Col(pos=7)
202
203     RepErr = pt.StringCol(itemsize=1, pos=9)
204     iters = pt.Int32Col(pos=10)
205     maxerror = pt.Float32Col(pos=11)
206
207     # Profile Info
208     Depth = pt.Float32Col(pos=12)
209     Sigv = pt.Float32Col(pos=13)
210     Sigveff = pt.Float32Col(pos=14)
211     Sigmeff = pt.Float32Col(pos=15)
212     Mod = pt.Float32Col(pos=16)
213     Damp = pt.Float32Col(pos=17)
214
215     Vs = pt.Float32Col(pos=18)
216     UnitWt = pt.Float32Col(pos=19)
217     LayerName = pt.StringCol(itemsize=32, pos=20)
218     t = pt.Float32Col(pos=21)
219     PI = pt.Float32Col(pos=22)
220     z_bed = pt.Float32Col(pos=23)
```

```
    Vsbed = pt.Float32Col(pos=24)
    Vs30 = pt.Float32Col(pos=25)
    Vs12 = pt.Float32Col(pos=26)
    NEHRP = pt.StringCol(itemsize=1,pos=27)
    Vs30_pref = pt.Float32Col(pos=28)

    # EQ Info
    EpicD = pt.Float32Col(pos=29)
    Eq_Mag = pt.Float32Col(pos=30)
    Mag_Type = pt.StringCol(itemsize=2,pos=31)
    Hypd = pt.Float32Col(pos=32)
    J_BD = pt.Float32Col(pos=33)
    RMSD = pt.Float32Col(pos=34)
    CLSTD = pt.Float32Col(pos=35)

    EQEvent = pt.Int32Col(pos=36)
    PGA = pt.Float32Col(pos=37)
    PGV = pt.Float32Col(pos=38)
    PGD = pt.Float32Col(pos=39)
    ThetaD = pt.Float32Col(pos=40)
    X = pt.Float32Col(pos=41)
    PhiD = pt.Float32Col(pos=42)
    Y = pt.Float32Col(pos=43)
    Rake = pt.Float32Col(pos=44)
    EQ_Mech = pt.Float32Col(pos=45)
    Dip = pt.Float32Col(pos=46)

    # More motion info
    T_d01 = pt.Float32Col(pos=47)
    T_d05 = pt.Float32Col(pos=48)
    T_tb95 = pt.Float32Col(pos=49)
    T_tb75 = pt.Float32Col(pos=50)
    I_a = pt.Float32Col(pos=51)

    amax_surf = pt.Float32Col(pos=52)

        # Other calc'd info
    rd = pt.Float32Col(pos=53)
    Ar = pt.Float32Col(pos=54)
    Artype = pt.StringCol(itemsize=8, pos=54)
    MaxError = pt.Float32Col(pos=55)
    Discard = pt.Float32Col(pos=56)
    m = pt.Float32Col(pos=56)
    cutoff = pt.Float32Col(pos=56)
    PMNeq = pt.Float32Col(pos=56)

class CEUSComb(pt.IsDescription):
    #Motion Name
    RecSeq = pt.Int32Col(pos=1)
    Motion0 = pt.StringCol(itemsize=32,pos=2)
    Motion1 = pt.StringCol(itemsize=32,pos=3)

    #Profile Name
    Profile = pt.StringCol(itemsize=32,pos=4)
    profiletype = pt.Int32Col(pos=5)

    # Calc'd Info
```

```
278     DissEn0 = pt.Float32Col(pos=6)
279     N_EQ0 = pt.Float32Col(pos=7)
280     TauAvg0 = pt.Float32Col(pos=8)
281
282     DissEn1 = pt.Float32Col(pos=9)
283     N_EQ1 = pt.Float32Col(pos=10)
284     TauAvg1 = pt.Float32Col(pos=11)
285
286     RepErr0 = pt.StringCol(itemsize=1, pos=12)
287     iters_0 = pt.Int32Col(pos=13)
288     maxerror_0 = pt.Float32Col(pos=14)
289     RepErr1 = pt.StringCol(itemsize=1, pos=15)
290     iters_1 = pt.Int32Col(pos=16)
291     maxerror_1 = pt.Float32Col(pos=17)
292
293     DE_comb = pt.Float32Col(pos=18)
294     Neq_comb = pt.Float32Col(pos=19)
295
296     # Profile Info
297     Depth = pt.Float32Col(pos=22)
298     Sigv = pt.Float32Col(pos=23)
299     Sigveff = pt.Float32Col(pos=24)
300     Sigmeff = pt.Float32Col(pos=25)
301     Mod = pt.Float32Col(pos=26)
302     Damp = pt.Float32Col(pos=27)
303     Mod1 = pt.Float32Col(pos=26)
304     Damp1 = pt.Float32Col(pos=27)
305
306     Vs = pt.Float32Col(pos=28)
307     UnitWt = pt.Float32Col(pos=29)
308     LayerName = pt.StringCol(itemsize=32, pos=30)
309     t = pt.Float32Col(pos=31)
310     PI = pt.Float32Col(pos=32)
311     z_bed = pt.Float32Col(pos=33)
312     Vsbed = pt.Float32Col(pos=34)
313     Vs30 = pt.Float32Col(pos=35)
314     Vs12 = pt.Float32Col(pos=36)
315     sa_vs30 = pt.Float32Col(pos=37)
316     Vs30_pref = pt.Float32Col(pos=38)
317
318     # EQ Info
319     Eq_Mag = pt.Float32Col(pos=39)
320     CLSTD = pt.Float32Col(pos=40)
321     EQEvent = pt.StringCol(itemsize=64, pos=41)
322     PGA = pt.Float32Col(pos=42)
323     PGV = pt.Float32Col(pos=43)
324     PGD = pt.Float32Col(pos=44)
325     PGA_2 = pt.Float32Col(pos=45)
326     PGV_2 = pt.Float32Col(pos=46)
327     PGD_2 = pt.Float32Col(pos=47)
328
329     T_d01_0 = pt.Float32Col(pos=59)
330     T_d01_1 = pt.Float32Col(pos=60)
331     T_d05_0 = pt.Float32Col(pos=61)
332     T_d05_1 = pt.Float32Col(pos=62)
333     #a_rms01_0 = pt.Float32Col(pos=63)
334     #a_rms01_1 = pt.Float32Col(pos=64)
```

```
    T_tb95_0 = pt.Float32Col(pos=65)
    T_tb95_1 = pt.Float32Col(pos=66)
    T_tb75_0 = pt.Float32Col(pos=63)
    T_tb75_1 = pt.Float32Col(pos=64)
    I_a_0 = pt.Float32Col(pos=67)
    I_a_1 = pt.Float32Col(pos=68)

    amax_surf_0 = pt.Float32Col(pos=60)
    amax_surf_1 = pt.Float32Col(pos=61)
    amax_surf_max = pt.Float32Col(pos=62)

        # Other calc'd info
    rd_0 = pt.Float32Col(pos=63)
    rd_1 = pt.Float32Col(pos=64)
    Ar_0 = pt.Float32Col(pos=65)
    Ar_1 = pt.Float32Col(pos=66)
    Artype = pt.StringCol(itemsize=8, pos=66)
    Bin = pt.StringCol(itemsize=16, pos=67)
    MaxError = pt.Float32Col(pos=68)
    Discard0 = pt.Float32Col(pos=77)
    Discard1 = pt.Float32Col(pos=77)
    m = pt.Float32Col(pos=77)
    cutoff = pt.Float32Col(pos=77)
    PMNeq0 = pt.Float32Col(pos=77)
    PMNeq1 = pt.Float32Col(pos=77)
    PMNeqVS = pt.Float32Col(pos=77)
    PMNeqVSdis = pt.Float32Col(pos=77)

class CEUSSingle(pt.IsDescription):
    #Motion Name
    RecSeq = pt.Int32Col(pos=1)
    Motion = pt.StringCol(itemsize=32,pos=2)

    #Profile Name
    Profile = pt.StringCol(itemsize=32,pos=3)
    profiletype = pt.Int32Col(pos=4)

    # Calc'd Info
    DissEn = pt.Float32Col(pos=5)
    N_EQ = pt.Float32Col(pos=6)
    TauAvg = pt.Float32Col(pos=7)

    RepErr = pt.StringCol(itemsize=1, pos=8)
    iters = pt.Int32Col(pos=9)
    maxerror = pt.Float32Col(pos=10)


    # Profile Info
    Depth = pt.Float32Col(pos=12)
    Sigv = pt.Float32Col(pos=13)
    Sigveff = pt.Float32Col(pos=14)
    Sigmeff = pt.Float32Col(pos=15)
    Mod = pt.Float32Col(pos=16)
    Damp = pt.Float32Col(pos=17)

    Vs = pt.Float32Col(pos=18)
    UnitWt = pt.Float32Col(pos=19)
```

```
    LayerName = pt.StringCol(itemsize=32, pos=20)
    t = pt.Float32Col(pos=21)
    PI = pt.Float32Col(pos=22)
    z_bed = pt.Float32Col(pos=23)
    Vsbed = pt.Float32Col(pos=23)
    Vs30 = pt.Float32Col(pos=24)
    Vs12 = pt.Float32Col(pos=25)
    sa_vs30 = pt.Float32Col(pos=26)
    Vs30_pref = pt.Float32Col(pos=27)

    # EQ Info
    Eq_Mag = pt.Float32Col(pos=28)
    CLSTD = pt.Float32Col(pos=29)
    EQEvent = pt.StringCol(itemsize=64, pos=30)
    PGA = pt.Float32Col(pos=31)
    PGV = pt.Float32Col(pos=32)
    PGD = pt.Float32Col(pos=33)

    T_d01 = pt.Float32Col(pos=34)
    T_d05 = pt.Float32Col(pos=35)
    T_tb95 = pt.Float32Col(pos=36)
    T_tb75 = pt.Float32Col(pos=37)
    I_a = pt.Float32Col(pos=38)

    amax_surf = pt.Float32Col(pos=39)

        # Other calc'd info
    rd = pt.Float32Col(pos=40)
    Ar = pt.Float32Col(pos=41)
    Artype = pt.StringCol(itemsize=8, pos=41)
    Bin = pt.StringCol(itemsize=16, pos=42)
    MaxError = pt.Float32Col(pos=43)
    Discard = pt.Float32Col(pos=44)
    m = pt.Float32Col(pos=44)
    cutoff = pt.Float32Col(pos=44)
    PMNeq = pt.Float32Col(pos=44)

def TokimatsuSeedDirect(gam,Gratio,damping,Gmax, tau_avg, **kwargs):
    '''
    This function will estimate degraded damping and degraded modulus
    by iteration and using shear modulus and damping degradation curves:

        Inputs:
            gam    An array of shear strains (decimal) of length n
            Gratio An array (matrix) of shear modulus degradation curves of
                   shape m X n where m is the number of layers
            damping An array of damping degradation curves, also m X n
            Gmax   The small strain shear modulus
            tau_avg The average shear stress
        Outputs: Three arrays: G, D, and gamma each of length m.

    '''
    returnError = kwargs.get('returnError', False)
    G_ = np.zeros(Gratio.shape[0],dtype=float)
    D_ = np.zeros(Gratio.shape[0],dtype=float)
    gam_ = np.zeros(Gratio.shape[0],dtype=float)
    ErrorList = []
```

```
    if np.isscalar(Gmax):
        Gmax = [Gmax]
    if np.isscalar(tau_avg):
        tau_avg = [tau_avg]
    for i in xrange(len(Gmax)):
        #if i == 208: pdb.set_trace()
        Damp = np.ravel(damping[i,:])
        Grat = np.ravel(Gratio[i,:])
        gam_Grat = gam * Grat
        gamGratiocurv = interp1d(np.log10(gam_Grat), np.log10(gam),
                                 kind='linear')

        Gratiocurv = interp1d(np.log10(gam), Grat,
                                 kind='linear')
        Dcurv = interp1d(np.log10(gam), Damp,
                             kind='linear')
        try:
            newgam = 10 ** (gamGratiocurv(np.log10(tau_avg[i] / Gmax[i])))
        except ValueError:
            #if newgam > gam_[-1]:
            G_[i] = Gmax[i] * Grat[-1]
            D_[i] = Damp[-1]
            gam_[i] = gam[-1]
            print('Gamma exceeds range!, i={}'.format(i))
            ErrorList.append(i)
        else:
            switch=False
            G_[i] = Gmax[i] * Gratiocurv(np.log10(newgam))
            D_[i] = Dcurv(np.log10(newgam))
            gam_[i] = newgam
    if returnError:
        if len(G_) == 1:
            return G_[0], D_[0], gam_[0], ErrorList
        else:
            return G_, D_, gam_, ErrorList
    else:
        if len(G_) == 1:
            return G_[0], D_[0], gam_[0]
        else:
            return G_, D_, gam_

def DobryIter(gam,Gratio,damping,Gmax, tau_avg,error=0.01):
    '''
    This function will estimate degraded damping and degraded modulus
    by iteration and using shear modulus and damping degradation curves:

        Inputs:
            gam    An array of shear strains (decimal) of length n
            Gratio An array (matrix) of shear modulus degradation curves of
                   shape m X n where m is the number of layers
            damping An array of damping degradation curves, also m X n
        Outputs: Two arrays: G and D, each of length m.

    '''
    try:
        len(tau_avg)
    except TypeError:
```

```
        tau_avg = [tau_avg]
        Gmax = [Gmax]
    G_ = np.zeros([len(tau_avg)],dtype=float)
    D_ = np.zeros([len(tau_avg)],dtype=float)
    gam_ = np.zeros([len(tau_avg)],dtype=float)
    for j in xrange(len(tau_avg)):
        if len(tau_avg) == 1:
            Damp = np.ravel(damping)
            Grat = np.ravel(Gratio)
        else:
            Damp = np.ravel(damping[j,:])
            Grat = np.ravel(Gratio[j,:])
        gamma = gam[0]
        gamma0 = gamma + 0.0000001

        Gratiocurv = interp1d(np.log10(gam), Grat,
                              kind='linear')
        Dcurv = interp1d(np.log10(gam), Damp,
                         kind='linear')
        thiserror = 1
        count = 0
        try:
            while thiserror > error:
                count += 1
                if count > 50:
                    print('Layer %s did not converge!'%str(j))
                    print('Error = {:.5f}'.format(thiserror))
                    break
                G = Gmax[j] * Gratiocurv(np.log10(gamma))
                gamma = tau_avg[j] / G
                thiserror = abs(gamma - gamma0) / gamma0
                gamma0 = gamma
            D = Dcurv(np.log10(gamma))
            gam__ = gamma
        except ValueError:
            if gamma > gam[-1]:
                G = Gmax[j] * Grat[-1]
                D = Damp[-1]
                gam__ = gam[-1]
            else:
                G = Gmax[j] * Grat[0]
                D = Damp[0]
                gam__ = gam[0]
            print('Interp error!')

        G_[j] = G
        D_[j] = D
        gam_[j] = gam__
    if len(tau_avg) == 1:
        return G_[0], D_[0], gam_[0]
    else:
        return G_, D_, gam_

class Profiles(object):
    def __init__(self, **kwargs):
        self.CEUS = kwargs.get('CEUS', False)
        if os.name == 'posix':
```

```
563             self.myfile = pt.open_file(
564                    u'/media/Storage/Documents/00-VtResearch/Energy-based/' +
565                        #u'Soil Profiles/SoilProfilesSimp7.h5')
566                        u'Soil Profiles/SoilProfiles2.h5')
567         else:
568             self.myfile = pt.open_file(
569                    u'/../../Documents/00-VtResearch/Energy-based/' +
570                        #u'Soil Profiles/SoilProfilesSimp7.h5')
571                        u'Soil Profiles/SoilProfiles2.h5')
572         if self.CEUS:
573             self.mywhere = '/CetinProfiles_CEUS/'
574         else:
575             self.mywhere = '/CetinProfiles/'
576 
577         self.groups = dict(
578             # Soil profile group catergories
579             groupA = ['Profile000','Profile001','Profile002'],
580             groupB = ['Profile003','Profile004','Profile005',
581                       'Profile011', 'Profile012', 'Profile013',
582                       'Profile014', 'Profile015', 'Profile016',
583                       'Profile017', 'Profile018', 'Profile019',
584                       'Profile024', 'Profile025'],
585             groupC = ['Profile006'],
586             groupD = ['Profile007','Profile008','Profile009', 'Profile010'],
587             groupE = ['Profile020', 'Profile021', 'Profile022', 'Profile023'],
588             groupF = ['Profile026', 'Profile027'],
589             groupG = ['Profile028', 'Profile029', 'Profile030',
590                       'Profile031', 'Profile032', 'Profile033',
591                       'Profile034', 'Profile035', 'Profile036',
592                       'Profile037', 'Profile038', 'Profile039',
593                       'Profile040', 'Profile041', 'Profile045',
594                       'Profile046', 'Profile047', 'Profile048',
595                       'Profile049'],
596             groupH = ['Profile042', 'Profile043', 'Profile044'])
597 
598         self.i = -1
599         self.profilelist = self.myfile.list_nodes(self.mywhere)
600         self.grouplist = [self.getGroup(prof.title) for prof
601                in self.profilelist]
602 
603 
604     def close(self):
605         self.myfile.close()
606 
607     def __iter__(self):
608         return self
609 
610     def next(self):
611         self.i += 1
612         if self.i < len(self.profilelist):
613             self.profile = self.profilelist[self.i]
614             self.group = self.grouplist[self.i]
615             sys.stdout.write('  {} --- {}  ---'.format(self.profile.title,
616                 strftime("%H:%M:%S", gmtime())))
617             sys.stdout.flush()
618             return self.profile
619         else:
```

```
620             self.i = -1
621             raise StopIteration()
622 
623     def status(self, prof):
624             sys.stdout.write('  {} --- {}  ---'.format(prof.title,
625                 strftime("%H:%M:%S", gmtime())))
626             sys.stdout.flush()
627 
628 
629     def getGroup(self, name):
630         i = -1
631         for values in self.groups.itervalues():
632             i += 1
633             if name in values:
634                 return i
635 
636 
637 class Motions(object):
638     def __init__(self, **kwargs):
639         self.CEUS = kwargs.get('CEUS', False)
640         #self.combined = kwargs.get('combinecomps', False)
641         if os.name == 'posix':
642             if kwargs.get('PM', False):
643                 deffile = ('/media/Storage/Documents/00-VtResearch/EQ_Motions/' +
644                            'NGA_Motions/WEST/RockAll-paired.h5')
645                 SummaryFile = ('/media/Storage/Documents/00-VtResearch/' +
646                        'EQ_Motions/NGA_Motions/WEST/NGA_Motions_Summary.h5')
647             else:
648                 if self.CEUS:
649                     deffile = ('/media/Storage/Documents/00-VtResearch/EQ_Motions/' +
650                                'McGuire/CEUS/Rock-paired.h5')
651                     SummaryFile = ('/media/Storage/Documents/00-VtResearch/' +
652                                'EQ_Motions/McGuire/CEUS/Rock_Motions_Summary.h5')
653                 else:
654                     deffile = ('/media/Storage/Documents/00-VtResearch/EQ_Motions/' +
655                                'NGA_Motions/WEST/Rock650-paired.h5')
656                     SummaryFile = ('/media/Storage/Documents/00-VtResearch/' +
657                            'EQ_Motions/NGA_Motions/WEST/NGA_Motions_Summary.h5')
658         else:
659             if kwargs.get('PM', False):
660                 deffile = ('/../../Documents/00-VtResearch/EQ_Motions/' +
661                            'NGA_Motions/WEST/RockAll-paired.h5')
662                 SummaryFile = ('/../../Documents/00-VtResearch/' +
663                        'EQ_Motions/NGA_Motions/WEST/NGA_Motions_Summary.h5')
664             else:
665                 if self.CEUS:
666                     deffile = ('/../../Documents/00-VtResearch/EQ_Motions/' +
667                                'McGuire/CEUS/Rock-paired.h5')
668                     SummaryFile = ('/../../Documents/00-VtResearch/' +
669                                'EQ_Motions/McGuire/CEUS/Rock_Motions_Summary.h5')
670                 else:
671                     deffile = ('/../../Documents/00-VtResearch/EQ_Motions/' +
672                                'NGA_Motions/WEST/Rock650-paired.h5')
673                     SummaryFile = ('/../../Documents/00-VtResearch/' +
674                            'EQ_Motions/NGA_Motions/WEST/NGA_Motions_Summary.h5')
675 
676         self.motfile = pt.open_file(kwargs.get('file', deffile), 'r')
```

```
677         self.sumfile = pt.open_file(kwargs.get('sumfile', SummaryFile), 'r')
678         self.sumtable = self.sumfile.root.SummaryTable
679         self.count = 0
680 
681     def close(self):
682         self.motfile.close()
683         self.sumfile.close()
684 
685     def __call__(self):
686         for mot in self.motfile.list_nodes('/Motions/'):
687             self.count += 1
688             self.motions = mot
689             self.RecNum = int(mot.col('RecSeqNum')[0])
690             self.motstats = self.getMotStats(self.RecNum)
691             print('\n\n\n\nM{} ------ {} ---- {}'.format(self.count, mot.title,
692                 strftime("%Y-%m-%d %H:%M:%S", gmtime())))
693             yield mot
694 
695     def getMotStats(self, RecNum):
696         rownum = self.sumtable.get_where_list('RecSeqNum == RecNum')[0]
697         return self.sumtable[rownum]
698 
699 class Dynamics(object):
700     def __init__(self, mot, Combined, **kwargs):
701         self.MakeProfs = kwargs.get('ProfMaker', False)
702         self.mot = mot
703         self.Combined = Combined
704         self.comb_alt = kwargs.get('comb_alt', False)
705         self.modtype = kwargs.get('modtype', 'FreqInd')
706         self.degradcurves = kwargs.get('DC', 'IZ')
707         self.iters = kwargs.get('iters', 20)
708         self.error = kwargs.get('error', 0.02)
709         self.verbose = kwargs.get('verbose', False)
710         self.m = kwargs.get('m', 0.34)
711         self.cutoff = kwargs.get('cutoff', 0.3)
712         if self.Combined:
713             self.VectorSum = kwargs.get('VectorSum', True)
714         else:
715             self.VectorSum = False
716         # Liu
717         # Lab Curve: m = 0.354, cutoff = 0.3
718         # Field Curve: m = 0.5076, cutoff = 0.1
719         # Seed et al. 1975: cutoff = 0.3
720 
721         self.Mot = []
722         for row in mot.iterrows():
723             motion = row['MotionData']
724             NPTS = int(row['NPTS'])
725             dt = float(row['TimeStep'])
726             if max(motion.shape) != NPTS :
727                 print('NPTS discrepancy!', max(motion.shape), NPTS)
728                 NPTS = max(motion.shape)
729             NFFT = 2 ** el.nextpow2(NPTS)
730             f = np.linspace(0, 1/dt/2, NFFT//2+1)
731             w = f[:NFFT/2+1] * 2 * np.pi
732 
733             fa = np.fft.rfft(motion * 9.81, NFFT)
```

```
            self.Mot.append(
                el.GroundMotion(
                    name=str(self.mot.title),
                    g = 9.81,
                    motion=motion,
                    NFFT = NFFT,
                    FA = fa,
                    f = f,
                    w = w,
                    NPTS = NPTS,
                    dt = dt
                    )
                )

    def calcResults(self):
        '''
        Performs the equivalent linear site response analysis and other calcs
        '''
        self.Results = []
        for MotObj in self.Mot:
            self.Results.append(
                el.Dynamics(
                    self.profile,
                    MotObj,
                    modtype=self.modtype,
                    verbose=self.verbose,
                    DegradCurves=self.degradcurves,
                    iters = self.iters,
                    Error = self.error,
                    start_iters=True,
                    run_all=True
                    )
                )
        if self.Combined and not self.comb_alt:
            self.calcCombined()
        self.amax_surf = []
        self.Neq = []
        self.tau_avg = []
        self.ratio = []
        self.DE = []
        self.Vs30,H = self.profile.calcVs30(returns=True)
        self.Vs12 = self.profile.calcAvgVs(12.)
        self.Sa = []
        for result in self.Results:
            self.amax_surf.append(result.calcAcc(0.0, MaxOnly=False,
                                  domain='time', MotType='within'))
            result.calc_r_d()
            N_eq, tau_avg, ratio, DE = result.calcNeq(set_ratio=0.65,
                                                      returns=True)
            self.Neq.append(N_eq)
            self.tau_avg.append(tau_avg)
            self.ratio.append(ratio)
            self.DE.append(np.maximum(DE, 0))
            # In some weird cases, DE can be slightly negative.
            # I only saw this occur for one motion, at depth (>100m).
            # I think it occurred because the motion (CAD340) had a strong
            # sinusoidal pattern.
```

```
            self.Sa.append(result.RespSpec(0.,
                   w=[2 * np.pi * self.Vs30 / (4 * H)]))
        if self.Combined:
            #if self.comb_alt:
            tau_avg = np.sqrt(self.tau_avg[0] * self.tau_avg[1]) # Geometric mean
            #G, D, gam = TokimatsuSeedDirect( # I have to get the D and G corresponding to
                 this tau_avg
            G, D, gam = DobryIter( # I have to get the D and G corresponding to this tau_avg
                   self.Results[0].gam,
                   self.Results[0].Gratio,
                   self.Results[0].damping,
                   self.Results[0].Gmax[:-1],
                   tau_avg,
                   #returnError=False
                   )

            self.DE1 = 2 * np.pi * D * tau_avg ** 2 / G
            #else:
                #self.DE1 = (2 * np.pi * self.Results[0].D[:-1] *
                           #(self.tau_avg[0] * self.tau_avg[1]) /
                           #self.Results[0].G[:-1])# Geometric mean
            self.Neq_comb = (self.DE[0] + self.DE[1]) / self.DE1
        self.calcDurations()
        self.calcPMNeq()

    def calcCombined(self):
        '''
        A few calcs for when I want the results from both components of motion.
        '''
        # G original is the non-softened shear modulus
        Gorig = self.profile.Vs ** 2 * self.profile.unit_wt / 9.81
        # Find out which profile is softer and keep it.
        #if (np.average(self.Results[0].G[:-1], weights=self.profile.t) >
             #np.average(self.Results[1].G[:-1], weights=self.profile.t)):
            #G = self.Results[1].G
            #D = self.Results[1].D
            #Gratio = G / Gorig
        #else:
            #G = self.Results[0].G
            #D = self.Results[0].D
            #Gratio = G / Gorig
        G = np.min((self.Results[0].G,self.Results[1].G), axis=0)
        D = np.max((self.Results[0].D,self.Results[1].D), axis=0)
        #print(np.all(self.Results[0].G>=self.Results[1].G))
        #if np.all(self.Results[0].G>=self.Results[1].G) == False:
            #print(self.Results[0].G>=self.Results[1].G)
            #pdb.set_trace()
        #print(np.all(self.Results[1].G>=self.Results[0].G))
        Gratio = G / Gorig
        Vs2 = (G * 9.81 / self.profile.unit_wt) ** 0.5
        # Run both motions again using the softer profile, for one iteration
        for i in range(2):
            self.Results[i] = el.Dynamics(
                                    self.profile,
                                    self.Mot[i],
                                    modtype=self.modtype,
                                    verbose=self.verbose,
```

```
                                    DegradCurves=self.degradcurves,
                                    iters = 0,
                                    InitDamping=D[:-1],
                                    BaseDamping=D[-1],
                                    InitGratio=Gratio[:-1],
                                    BaseGratio=Gratio[-1],
                                    Error = 100.,
                                    start_iters=True,
                                    run_all=True
                                    )

    def calcDurations(self):
        '''
        These are the durations of the calculated surface motions.
        '''
        self.Tb_01 = []
        self.Tb_05 = []
        self.Ttb_75 = []
        self.Ttb_95 = []
        self.I_a = []
        self.Ar = []
        self.Artype= '5to75'
        for i,amax in enumerate(self.amax_surf):
            dt = self.Mot[i].dt
            #Bracketed duration
            time = np.arange(len(amax)) * dt
            ind1 = np.where(np.abs(amax) > 0.01) # Notice this!!! 0.01g's
            time1 = time[ind1]
            try:
                self.Tb_01.append(time1[-1] - time1[0])
                #rms acceleration in m/s/s
                #a_rms01 = np.sqrt(1/T_d01 *
                           #np.trapz((motion[ind1[0][0]:ind1[0][-1]] * 9.81)
                           #** 2,dx=dt))
            except IndexError:
                self.Tb_01.append( np.nan)
                #a_rms01 = np.nan
            ind2 = np.where(np.abs(amax) > 0.05)
            time2 = time[ind2]
            try:
                self.Tb_05.append(time2[-1] - time2[0])
                #rms acceleration in m/s/s
                #a_rms05 = np.sqrt(1/T_d05 *
                           #np.trapz((motion[ind2[0][0]:ind2[0][-1]] * 9.81)
                           #** 2,dx=dt))
            except IndexError:
                self.Tb_05.append(np.nan)
                #a_rms05 = np.nan

            # Another Duration (Significant)
            acc2 = (amax * 9.81) ** 2
            cumacc2 = np.cumsum(acc2)
            ind3 = np.where((cumacc2 > 0.05 * cumacc2[-1]) &
                           (cumacc2 < 0.95 * cumacc2[-1]))
            time3 = time[ind3]
            self.Ttb_95.append(time3[-1] - time3[0])

```

```
            ind4 = np.where((cumacc2 > 0.05 * cumacc2[-1]) &
                            (cumacc2 < 0.75 * cumacc2[-1]))
            time4 = time[ind4]
            self.Ttb_75.append(time4[-1] - time4[0])
            #Arias intensity in m/s
            self.I_a.append(np.pi / (2 * 9.81) * np.trapz(acc2 , dx=dt))
            self.Ar.append(el.calcAriasRatio(amax, dt, duration=self.Artype))


    def calcPMNeq(self):
        #m = 0.35 # I've assigned m all the way up at line 24
        #cutoff = 0.3
        m = self.m
        cutoff = self.cutoff
        self.PMNeq = []
        self.Discards = []
        overallmax = np.max(self.amax_surf) # I may need to make this an array first
        combinedhere = False
        for amax in self.amax_surf:
            #if np.isscalar(m):
                #mm = np.array([m])
            #else:
            mm = m
            mymax = np.max(np.abs(amax))
            if combinedhere:
                amaxn = amax / overallmax
            else:
                amaxn = amax / mymax
            # Now I need to get the peak values for the positive and negatives parts.
            ind = np.where(np.sign(amaxn[:-1]) != np.sign(amaxn[1:]))[0] # Index of each
                 zero crossing
            #Npos = 0.#np.zeros(mm.shape)
            #Nneg = 0.#np.zeros(mm.shape)
            myN = 0.
            discard = 0.
            Ncyc = 0.5 # I'm assuming the result of the mean crossing is one half cycle,
                 change if needed.
            for i, val in enumerate(ind):
                try:
                    if i == 0:
                        peak = np.max(np.abs(amaxn[0:val]))
                        #sign = np.sign(np.mean(amaxn[0:val]))
                    else:
                        peak = np.max(np.abs(amaxn[ind[i-1]:val]))
                        #sign = np.sign(np.mean(amaxn[ind[i-1]:val]))
                except ValueError:
                    continue
                if peak < cutoff:
                    discard += Ncyc * np.exp(np.log(peak / 0.65) / mm)
                    continue
                else:
                    myN += Ncyc * np.exp(np.log(peak / 0.65) / mm)
                #if sign == -1:
                    #Nneg += Ncyc * np.exp(np.log(peak / 0.65) / mm)
                #else:
                    #Npos += Ncyc * np.exp(np.log(peak / 0.65) / mm)
            self.PMNeq.append(myN) # Just summing here since Ncyc is 0.5
```

```
            self.Discards.append(discard)
        if self.VectorSum:
            P2P = False
            VS = np.sqrt(self.amax_surf[0] ** 2 + self.amax_surf[1] ** 2)
            VS = VS / np.max(VS)
            alt = False
            if alt:
                import CycleCounting as cc
                CC = cc.Load(VS)
                Peaks, Cycles = CC.calcPeak2Peak(AmpType='ASTM')
                self.PMNeqVS = self.PMFloor(Peaks, Cycles)
                self.PMNeqVSdis = 0. # Not a good way to do this using the alt method.
            else:
                if P2P:
                    import CycleCounting as cc
                    CC = cc.Load(VS)
                    Peaks, Cycles = CC.calcPeak2Peak(AmpType='ASTM')
                    ind1 = np.where(Peaks >= cutoff)
                    ind2 = np.where(Peaks < cutoff)
                    self.PMNeqVS = np.sum(Cycles[ind1] * np.exp(np.log(
                                        Peaks[ind1] / 0.65)/m))
                    self.PMNeqVSdis = np.sum(Cycles[ind2] * np.exp(np.log(
                                        Peaks[ind2] / 0.65)/m))
                else:
                    # Find the location of the peaks:
                    dVS = np.diff(VS)
                    ind0 = np.where(np.sign(dVS[1:]) != np.sign(dVS[:-1]))[0] + 1
                    Peaks = VS[ind0]
                    #import matplotlib.pyplot as plt
                    #plt.plot(VS)
                    #plt.plot(ind0, Peaks, 'or')
                    #plt.show()
                    #pdb.set_trace()
                    ind1 = np.where(Peaks >= cutoff)
                    ind2 = np.where(Peaks < cutoff)
                    self.PMNeqVS = np.sum(0.5 * np.exp(np.log(
                                        Peaks[ind1] / 0.65)/m))
                    self.PMNeqVSdis = np.sum(0.5 * np.exp(np.log(
                                        Peaks[ind2] / 0.65)/m))
                    #print(self.PMNeqVS, self.PMNeqVSdis)

        else:
            self.PMNeqVS = 0.
            self.PMNeqVSdis = 0.


    def __call__(self, profile):
        layername = list(profile.col('LayerName'))
        unit_wt = profile.col('UnitWt')
        t = profile.col('Thickness')[:-1]
        ko = profile.col('LatPressCo')
        phi = profile.col('FricAngle')[:-1]
        Vs = profile.col('ShearWaveVel')
        PI = profile.col('PlastIndex')[:-1]
        DS_OCR = profile.col('DS_OCR')
        DS_soil = profile.col('DS_soil')
        gwt = 0
```

```
        self.profile = el.SoilProfile(
                   str(profile.title),
                   unit_wt,
                   Vs,
                   t,
                   ko=ko,
                   phi=phi,
                   PI=PI,
                   gwt=gwt,
                   layername=layername,
                   DS_soil=DS_soil,
                   OCR=DS_OCR,
                   strainratio=0.65,
                   loc_mot=0, #bedrock
                   outcrop_mot=0 # 0 for outcrop
                   )
        if self.MakeProfs:
            if not os.path.exists('./CSVProfiles/'):
                os.makedirs('./CSVProfiles/')
            self.profile.to_csv(filename='./CSVProfiles/{}.csv'.format(
                self.profile.profile_name))
            return 0
        self.calcResults()
        return 0

class Output(object):
    def __init__(self, CEUS, combinecomps, **kwargs):
        self.CEUS = CEUS
        self.combined = combinecomps
        DC = kwargs.get('DC')
        self.archive = kwargs.get('archive', False)
        if kwargs.get('PM', False):
            filename = 'IO{}-{}.h5'.format('_WUS_PM',
                   strftime('%d%b%Y', gmtime()))
            OutTable = WUSPM
            self.writer = self.PM
            self.outfile = pt.open_file(filename, mode='w',
                               title='Results of EL Iterations')
            self.outtable = self.outfile.createTable('/', 'EquivLinIters',
                               OutTable, expectedrows=7000)
        else:
            if kwargs.get('comb_alt', False):
                Altstr = 'ALT'
            else:
                Altstr = ''
            outdirbase = './Archive/'
            if CEUS:
                if combinecomps:
                    filename = 'IO{}-{}-{}.h5'.format('_CEUS_Comb' + Altstr,
                           DC, strftime('%d%b%Y', gmtime()))
                    OutTable = CEUSComb
                    self.writer = self.Combed
                    self.outdir = (outdirbase + strftime('%d%b%Y', gmtime()) +
                           '/CEUS' + Altstr + '/')
                else:
                    filename = 'IO{}-{}-{}.h5'.format('_CEUS_Sing', DC,
                           strftime('%d%b%Y', gmtime()))
```

```
                    OutTable = CEUSSingle
                    self.writer = self.Single
            else:
                if combinecomps:
                    filename = 'IO{}-{}-{}.h5'.format('_WUS_Comb' + Altstr,
                           DC, strftime('%d%b%Y', gmtime()))
                    OutTable = WUSComb
                    self.writer = self.Combed
                    self.outdir = (outdirbase + strftime('%d%b%Y', gmtime()) +
                           '/WUS' + Altstr + '/')
                else:
                    filename = 'IO{}-{}-{}.h5'.format('_WUS_Sing', DC,
                           strftime('%d%b%Y', gmtime()))
                    OutTable = WUSSingle
                    self.writer = self.Single
            self.outfile = pt.open_file(filename, mode='w',
                               title='Results of EL Iterations')
            self.outtable = self.outfile.createTable('/', 'EquivLinIters', OutTable,
                                               expectedrows=200000)


    def close(self):
        self.outfile.close()

    def __call__(self, ResObj, ProfileGroup, Motions):
        self.writer(ResObj, ProfileGroup, Motions)

    def Single(self, ResObj, ProfileGroup, Motions):
        for j, result in enumerate(ResObj.Results):
            t = ResObj.profile.t
            ptmotions = Motions.motions
            motstats = Motions.motstats
            soilprofile = ResObj.profile

            for i in xrange(len(t)):
                entry = self.outtable.row
                entry['RecSeq'] = ptmotions[j]['RecSeqNum']
                entry['Profile'] = soilprofile.profile_name
                entry['Depth'] = soilprofile.t_mid[i]
                entry['Sigv'] = soilprofile.sigv[i]
                entry['Sigveff'] = soilprofile.sigveff[i]
                entry['Sigmeff'] = soilprofile.sigmeff[i]
                entry['Mod'] = result.G[i] # Softened condition
                entry['Damp'] = result.D[i] # Softened condition

                entry['DissEn'] = ResObj.DE[j][i]
                entry['N_EQ'] = ResObj.Neq[j][i]
                entry['TauAvg'] = ResObj.tau_avg[j][i]
                entry['RepErr'] = result.ReportError
                entry['iters'] = result.count
                entry['maxerror'] = result.MaxError

                entry['rd'] = result.rd[i]

                entry['Vs'] = soilprofile.Vs[i] # Original shear wave velocity, not softened.
                entry['UnitWt'] = soilprofile.unit_wt[i]
                entry['LayerName'] = str(soilprofile.layername[i])
```

```
                if self.CEUS:
                    if j == 0:
                        entry['PGA'] = motstats['PGA_H1']
                        entry['PGV'] = motstats['PGV_H1']
                        entry['PGD'] = motstats['PGD_H1']
                    else:
                        entry['PGA'] = motstats['PGA_H2']
                        entry['PGV'] = motstats['PGV_H2']
                        entry['PGD'] = motstats['PGD_H2']
                    entry['Bin'] = motstats['Bin']
                    #entry['Motion0'] = str(ptmotions[0]['Station_Name']) #for CEUS:
                         Station_Name
                    entry['EQEvent'] = motstats['EQ_name']
                else:
                    entry['EpicD'] = motstats['EpiD']
                    entry['Mag_Type'] = motstats['Mag_type']
                    entry['Hypd'] = motstats['HypD']
                    entry['J_BD'] = motstats['Joy_Boore']
                    entry['RMSD'] = motstats['RmsD']
                    entry['Vs30_pref'] = motstats['Vs30_pref']
                    entry['NEHRP'] = motstats['NEHRP'] # StringCol(itemsize=1)
                    entry['ThetaD'] = motstats['ThetaD']
                    entry['PhiD'] = motstats['Phi_D']
                    entry['X'] = motstats['X']
                    entry['Y'] = motstats['Y']
                    entry['Rake'] = motstats['Rake']
                    entry['EQ_Mech'] = motstats['EQ_Mech']
                    entry['Dip'] = motstats['Dip']
                    entry['EQEvent'] = motstats['EQID']
                    entry['PGA'] = motstats['PGA']
                    entry['PGV'] = motstats['PGV']
                    entry['PGD'] = motstats['PGD']

                entry['Motion'] = str(ptmotions[j]['Name']) #for CEUS: Station_Name
                entry['Eq_Mag'] = motstats['EQ_Mag']
                entry['CLSTD'] = motstats['ClstD']

                entry['T_d01'] = ResObj.Tb_01[j] #Bracketed duration (> 0.01g)
                entry['T_d05'] = ResObj.Tb_05[j] #Bracketed duration (> 0.05g)
                entry['T_tb95'] = ResObj.Ttb_95[j]   #Trifunac and Brady duration based on
                     5-95% energy
                entry['T_tb75'] = ResObj.Ttb_75[j]   #Trifunac and Brady duration based on
                     5-95% energy
                entry['I_a'] = ResObj.I_a[j]      # Arias Intensity
                entry['Ar'] = ResObj.Ar[j]
                entry['Artype'] = ResObj.Artype

                entry['t'] = t[i] #layer thickness
                entry['z_bed'] = np.sum(t)
                entry['Vsbed'] = soilprofile.Vs[-1]
                entry['amax_surf'] = np.max(np.abs(ResObj.amax_surf[j]))
                entry['PI'] = soilprofile.PI[i]
                entry['profiletype'] = ProfileGroup

                entry['Vs30'] = ResObj.Vs30
                entry['Vs12'] = ResObj.Vs12
                #entry['sa_vs30'] = Sa0
```

```
                #entry['sa_vs31'] = Sa1
                entry['PMNeq'] = ResObj.PMNeq[j]
                entry['Discard'] = ResObj.Discards[j]
                entry['m'] = ResObj.m
                entry['cutoff'] = ResObj.cutoff
                #entry['PMNeq20'] = ResObj.PMNeq[j][0]
                #entry['PMNeq22'] = ResObj.PMNeq[j][1]
                #entry['PMNeq24'] = ResObj.PMNeq[j][2]
                #entry['PMNeq26'] = ResObj.PMNeq[j][3]
                #entry['PMNeq28'] = ResObj.PMNeq[j][4]
                #entry['PMNeq30'] = ResObj.PMNeq[j][5]
                #entry['PMNeq32'] = ResObj.PMNeq[j][6]
                #entry['PMNeq34'] = ResObj.PMNeq[j][7]
                #entry['PMNeq36'] = ResObj.PMNeq[j][8]
                #entry['PMNeq38'] = ResObj.PMNeq[j][9]
                #entry['PMNeq40'] = ResObj.PMNeq[j][10]
                entry.append()
            self.outtable.flush()

    def Combed(self, ResObj, ProfileGroup, Motions):
        t = ResObj.profile.t
        ptmotions = Motions.motions
        motstats = Motions.motstats
        soilprofile = ResObj.profile
        DE_comb = np.sum(ResObj.DE, axis=0)

        if self.archive:
            self.makeCSVs(ResObj, ProfileGroup, Motions)

        for i in xrange(len(t)):
            entry = self.outtable.row
            entry['RecSeq'] = ptmotions[0]['RecSeqNum']
            entry['Profile'] = soilprofile.profile_name
            entry['Depth'] = soilprofile.t_mid[i]
            entry['Sigv'] = soilprofile.sigv[i]
            entry['Sigveff'] = soilprofile.sigveff[i]
            entry['Sigmeff'] = soilprofile.sigmeff[i]
            entry['Mod'] = ResObj.Results[0].G[i] # Softened condition
            entry['Damp'] = ResObj.Results[0].D[i] # Softened condition
            entry['Mod1'] = ResObj.Results[1].G[i] # Softened condition
            entry['Damp1'] = ResObj.Results[1].D[i] # Softened condition

            entry['DissEn0'] = ResObj.DE[0][i]
            entry['N_EQ0'] = ResObj.Neq[0][i]
            entry['TauAvg0'] = ResObj.tau_avg[0][i]
            entry['RepErr0'] = ResObj.Results[0].ReportError
            entry['iters_0'] = ResObj.Results[0].count
            entry['maxerror_0'] = ResObj.Results[0].MaxError

            entry['DissEn1'] = ResObj.DE[1][i]
            entry['N_EQ1'] = ResObj.Neq[1][i]
            entry['TauAvg1'] = ResObj.tau_avg[1][i]
            entry['RepErr1'] = ResObj.Results[1].ReportError
            entry['iters_1'] = ResObj.Results[1].count
            entry['maxerror_1'] = ResObj.Results[1].MaxError

            entry['DE_comb'] = DE_comb[i]
```

```
            entry['Neq_comb'] = ResObj.Neq_comb[i]

            entry['rd_0'] = ResObj.Results[0].rd[i]
            entry['rd_1'] = ResObj.Results[1].rd[i]

            entry['Vs'] = soilprofile.Vs[i] # Original shear wave velocity, not softened.
            entry['UnitWt'] = soilprofile.unit_wt[i]
            entry['LayerName'] = str(soilprofile.layername[i])
            if self.CEUS:
                entry['PGA_2'] = motstats['PGA_H2']
                entry['PGV_2'] = motstats['PGV_H2']
                entry['PGD_2'] = motstats['PGD_H2']
                entry['PGA'] = motstats['PGA_H1']
                entry['PGV'] = motstats['PGV_H1']
                entry['PGD'] = motstats['PGD_H1']
                entry['Bin'] = motstats['Bin']
                #entry['Motion0'] = str(ptmotions[0]['Station_Name']) #for CEUS: Station_Name
                #entry['Motion1'] = str(ptmotions[1]['Station_Name'])
                entry['EQEvent'] = motstats['EQ_name']
            else:
                entry['EpicD'] = motstats['EpiD']
                entry['Mag_Type'] = motstats['Mag_type']
                entry['Hypd'] = motstats['HypD']
                entry['J_BD'] = motstats['Joy_Boore']
                entry['RMSD'] = motstats['RmsD']
                entry['Vs30_pref'] = motstats['Vs30_pref']
                entry['NEHRP'] = motstats['NEHRP'] # StringCol(itemsize=1)
                entry['ThetaD'] = motstats['ThetaD']
                entry['PhiD'] = motstats['Phi_D']
                entry['X'] = motstats['X']
                entry['Y'] = motstats['Y']
                entry['Rake'] = motstats['Rake']
                entry['EQ_Mech'] = motstats['EQ_Mech']
                entry['Dip'] = motstats['Dip']
                entry['EQEvent'] = motstats['EQID']
                entry['PGA'] = motstats['PGA']
                entry['PGV'] = motstats['PGV']
                entry['PGD'] = motstats['PGD']

            entry['Motion0'] = str(ptmotions[0]['Name'])
            entry['Motion1'] = str(ptmotions[1]['Name'])
            entry['Eq_Mag'] = motstats['EQ_Mag']
            entry['CLSTD'] = motstats['ClstD']

            entry['T_d01_0'] = ResObj.Tb_01[0] #Bracketed duration (> 0.01g)
            entry['T_d05_0'] = ResObj.Tb_05[0] #Bracketed duration (> 0.05g)
            entry['T_tb95_0'] = ResObj.Ttb_95[0]   #Trifunac and Brady duration based on
                5-95% energy
            entry['T_tb75_0'] = ResObj.Ttb_75[0]   #Trifunac and Brady duration based on
                5-95% energy
            entry['I_a_0'] = ResObj.I_a[0]      # Arias Intensity
            entry['Ar_0'] = ResObj.Ar[0]
            entry['T_d01_1'] = ResObj.Tb_01[1] #Bracketed duration (> 0.01g)
            entry['T_d05_1'] = ResObj.Tb_05[1] #Bracketed duration (> 0.05g)
            entry['T_tb95_1'] = ResObj.Ttb_95[1]   #Trifunac and Brady duration based on
                5-95% energy
            entry['T_tb75_1'] = ResObj.Ttb_75[1]   #Trifunac and Brady duration based on
```

```
                5-95% energy
            entry['I_a_1'] = ResObj.I_a[1]      # Arias Intensity
            entry['Ar_1'] = ResObj.Ar[1]
            entry['Artype'] = ResObj.Artype

            entry['t'] = t[i] #layer thickness
            entry['z_bed'] = np.sum(t)
            entry['Vsbed'] = soilprofile.Vs[-1]
            entry['amax_surf_0'] = np.max(np.abs(ResObj.amax_surf[0]))
            entry['amax_surf_1'] = np.max(np.abs(ResObj.amax_surf[1]))
            entry['amax_surf_max'] = np.max(np.abs(ResObj.amax_surf))
            entry['PI'] = soilprofile.PI[i]
            entry['profiletype'] = ProfileGroup

            entry['Vs30'] = ResObj.Vs30
            entry['Vs12'] = ResObj.Vs12
            #entry['sa_vs30'] = Sa0
            #entry['sa_vs31'] = Sa1
            entry['PMNeq0'] = ResObj.PMNeq[0]
            entry['PMNeq1'] = ResObj.PMNeq[1]
            entry['PMNeqVS'] = ResObj.PMNeqVS
            entry['PMNeqVSdis'] = ResObj.PMNeqVSdis
            entry['Discard0'] = ResObj.Discards[0]
            entry['Discard1'] = ResObj.Discards[1]
            entry['m'] = ResObj.m
            entry['cutoff'] = ResObj.cutoff
            #entry['PMNeq020'] = ResObj.PMNeq[0][0]
            #entry['PMNeq022'] = ResObj.PMNeq[0][1]
            #entry['PMNeq024'] = ResObj.PMNeq[0][2]
            #entry['PMNeq026'] = ResObj.PMNeq[0][3]
            #entry['PMNeq028'] = ResObj.PMNeq[0][4]
            #entry['PMNeq030'] = ResObj.PMNeq[0][5]
            #entry['PMNeq032'] = ResObj.PMNeq[0][6]
            #entry['PMNeq034'] = ResObj.PMNeq[0][7]
            #entry['PMNeq036'] = ResObj.PMNeq[0][8]
            #entry['PMNeq038'] = ResObj.PMNeq[0][9]
            #entry['PMNeq040'] = ResObj.PMNeq[0][10]
            #entry['PMNeq120'] = ResObj.PMNeq[1][0]
            #entry['PMNeq122'] = ResObj.PMNeq[1][1]
            #entry['PMNeq124'] = ResObj.PMNeq[1][2]
            #entry['PMNeq126'] = ResObj.PMNeq[1][3]
            #entry['PMNeq128'] = ResObj.PMNeq[1][4]
            #entry['PMNeq130'] = ResObj.PMNeq[1][5]
            #entry['PMNeq132'] = ResObj.PMNeq[1][6]
            #entry['PMNeq134'] = ResObj.PMNeq[1][7]
            #entry['PMNeq136'] = ResObj.PMNeq[1][8]
            #entry['PMNeq138'] = ResObj.PMNeq[1][9]
            #entry['PMNeq140'] = ResObj.PMNeq[1][10]
            entry.append()
        self.outtable.flush()

    def makeCSVs(self, ResObj, ProfileGroup, Motions):
        '''
        This creates .csv files for Dr. Green's archiving purposes

        '''
        soilprofile = ResObj.profile
```

```
        ptmotions = Motions.motions
        outdir = self.outdir + '/{}/'.format(soilprofile.profile_name)
        if not os.path.exists(outdir):
            os.makedirs(outdir)

        headerlinesprof = ['Depth (m)',
                           #'Layer Name',
                           'Total Stress (kPa)',
                           'Eff. Stress (kPa)',
                           'Mean Eff. Stress (kPa)',
                           'Diss En 1 (kPa)',
                           'Diss En 2 (kPa)',
                           'G 1 (kPa)',
                           'G 2 (kPa)',
                           'D 1',
                           'D 2',
                           'Tau_avg1 (kPa)',
                           'Tau_avg2 (kPa)',
                           'Neq_comb',
                           'r_d1',
                           'r_d2']
        outarrayprof = np.vstack((soilprofile.t_mid,
                                  #layername[:-1],
                                  soilprofile.sigv,
                                  soilprofile.sigveff,
                                  soilprofile.sigmeff,
                                  ResObj.DE[0],
                                  ResObj.DE[1],
                                  ResObj.Results[0].G[:-1],
                                  ResObj.Results[1].G[:-1],
                                  ResObj.Results[0].D[:-1],
                                  ResObj.Results[1].D[:-1],
                                  ResObj.tau_avg[0],
                                  ResObj.tau_avg[1],
                                  ResObj.Neq_comb,
                                  ResObj.Results[0].rd,
                                  ResObj.Results[1].rd)).T
        footerprof = ('Profile: {}, '.format(soilprofile.profile_name) +
                      'Motion 1: {}, Motion2: {} '.format(
                          str(ptmotions[0]['Name']),
                          str(ptmotions[1]['Name'])))
        if self.CEUS:
            filenameprof = (outdir + '{}_profile.csv'.format(
                                  str(ptmotions[0]['Name'])+
                                  str(ptmotions[0]['Bin'][:7])))
        else:
            filenameprof = (outdir + '{}_profile.csv'.format(
                                  str(ptmotions[0]['Name'])[7:-4]))
        np.savetxt(filenameprof, outarrayprof, delimiter=',',
                   newline='\r\n', header=','.join(headerlinesprof),
                   footer=footerprof, comments='')

        #NPTS = ptmotions[0]['NPTS']
        NPTS = ptmotions[0]['MotionData'].shape[0]
        dt = ptmotions[0]['TimeStep']
        time = np.arange(NPTS) * dt
        headerlinesth = ['Time (s)', 'Base Acc 1 (g)',
```

```
                        'Base Acc 2 (g)',
                        'Surface Acc 1 (g)',
                        'Surface Acc 2 (g)',
                        'NPTS={}'.format(NPTS),
                        'Motion 1: {}, Motion 2: {}'.format(
                        str(ptmotions[0]['Name']),
                        str(ptmotions[1]['Name'])),
                        '{}'.format(soilprofile.profile_name)]
        outarrayth = np.vstack((time,
                            ptmotions[0]['MotionData'],
                            ptmotions[1]['MotionData'],
                            ResObj.Results[0].calcAcc(0.,
                                MaxOnly=False,
                                domain='time',
                                MotType='within').ravel(),
                            ResObj.Results[1].calcAcc(0.,
                                MaxOnly=False,
                                domain='time',
                                MotType='within').ravel())).T
        if self.CEUS:
            filenameth = (outdir + '{}_timehist.csv'.format(
                            str(ptmotions[0]['Name'])+
                            str(ptmotions[0]['Bin'][:7])))
        else:
            filenameth = (outdir + '{}_timehist.csv'.format(
                            str(ptmotions[0]['Name'])[7:-4]))
        np.savetxt(filenameth, outarrayth, delimiter=',',
                newline='\r\n', header=','.join(headerlinesth),
                comments='')

    def PM(self, ResObj, PlaceHolder, Motions):
        for j in range(2):
            ptmotions = Motions.motions
            motstats = Motions.motstats

            entry = self.outtable.row
            entry['RecSeq'] = ptmotions[j]['RecSeqNum']

            if self.CEUS:
                if j == 0:
                    entry['PGA'] = motstats['PGA_H1']
                    entry['PGV'] = motstats['PGV_H1']
                    entry['PGD'] = motstats['PGD_H1']
                else:
                    entry['PGA'] = motstats['PGA_H2']
                    entry['PGV'] = motstats['PGV_H2']
                    entry['PGD'] = motstats['PGD_H2']
                entry['Bin'] = motstats['Bin']
                #entry['Motion0'] = str(ptmotions[0]['Station_Name']) #for CEUS: Station_Name
                entry['EQEvent'] = motstats['EQ_name']
            else:
                entry['EpicD'] = motstats['EpiD']
                entry['Mag_Type'] = motstats['Mag_type']
                entry['Hypd'] = motstats['HypD']
                entry['J_BD'] = motstats['Joy_Boore']
                entry['RMSD'] = motstats['RmsD']
                entry['Vs30_pref'] = motstats['Vs30_pref']
```

```
                entry['NEHRP'] = motstats['NEHRP'] # StringCol(itemsize=1)
                entry['ThetaD'] = motstats['ThetaD']
                entry['PhiD'] = motstats['Phi_D']
                entry['X'] = motstats['X']
                entry['Y'] = motstats['Y']
                entry['Rake'] = motstats['Rake']
                entry['EQ_Mech'] = motstats['EQ_Mech']
                entry['Dip'] = motstats['Dip']
                entry['EQEvent'] = motstats['EQID']
                entry['PGA'] = motstats['PGA']
                entry['PGV'] = motstats['PGV']
                entry['PGD'] = motstats['PGD']

            entry['Motion'] = str(ptmotions[j]['Name']) #for CEUS: Station_Name
            entry['Eq_Mag'] = motstats['EQ_Mag']
            entry['CLSTD'] = motstats['ClstD']

            entry['T_d01'] = ResObj.Tb_01[j] #Bracketed duration (> 0.01g)
            entry['T_d05'] = ResObj.Tb_05[j] #Bracketed duration (> 0.05g)
            entry['T_tb95'] = ResObj.Ttb_95[j]   #Trifunac and Brady duration based on 5-95%
                 energy
            entry['T_tb75'] = ResObj.Ttb_75[j]   #Trifunac and Brady duration based on 5-95%
                 energy
            entry['I_a'] = ResObj.I_a[j]      # Arias Intensity
            entry['Ar'] = ResObj.Ar[j]
            entry['Artype'] = ResObj.Artype

            entry['amax_surf'] = ResObj.amax_surf[j]

            entry['PMNeq'] = ResObj.PMNeq[j]
            entry['PMNeqVS'] = ResObj.PMNeqVS
            entry['PMNeqVSdis'] = ResObj.PMNeqVSdis
            entry['Discard'] = ResObj.Discards[j]
            entry['m'] = ResObj.m
            entry['cutoff'] = ResObj.cutoff
            entry.append()
        self.outtable.flush()

def Runner(**kwargs):
    CEUS = kwargs.get('CEUS', True)
    combinecomps = kwargs.get('combinecomps', False)
    comb_alt = kwargs.get('comb_alt', True)
    archive = kwargs.get('archive', False)
    DC = kwargs.get('DC', 'DS')

    motions = Motions(CEUS=CEUS)
    profiles = Profiles(CEUS=CEUS)
    out = Output(CEUS, combinecomps, DC=DC, comb_alt=comb_alt,
           archive=archive)

    for mot in motions():
        if CEUS:
            pass
        else:
            if str(mot.col('Orient')[0]) == 'Vert':
                print('Contained a vertical component... Skipping!')
                continue
```

```
        if len(mot.col('Name')) < 2:
            print('Contained less than 2 components... Skipping!')
            continue
        DynObj = Dynamics(mot, combinecomps, DC=DC, comb_alt=comb_alt)
        for prof in profiles:
            retcode = DynObj(prof)
            #if retcode != 0:
                #pdb.set_trace()
            out(DynObj, profiles.group, motions)
        del DynObj

    out.close()
    motions.close()
    profiles.close()

def ProfCSVMaker(**kwargs):
    '''
    This writes out my input profiles as *.csv files
    '''
    CEUS = kwargs.get('CEUS', False)
    combinecomps = kwargs.get('combinecomps', False)
    DC = kwargs.get('DC', 'DS')

    motions = Motions(CEUS=CEUS)
    profiles = Profiles(CEUS=CEUS)
    #out = Output(CEUS, combinecomps, DC=DC)

    for mot in motions():
        if CEUS:
            pass
        else:
            if str(mot.col('Orient')[0]) == 'Vert':
                print('Contained a vertical component... Skipping!')
                continue
        if len(mot.col('Name')) < 2:
            print('Contained less than 2 components... Skipping!')
            continue
        DynObj = Dynamics(mot, combinecomps, DC=DC, ProfMaker=True)
        for prof in profiles:
            retcode = DynObj(prof)
            #if retcode != 0:
                #pdb.set_trace()
            #out(DynObj, profiles.group, motions)
        del DynObj
        break

    #out.close()
    motions.close()
    profiles.close()
```

J.5 Regression of n_{eq}

The file `NeqStats2.py` uses the `R` language and bootstrapping to obtain the regression coefficients for n_{eq}.

```
,
import sys
sys.path.append('/media/Storage/Documents/Python/Modules')
#import pdb
from time import gmtime, strftime
try:
    import ipdb as pdb
except:
    import pdb
import numpy as np
import pandas as pd
import matplotlib
matplotlib.rcParams.update({'font.size': 18, 'font.family': 'STIXGeneral',
     'mathtext.fontset': 'stix'})
import matplotlib.pyplot as plt
import statsmodels.api as sm
import pandas.rpy.common as com
import rpy2.robjects as robjects
from rpy2.robjects.packages import importr
base = importr('base')
stats = importr('stats')
lme4 = importr('lme4')
lattice = importr('lattice')
sys.path.append('/media/Storage/Documents/Python/GMPEs')
from AS_2014_nga import AS_2014
from BA_2014_nga import BA_2014
from CY_2014_nga import CY_2014
from CB_2014_nga import CB_2014
from numpy.random import normal
import pandoctable as pan


def Neq(**kwargs):
    comb = kwargs.get('comb', False)
    CEUS = kwargs.get('CEUS', False)
    if comb:
        if CEUS:
            infile = kwargs.get('infile',
                   'IO_CEUS_Comb23Mar2015.h5')
        else:
            infile = kwargs.get('infile',
                   'IO_WUS_Comb23Mar2015.h5')
    else:
        if CEUS:
            infile = kwargs.get('infile',
                   'IO_CEUS_Sing20Mar2015.h5')
        else:
            infile = kwargs.get('infile',
                   'IO_WUS_Sing19Mar2015.h5')
    fileout = 'NeqStats-out.txt'
```

```
    results0 = pd.read_hdf(infile, 'EquivLinIters')
    #PIcutoff = kwargs.get('PIcutoff', 50)
    Depthcutoff = kwargs.get('Depthcutoff', 100)
    if comb:
        results = results0[(results0['Depth'] < Depthcutoff) & (results0.Neq_comb > 0)]
    else:
        results = results0[(results0['Depth'] < Depthcutoff) & (results0.N_EQ > 0)]
    #results['EQEvent'] = results['EQEvent'].astype('category')
    eqevnt = pd.Categorical.from_array(results['EQEvent'])
    results['EQEvent'] = eqevnt.codes
    #results['EQEvent'] = eqevnt.labels
    #results['profiletype'] = results['profiletype'].astype('category')
    results['EQEvent'] = results['EQEvent'].astype(int)
    results['profiletype'] = results['profiletype'].astype(int)
    print('File In: {}'.format(infile))
    print('File Out: {}'.format(fileout))
    N = kwargs.get('N', 10)
    samplesize = kwargs.get('samplesize', 1000)
    print('Number of Iterations: {}'.format(N))
    print('Sample Size: {}'.format(samplesize))
    if comb:
        results['amax_surf'] = np.sqrt(results['amax_surf_0'].values *
                                       results['amax_surf_1'].values)
        results['N_EQ'] = np.maximum(results.Neq_comb.values, 1e-10)
    else:
        results.loc[:,'N_EQ'] = np.maximum(results.N_EQ.values, 1e-10)
    ycol = kwargs.get('ycol', 'N_EQ') #Assume this is log() for now
    mycolumns = kwargs.get('mycolumns',
                ['N_EQ','amax_surf','Eq_Mag','Depth',
              'EQEvent', 'profiletype', 'CLSTD', 'Vs'])
    if CEUS:
        formulas = [
                    'log(N_EQ) ~ log(amax_surf) + Eq_Mag +'+
                              '(1|profiletype)',
              'log(N_EQ) ~ Eq_Mag + log(CLSTD) + '+
                  '(1|profiletype)']
        lenresults = [3, 3]
    else:
        formulas = [
                    'log(N_EQ) ~ log(amax_surf) + Eq_Mag +'+
                              '(1|EQEvent) + (1|profiletype)',
              'log(N_EQ) ~ Eq_Mag + log(CLSTD) + '+
                  '(1|EQEvent) + (1|profiletype)']
        lenresults = [3, 3]
    mycoeffs = []
    stdevs = []
    means = []
    stdevsa = []
    meansa = []
    taupro = []
    tauEQ = []

    formulas = kwargs.get('formulas', formulas)
    lenresults = kwargs.get('lenresults', lenresults)
    #results = results[results['Vs30_pref']>= 760]

    for i in xrange(N):
```

```
        if i % 5 == 0:
            sys.stdout.write(' .')
            if i % 25 == 0:
                sys.stdout.write('{}'.format(i))
            sys.stdout.flush()
        ind = np.random.choice(results.index.values, samplesize, replace=False)

        subset = results.loc[ind, mycolumns]
        subset.dropna(inplace=True) # I have to do this because of missing CLSTD values
        for j,formula in enumerate(formulas):
            n = lenresults[j]
            if i == 0:
                mycoeffs.append(np.zeros((N,n), dtype=float))
                stdevs.append(np.zeros((N,)))
                means.append(np.zeros((N,)))
                stdevsa.append(np.zeros((N,)))
                meansa.append(np.zeros((N,)))
                taupro.append(np.zeros((N,)))
                if not CEUS:
                    tauEQ.append(np.zeros((N,)))

            # Do it with rpy2
            dfr = com.convert_to_r_dataframe(subset, True)
            test1=lme4.lmer(robjects.Formula(formula), data=dfr)


            testsum = base.summary(test1)
            coefs = np.array(testsum.rx2("coefficients"))[:n,0]
            varcor = com.convert_robj(testsum.rx2("varcor"))
            taupro[j][i] = varcor['profiletype']['(Intercept)']['(Intercept)'] ** 0.5
            if not CEUS:
                tauEQ[j][i] = varcor['EQEvent']['(Intercept)']['(Intercept)'] ** 0.5

            try:
                resids = (np.log(subset[ycol].values) - # Need to change this if predicting
                     something other than log(ycol)
                       np.array(stats.predict(test1,REform=robjects.r("NULL"))))
                residsa = (np.log(subset[ycol].values) - # Need to change this if predicting
                     something other than N_EQ
                       np.array(stats.predict(test1,REform=robjects.r("NA"))))
            except ValueError:
                pdb.set_trace()
            stdevs[j][i] = np.std(resids)
            means[j][i] = np.mean(resids)
            stdevsa[j][i] = np.std(residsa)
            meansa[j][i] = np.mean(residsa)
            mycoeffs[j][i,:] = coefs

            if kwargs.get('ranefs', False):
                randicts = com.convert_robj(robjects.r.ranef(test1))
                pft = randicts['profiletype']
                eqe = randicts['EQEvent']
                pft['profiletype'] = pft.index
                eqe['EQEvent'] = eqe.index
                subset = pd.merge(subset,pft)
                subset.rename(columns={'(Intercept)':'ranefpt{}'.format(j)}, inplace=True)
                subset = pd.merge(subset,eqe)
```

```
            subset.rename(columns={'(Intercept)':'ranefEQ{}'.format(j)}, inplace=True)
            plt.figure('EQ')
            plt.plot(subset.Eq_Mag, subset['ranefEQ{}'.format(j)],'o', color='0.3')
            plt.xlabel('$M_w$')
            plt.ylabel('Random Effect of EQEvent')
            plt.figure('pt')
            plt.plot(subset.profiletype, subset['ranefpt{}'.format(j)],'o')
            plt.xlabel('Profile Type')
            plt.ylabel('Random Effect of ProfileType')

            if j == 0:
                plt.figure('Resid vs amax')
                plt.semilogx(subset.amax_surf.values, resids, '.', color='0.3')
                plt.xlabel('PGA')
                plt.ylabel('Residuals')
                plt.figure('Resid vs Mw')
                plt.plot(subset.Eq_Mag.values, resids, 'o', color='0.3')
                plt.xlabel('$M_w$')
                plt.ylabel('Residuals')
            else:
                plt.figure('Resid vs CLSTD')
                plt.semilogx(subset.CLSTD.values, resids, 'o', color='0.3')
                plt.xlabel('$R$ (km)')
                plt.ylabel('Residuals')
                plt.figure('Resid vs Mw')
                plt.plot(subset.Eq_Mag.values, resids, 'o', color='0.3')
                plt.xlabel('$M_w$')
                plt.ylabel('Residuals')

        if kwargs.get('plots', False):
            print(np.std(resids))
            fig, axs = plt.subplots(nrows=2, ncols=1, num=j, sharex=False,
                    sharey=True)
            ms = 2.5
            alpha = 0.8
            marker = 'o'
            color = '0.2'
            linestyle = ''
            ylim = 2
            if j == 0:
                #plt.figure('Resid vs amax')
                axs[0].semilogx(subset.amax_surf.values[::10], resids[::10],
                        marker=marker, linestyle=linestyle, ms=ms, color=color,
                             alpha=alpha)
                xmin, xmax = subset.amax_surf.min(), subset.amax_surf.max()
                axs[0].plot([0.00001,5], [0,0], lw=0.8, color='0.30')
                axs[0].axis([0.001, 2, -ylim, ylim])
                axs[0].set_xlabel('$PGA$ (g)')
                axs[0].set_ylabel('Residuals')
                #plt.figure('Resid vs Mw')
                axs[1].plot(subset.Eq_Mag.values[::10], resids[::10],
                        marker=marker, linestyle=linestyle, ms=ms, color=color,
                             alpha=alpha)
                xmin, xmax = subset.Eq_Mag.min(), subset.Eq_Mag.max()
                axs[1].plot([xmin,xmax], [0,0], lw=0.8, color='0.30')
                axs[1].axis([xmin, xmax, -ylim, ylim])
                axs[1].set_xlabel('$M_w$')
```

```
                axs[1].set_ylabel('Residuals')
            else:
                #plt.figure('Resid vs CLSTD')
                axs[0].semilogx(subset.CLSTD.values[::10], resids[::10],
                       marker=marker, linestyle=linestyle, ms=ms, color=color,
                            alpha=alpha)
                xmin, xmax = subset.CLSTD.min(), subset.CLSTD.max()
                axs[0].plot([1,200], [0,0], lw=0.8, color='0.30')
                axs[0].axis([1, 200, -ylim, ylim])
                axs[0].set_xlabel('$R$ (km)')
                axs[0].set_ylabel('Residuals')
                #plt.figure('Resid vs Mw')
                axs[1].plot(subset.Eq_Mag.values[::10], resids[::10],
                       marker=marker, linestyle=linestyle, ms=ms, color=color,
                            alpha=alpha)
                xmin, xmax = subset.Eq_Mag.min(), subset.Eq_Mag.max()
                axs[1].plot([xmin,xmax], [0,0], lw=0.8, color='0.30')
                axs[1].axis([xmin, xmax, -ylim, ylim])
                axs[1].set_xlabel('$M_w$')
                axs[1].set_ylabel('Residuals')
            #plt.tight_layout()
            plt.subplots_adjust(wspace=0.2, hspace=0.55)
            plt.locator_params(axis='y', nbins=4)


            plt.show()

    # Write the results out
    if kwargs.get('PrintOut', True):
        #pdb.set_trace()
        with open(fileout, 'a') as f:
            f.write(strftime('%d%b%Y', gmtime())+ '\n')
            f.write('Data File: {}\n'.format(infile))
            f.write('Number of Iterations: {}\n'.format(N))
            f.write('Sample Size: {}\n'.format(samplesize))
            #f.write('PI<{}\n\n'.format(PIcutoff))
            f.write('z<{}m\n\n'.format(Depthcutoff))

            for i, formula in enumerate(formulas):
                f.write('Formula No. {}:\n'.format(i))
                f.write(formula + '\n\n')

                f.write('Table: Regression Coefficients\n\n')

                align = [('^','<')] + [ ('^','^') for j in range(lenresults[i]+1)]
                header = (['Approach', 'Database'] +
                       [ '$a_{}$'.format(j+1) for j in range(lenresults[i])])
                database = 'CEUS' if CEUS else 'WUS'
                #Component = 'Combined' if comb else 'Single'
                if comb:
                    Component = '23'
                else:
                    Component = '1'
                data = [tuple([Component, database] +
                    ['{:10.4g}'.format(val) for val
                        in np.median(mycoeffs[i], axis=0)])]#,
                        #tuple([' ' for j in range(lenresults[i]+2)])]
```

```
                fields = range(lenresults[i]+2)
                pan.table(f, data, fields, header, align)

                f.write('\n\n')

                f.write('Table: Standard Deviations of Regression Coefficients\n\n')

                header = (['Approach','Database'] + [ '$\sigma_{' +
                    'a_{}'.format(j+1) + '}$' for j in range(lenresults[i])])
                data = [tuple([Component, database] + ['{:10.4g}'.format(val) for val
                               in np.std(mycoeffs[i], axis=0)]),
                        #tuple([' ' for j in range(lenresults[i]+2)])
                        ]
                pan.table(f, data, range(lenresults[i]+2), header, align)

                f.write('\n\n')

                f.write('Table: Mean and Standard Deviations of $\epsilon$\n\n')

                header = ['Component','Database', 'With Random Effects', 'Without RE']
                align = [('^','<')] + [ ('^','^') for j in range(3)]
                data = [ ('', '', 'mean, stdev.', 'mean, stdev.'),
                        (Component, database,
                            '{:.4f},{:.4f}'.format(np.mean(means[i]),
                            np.mean(stdevs[i])),
                            '{:.4f},{:.4f}'.format(np.mean(meansa[i]),
                            np.mean(stdevsa[i])))]
                pan.table(f, data, range(4), header, align)
```

J.6 Regression of r_d

The file `rdstats.py` uses the `R` language and bootstrapping to obtain the regression coefficients for r_d.

,

```
import sys
sys.path.append('/media/Storage/Documents/Python/Modules')
#import pdb
from time import gmtime, strftime
try:
    import ipdb as pdb
except:
    import pdb
import numpy as np
import pandas as pd
import matplotlib
matplotlib.rcParams.update({'font.size': 18, 'font.family': 'STIXGeneral',
     'mathtext.fontset': 'stix'})
import matplotlib.pyplot as plt
import pandoctable as pan
from scipy.optimize import minimize
from scipy.stats import norm
```

```
import pandas.rpy.common as com
import rpy2.robjects as robjects
from rpy2.robjects.packages import importr
base = importr('base')
stats = importr('stats')
lme4 = importr('lme4')
lattice = importr('lattice')

def g_rd(*beta, **kwargs):
    '''
    This is for rd
    '''
    df = kwargs.get('df')

    Mw = kwargs.get('Mw', df.Eq_Mag.values)
    z = kwargs.get('z', df.Depth.values)

    a = np.exp(beta[0] + beta[1] * Mw )
    b = beta[2] + beta[3] * Mw
    g = (1.-a) * np.exp(-(z) / b) + a
    #g = 1./ (a + 1.) * np.exp(-(z) / b) + a

    return g

def g_rd_complex(*beta, **kwargs):
    '''
    This is for rd
    '''
    beta_Lasley6e = [-6.2, 0.4, -0.005, 1.3, -16, 2.3, -2.4, 2.8, 0.15]
    df = kwargs.get('df')
    amax = df['amax_surf'].values
    Mw = df['Eq_Mag'].values
    z = df['Depth'].values
    Vs12 = df.Vs12.values

    a = np.exp(beta[0] + beta[1] * Mw + beta[2] * np.log(Vs12) + beta[3] * amax )
    b = np.exp(beta[4] + beta[5] * np.log(Mw) + beta[6] * amax + beta[7] * np.log(Vs12))
    g = (1.-a) * np.exp(-(z) / b) + a
    #g = 1./ (a + 1.) * np.exp(-(z) / b) + a

    return g


def g_rd_2(*beta, **kwargs):
    '''
    This is for rd
    '''
    #beta_Lasley6ab = [ -4.5, 0.43, -13, 3.1, 0.15]
    df = kwargs.get('df')
    Mw = df['Eq_Mag'].values
    z = df['Depth'].values
    Vs12 = df.Vs12.values

    a = np.exp(beta[0] + beta[1] * Mw)
    b = beta[2] + beta[3] * np.log(Vs12)
    g = (1.-a) * np.exp(-(z) / b) + a
    #g = 1./ (a + 1.) * np.exp(-(z) / b) + a
```

```
74
75      return g
76
77  def g_rd_2a(*beta, **kwargs):
78      '''
79      This is for rd
80      '''
81      df = kwargs.get('df')
82      Mw = df['Eq_Mag'].values
83      z = df['Depth'].values
84      Vs12 = df.Vs12.values
85
86      a = np.exp(beta[0] + beta[1] * Mw + beta[2]* np.log(Vs12))
87      b = beta[3] + beta[4] * Mw + beta[5] * np.log(Vs12)
88      g = (1.-a) * np.exp(-(z) / b) + a
89      #g = 1./ (a + 1.) * np.exp(-(z) / b) + a
90
91      return g
92
93  def g_rd_2b(*beta, **kwargs):
94      '''
95      This is for rd
96      '''
97      df = kwargs.get('df')
98      Mw = df['Eq_Mag'].values
99      z = df['Depth'].values
100     Vs12 = df.Vs12.values
101
102     a = np.exp(beta[0] + beta[1] * Mw + beta[2]* (Vs12))
103     b = np.exp(beta[3] + beta[4] * Mw + beta[5] * (Vs12))
104     g = (1.-a) * np.exp(-(z) / b) + a
105     #g = 1./ (a + 1.) * np.exp(-(z) / b) + a
106
107     return g
108
109
110 def g_rd_2c(*beta, **kwargs):
111     '''
112     This is for rd
113     '''
114     df = kwargs.get('df')
115     Mw = df['Eq_Mag'].values
116     z = df['Depth'].values
117     Vs12 = df.Vs12.values
118
119     a = np.exp(beta[0] + beta[1] * Mw + beta[2]* (Vs12))
120     b = 123.5 * a
121     g = (1.-a) * np.exp(-(z) / b) + a
122     #g = 1./ (a + 1.) * np.exp(-(z) / b) + a
123
124     return g
125
126 def g_rd_med(*beta, **kwargs):
127     '''
128     This is for rd
129     '''
130     betaorig =[ -4.18869375e+00, 4.07780834e-01, -1.08253122e-03,
```

```
                -1.17358503e+00, 3.00320953e-01, 1.34786963e-02,
                1.24475915e-01]
    df = kwargs.get('df')
    Mw = df['Eq_Mag'].values
    z = df['Depth'].values
    Vs12 = df.Vs12.values

    a = np.exp(beta[0] + beta[1] * Mw + beta[2]* (Vs12))
    b = np.exp(beta[3] + beta[4] * Mw + beta[5] * Vs12)
    g = (1.-a) * np.exp(-(z) / b) + a
    #g = 1./ (a + 1.) * np.exp(-(z) / b) + a

    return g

def likelihood(beta, g, df, log, key, **kwargs):
    x = df[key].values

    sig = beta[-1]
    xpred = g(*beta, df=df)

    if log == True:
        like = - np.sum(np.log(1. / (sig * np.sqrt( 2 * np.pi))) -
               (xpred - x) ** 2 / (2 * sig ** 2))
    else:
        like = - np.prod(1. / (sig * np.sqrt( 2. * np.pi)) * np.exp(-(xpred - x) ** 2. /
             (2. * sig ** 2.)))
    #if log == True:
        #like = - np.sum(np.log(norm.pdf((xpred-x) / sig)))
    #else:
        #like = - np.prod(norm.pdf((xpred-x)/ sig ))

    return like

def calcRd(df, **kwargs):
    '''
    This function will iterate through all combos of profiletype and EQEvent
    '''
    beta = kwargs.get('beta', [ -4.49, 0.412, -46.3, 10, 0.5])
    g = kwargs.get('g', g_rd)
    method = kwargs.get('method', 'Nelder-Mead')
    coefflen = len(beta) - 1
    EQEvents = set(df.EQEvent.values)
    profiletypes = set(df.profiletype.values)
    coeffs = np.zeros((len(EQEvents), len(profiletypes), coefflen))
    outcoeffs = np.zeros(4)

    if kwargs.get('mixedeffects', False):
        for i,EQ in enumerate(EQEvents):
            for j,prof in enumerate(profiletypes):
                mydf = df[(df['EQEvent'] == EQ) & (df['profiletype'] == prof)]
                if len(mydf) < 30:
                    coeffs[i,j,:] = np.nan
                    print('i: {}, j:{}.. Too few points({})'.format(i,j, len(mydf)))
                    continue
                try:
                    results = minimize(likelihood, beta,
                                       args=(g,mydf,True,'rd'),
```

```
                                    method=method,
                                    options={'maxiter': 1000, 'disp': False})
                except:
                    pdb.set_trace()
                print(results.x)
                coeffs[i, j, :] = results.x[:-1]
        outcoeffs = [np.nanmean(coeffs[:,:,i]) for i in range(coefflen)]
        #outcoeffs = [np.median(coeffs[:,:,i]) for i in range(4)]

    else:
        try:
            results = minimize(likelihood, beta,
                               args=(g,df,True,'rd'),
                               method=method,
                               options={'maxiter': 1000, 'disp': False})
        except:
            pdb.set_trace()
        outcoeffs = results.x[:-1]
    resids = df.rd.values - g(*outcoeffs, df = df)
    return outcoeffs, resids

def calcRdARM(df, **kwargs):
    '''
    From the directions given by Dr. Rodriguez-Marek on 23 April 2015
    '''
    method = kwargs.get('method', 'Nelder-Mead')
    g = kwargs.get('g', g_rd)

    df['Rdst'] = df['rd'].values
    # Get the coefficients for maximum likelihood
    beta = kwargs.get('beta', [ -4.49, 0.412, -46.3, 10, 0.5])
    n = len(beta) - 1
    error = 2.5
    i = 0
    # Perform mixed effects on the following equation
    formula='delta ~ (1|EQEvent) + (1|profiletype)'
    while error > 0.05:
        i += 1
        if i > 50:
            print('{} iterations w/o convergence'.format(i))
            pdb.set_trace()
            return np.nan * np.ones(n), np.nan * np.ones(len(df))
        results = minimize(likelihood, beta,
                           args=(g,df,True,'Rdst'),
                           method=method,
                           options={'maxiter': 2000, 'disp': False})

        df['delta'] = df.rd.values - g_rd(*results.x, df=df)
        #print(df['delta'].describe())
        #print(error)
        #pdb.set_trace()
        # Do it with rpy2
        dfr = com.convert_to_r_dataframe(df, True)
        test1=lme4.lmer(robjects.Formula(formula), data=dfr)
        #testsum = base.summary(test1)
        #coefs = np.array(testsum.rx2("coefficients"))[:n,0]
        #print('{}:'.format(i),coefs)
```

```
        #print(results.x)

        # get the random effects out and
        randicts = com.convert_robj(robjects.r.ranef(test1))
        pft = randicts['profiletype']
        eqe = randicts['EQEvent']
        pft['profiletype'] = pft.index
        eqe['EQEvent'] = eqe.index
        df = pd.merge(df,pft)
        df.rename(columns={'(Intercept)':'ranefpt'}, inplace=True)
        df = pd.merge(df,eqe)
        df.rename(columns={'(Intercept)':'ranefEQ'}, inplace=True)
        df['Rdst'] = df['rd'] - df['ranefpt'] - df['ranefEQ']

        error = np.max(np.abs(beta - results.x)/results.x)
        beta = results.x
        df.drop(['ranefpt', 'ranefEQ'], inplace=True, axis=1)

        # Run the maximum likelihood on RDst
        #Calculate Delta again and iterate until something converges.
    return results.x[:-1], df['delta'].values

def calcRdCoeffs(**kwargs):
    funform = kwargs.get('funform', 'old')
    # THis is for the old Method with Mw only...
    if funform == 'old':
        g = g_rd
        beta = [ -4.49, 0.412, -46.3, 10, 0.5]
    elif funform == 'newsimp':
        g = g_rd_2
        beta = [ -5.5, 0.58, -178, 38.7, 0.15]
    elif funform == 'newsimp2':
        g = g_rd_2b
        #beta = [ 1.45778659e+00, 4.07121989e-01, -1.12987457e+00,
               #-2.42362033e+02, 3.41554683e+00, 4.70286054e+01,
               #1.28247701e-01]
        beta = [ -4.23574983e+00, 4.04362656e-01, -6.01823842e-04, -8.72882189e-01,
   2.45975834e-01, 1.33266544e-02, 1.25455003e-01]
    elif funform == 'newsimpc':
        g = g_rd_2c
        #beta = [ 1.45778659e+00, 4.07121989e-01, -1.12987457e+00,
               #-2.42362033e+02, 3.41554683e+00, 4.70286054e+01,
               #1.28247701e-01]
        beta = [ -3.76528156e+00, 3.43265161e-01, -7.11339416e-04, -3.74960396e-01,
   1.32108560e-01]
    elif funform == 'complex':
        g = g_rd_complex
        beta = [-6.2, 0.4, -0.005, 1.3, -16, 2.3, -2.4, 2.8, 0.15]
    elif funform == 'med':
        g = g_rd_med
        beta =[ -4.18869375e+00, 4.07780834e-01, -1.08253122e-03,
               -1.17358503e+00, 3.00320953e-01, 1.34786963e-02, 0.15]
    else:
        g = kwargs.get('g')
        beta = kwargs.get('beta')
    lenresults = len(beta) - 1
    comb = kwargs.get('comb', False)
```

```
    CEUS = kwargs.get('CEUS', False)
    if comb:
        if CEUS:
            infile = kwargs.get('infile',
                   'IO_CEUS_Comb23Mar2015.h5')
        else:
            infile = kwargs.get('infile',
                   'IO_WUS_Comb23Mar2015.h5')
    else:
        if CEUS:
            infile = kwargs.get('infile',
                   'IO_CEUS_Sing20Mar2015.h5')
        else:
            infile = kwargs.get('infile',
                   'IO_WUS_Sing19Mar2015.h5')
    fileout = 'rdStats-out.txt'
    results0 = pd.read_hdf(infile, 'EquivLinIters')
    #PIcutoff = kwargs.get('PIcutoff', 50)
    Depthcutoff = kwargs.get('Depthcutoff', 3555)
    if comb:
        results = results0[(results0['Depth'] < Depthcutoff) & (results0.Neq_comb > 0)]
    else:
        results = results0[(results0['Depth'] < Depthcutoff) & (results0.N_EQ > 0)]
    #results['EQEvent'] = results['EQEvent'].astype('category')
    eqevnt = pd.Categorical.from_array(results['EQEvent'])
    results['EQEvent'] = eqevnt.codes
    #results['EQEvent'] = eqevnt.labels
    #results['profiletype'] = results['profiletype'].astype('category')
    results['EQEvent'] = results['EQEvent'].astype(int)
    results['profiletype'] = results['profiletype'].astype(int)
    print('\n\nFile In: {}'.format(infile))
    print('File Out: {}'.format(fileout))
    N = kwargs.get('N', 10)
    samplesize = kwargs.get('samplesize', 1000)
    print('Number of Iterations: {}'.format(N))
    print('Sample Size: {}'.format(samplesize))
    if comb:
        results['rd'] = np.mean((results.rd_0.values, results.rd_1.values), axis=0)
        results['N_EQ'] = results.Neq_comb.values
        results['amax_surf'] = np.sqrt(results.amax_surf_0.values *
                                       results.amax_surf1.values)
    mycolumns = kwargs.get('mycolumns',
                   ['rd','Eq_Mag','Depth',
               'EQEvent', 'profiletype', 'amax_surf', 'Vs12', 'Vs30'])
    mycoeffs = np.zeros((N,lenresults), dtype=float)
    stdevs = np.zeros((N,))
    means = np.zeros((N,))
    allresids = np.zeros((N, samplesize))


    for i in xrange(N):
        if i % 5 == 0:
            sys.stdout.write(' .')
            if i % 25 == 0:
                sys.stdout.write('{}'.format(i))
            sys.stdout.flush()
        ind = np.random.choice(results.index.values, samplesize, replace=False)
```

```
358
359         subset = results.loc[ind, mycolumns]
360         #subset.dropna(inplace=True) # I have to do this because of missing CLSTD values
361         #n = 4
362
363         if kwargs.get('RegressType', 'Old') == 'ARM':
364             coefs, resids = calcRdARM(subset, g = g, beta=beta)
365         else:
366             coefs, resids = calcRd(subset, g = g, beta=beta,
367                    mixedeffects=False)
368         stdevs[i] = np.sqrt(np.nanvar(resids))
369         means[i] = np.nanmean(resids)
370         if np.any(np.abs(resids) == np.inf):
371             pdb.set_trace()
372             mycoeffs[i, :] = np.nan
373         else:
374             mycoeffs[i,:] = coefs
375         allresids[i, :] = resids
376
377         #Sanitize it
378         stdevs[stdevs == np.inf] = np.nan
379         means[means == np.inf] = np.nan
380         mycoeffs[mycoeffs == np.inf] = np.nan
381         allresids[allresids == np.inf] = np.nan
382         stdevs[stdevs == -np.inf] = np.nan
383         means[means == -np.inf] = np.nan
384         mycoeffs[mycoeffs == -np.inf] = np.nan
385         allresids[allresids == -np.inf] = np.nan
386
387
388         if kwargs.get('plots', False):
389             print(np.std(resids))
390             fig, axs = plt.subplots(nrows=2, ncols=1, num=j, sharex=False,
391                    sharey=True)
392             ms = 2.5
393             alpha = 0.8
394             marker = 'o'
395             color = '0.2'
396             linestyle = ''
397             ylim = 2
398             if j == 0:
399                 #plt.figure('Resid vs amax')
400                 axs[0].semilogx(subset.amax_surf.values[::10], resids[::10],
401                        marker=marker, linestyle=linestyle, ms=ms, color=color, alpha=alpha)
402                 xmin, xmax = subset.amax_surf.min(), subset.amax_surf.max()
403                 axs[0].plot([0.00001,5], [0,0], lw=0.8, color='0.30')
404                 axs[0].axis([0.001, 2, -ylim, ylim])
405                 axs[0].set_xlabel('$PGA$ (g)')
406                 axs[0].set_ylabel('Residuals')
407                 #plt.figure('Resid vs Mw')
408                 axs[1].plot(subset.Eq_Mag.values[::10], resids[::10],
409                        marker=marker, linestyle=linestyle, ms=ms, color=color, alpha=alpha)
410                 xmin, xmax = subset.Eq_Mag.min(), subset.Eq_Mag.max()
411                 axs[1].plot([xmin,xmax], [0,0], lw=0.8, color='0.30')
412                 axs[1].axis([xmin, xmax, -ylim, ylim])
413                 axs[1].set_xlabel('$M_w$')
414                 axs[1].set_ylabel('Residuals')
```

```
            else:
                #plt.figure('Resid vs CLSTD')
                axs[0].semilogx(subset.CLSTD.values[::10], resids[::10],
                       marker=marker, linestyle=linestyle, ms=ms, color=color, alpha=alpha)
                xmin, xmax = subset.CLSTD.min(), subset.CLSTD.max()
                axs[0].plot([1,200], [0,0], lw=0.8, color='0.30')
                axs[0].axis([1, 200, -ylim, ylim])
                axs[0].set_xlabel('$R$ (km)')
                axs[0].set_ylabel('Residuals')
                #plt.figure('Resid vs Mw')
                axs[1].plot(subset.Eq_Mag.values[::10], resids[::10],
                       marker=marker, linestyle=linestyle, ms=ms, color=color, alpha=alpha)
                xmin, xmax = subset.Eq_Mag.min(), subset.Eq_Mag.max()
                axs[1].plot([xmin,xmax], [0,0], lw=0.8, color='0.30')
                axs[1].axis([xmin, xmax, -ylim, ylim])
                axs[1].set_xlabel('$M_w$')
                axs[1].set_ylabel('Residuals')
            #plt.tight_layout()
            plt.subplots_adjust(wspace=0.2, hspace=0.55)
            plt.locator_params(axis='y', nbins=4)


    # Write the results out
    if kwargs.get('PrintOut', True):
        with open(fileout, 'a') as f:
            f.write(strftime('%d%b%Y', gmtime())+ '\n')
            f.write('Data File: {}\n'.format(infile))
            f.write('Number of Iterations: {}\n'.format(N))
            f.write('Sample Size: {}\n'.format(samplesize))
            #f.write('PI<{}\n\n'.format(PIcutoff))
            f.write('z<{}m\n\n'.format(Depthcutoff))

            f.write('Formula: {}:\n'.format(funform))
            f.write('g: {}\n\n'.format(g))
            #f.write(formula + '\n\n')

            f.write('Table: Regression Coefficients\n\n')

            align = [('^','<')] + [ ('^','^') for j in range(lenresults)]
            header = (['Database'] +
                   [ '$a_{}$'.format(j+1) for j in range(lenresults)])
            database = 'CEUS' if CEUS else 'WUS'
            data = [tuple([database] +
                ['{:10.4g}'.format(val) for val
                    in np.nanmedian(mycoeffs, axis=0)])]#,
                    #tuple([' ' for j in range(lenresults[i]+1)])]
            fields = range(lenresults+1)
            pan.table(f, data, fields, header, align)

            f.write('\n\n')

            f.write('Table: Standard Deviations of Regression Coefficients\n\n')

            header = (['Database'] + [ '$\sigma_{' +
                'a_{}'.format(j+1) + '}$' for j in range(lenresults)])
            data = [tuple([database] + ['{:10.4g}'.format(val) for val
                           in np.nanstd(mycoeffs, axis=0)]),
```

```
                tuple([' ' for j in range(lenresults+1)])]
        pan.table(f, data, range(lenresults+1), header, align)

        f.write('\n\n')

        f.write('Table: Mean and Standard Deviations of $\epsilon$\n\n')

        header = ['Database', 'With Random Effects']
        data = [ ('', 'mean, stdev.'),
               (database,
                   '{:.4f},{:.4f}'.format(np.nanmean(means),
                   np.nanmean(stdevs))
                   )]
        pan.table(f, data, range(2), header, align)

        f.write('\n\n\n')


if __name__ == '__main__':
    calcRdCoeffs(CEUS=False, comb=False, infile='IO_WUS_Sing-DS-15Apr2015.h5',
           N=1000, samplesize=10000, funform='med')
    calcRdCoeffs(CEUS=True, comb=False, infile='IO_CEUS_Sing-DS-16Apr2015.h5',
           N=1000, samplesize=10000, funform='med')
    # Simp 2:
    calcRdCoeffs(CEUS=False, comb=False, infile='IO_WUS_Sing-DS-15Apr2015.h5',
           N=1000, samplesize=10000, funform='newsimp')
    calcRdCoeffs(CEUS=True, comb=False, infile='IO_CEUS_Sing-DS-16Apr2015.h5',
           N=1000, samplesize=10000, funform='newsimp')
    #calcRdCoeffs(CEUS=False, comb=False, infile='IO_WUS_Sing-DS-15Apr2015.h5',
           #N=10, samplesize=10000, funform='newsimp')
    #calcRdCoeffs(CEUS=True, comb=False, infile='IO_CEUS_Sing-DS-16Apr2015.h5',
           #N=10, samplesize=10000, funform='newsimp')
    # Complex:
    calcRdCoeffs(CEUS=False, comb=False, infile='IO_WUS_Sing-DS-15Apr2015.h5',
           N=1000, samplesize=10000, funform='complex')
    calcRdCoeffs(CEUS=True, comb=False, infile='IO_CEUS_Sing-DS-16Apr2015.h5',
           N=1000, samplesize=10000, funform='complex')
    # old:
    calcRdCoeffs(CEUS=False, comb=False, infile='IO_WUS_Sing-DS-15Apr2015.h5',
           N=1000, samplesize=10000)
    calcRdCoeffs(CEUS=True, comb=False, infile='IO_CEUS_Sing-DS-16Apr2015.h5',
           N=1000, samplesize=10000)
    # Combined:
    #calcRdCoeffs(CEUS=True, comb=True, infile='IO_CEUS_Comb-DS-17Apr2015.h5', N=1000,
        samplesize=10000)
    #calcRdCoeffs(CEUS=False, comb=True, infile='IO_WUS_Comb-DS-16Apr2015.h5', N=1000,
        samplesize=10000)
```

J.7 Fit of Limit-State Curves

The file `CaseHistoriesv3.py` calculates the dissipated energy for each case history in a database and fits a limit-state curve to the data. It also deals with the uncertainties and

compares with other prominent liquefaction evaluation procedures for SPT.

```
,
import sys
import os
if os.name == 'posix':
    sys.path.append('/media/Storage/Documents/Python/Modules')
else:
    sys.path.append('E:\Documents\Python\Modules')

import matplotlib
matplotlib.rcParams.update({'font.size': 18, 'text.usetex': True,
'legend.fontsize': 16})
#matplotlib.rcParams.update({'font.size': 18, 'font.family': 'STIXGeneral',
       #'mathtext.fontset': 'stix',
       #'legend.fontsize': 16})
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.ticker import MaxNLocator
import numpy as np
import pandas as pd
from scipy.stats import norm
try:
    import ipdb as pdb
except:
    import pdb
from scipy.optimize import minimize, curve_fit
from scipy.interpolate import interp1d, RectBivariateSpline
import degrcurv as dc
import pandoctable as pan

def strip(text):
    try:
        return text.strip()
    except AttributeError:
        return text

def getDegrad(Degrad, PI, sigmeff, **kwargs):
       '''
       Outputs a 1D array (gam = shear strain (decimal) and 2- 2D arrays
       (Gratio = G/Gmax and damping (decimal)) that represent the shear
       modulus and damping degradation curves. It depends on the module
       degrcurv, so use that if you want to customize it. Or, create your own
       degradation curves.
       Right now, Degrad must equal 'IZ' for the Ishibashi and Zhang (1993) or
       'DS' for Darendeli and Stokoe (2001) curves.

       sigmeff must be in kPa
       '''
       #straindef = np.array([0.000001, 0.000003, 0.00001, 0.00003, 0.0001,
                           #0.0003, 0.001, 0.003, 0.01, 0.99,10,100])
       #straindef = np.logspace(-6, 0, 100) / 100.
       straindef = np.logspace(-5, 0., 300) / 100. # Darendeli (2001): pg.228 (258)
       if Degrad == 'DS':
           if np.isscalar(PI):
               PI = np.array([PI])
```

```
            OCR = kwargs.get('OCR',1.)
            soil = kwargs.get('soil', 1.)
            N = kwargs.get('N', 10.)
            freq = kwargs.get('freq',1.)
            Dmintype = kwargs.get('DS_Dmin')
            if np.isscalar(OCR):
                OCR = np.ones(len(PI),dtype=float) * OCR
            if np.isscalar(soil):
                soil = np.ones(len(PI),dtype=int) * soil
            if np.isscalar(N):
                N = np.ones(len(PI),dtype=float) * N
            if np.isscalar(freq):
                freq = np.ones(len(PI),dtype=float) * freq
        if np.isscalar(sigmeff):
            sigmeff = np.ones(len(PI),dtype=float) * sigmeff
        gam = kwargs.get('gam',straindef)
        damping = np.zeros([len(PI),len(gam)],dtype=float)
        Gratio = np.zeros([len(PI),len(gam)],dtype=float)
        for i in xrange(len(PI)):
            if Degrad == 'IZ':
                gam, Gratio[i,:], damping[i,:] = dc.IZ_1993(gam = gam,
                                                    PI=PI[i],
                                                    sigm=sigmeff[i])
            elif Degrad == 'DS':
                gam,Gratio[i],damping[i] = dc.DS_2001(gam = gam,
                                                PI=PI[i],
                                                sigm=sigmeff[i],
                                                OCR=OCR[i],
                                                soil=soil[i],
                                                N=N[i],
                                                frq=freq[i])

        return gam, Gratio, damping

def DobryIter(gam,Gratio,damping,Gmax, tau_avg,error=0.01):
    '''
    This function will estimate degraded damping and degraded modulus
    by iteration and using shear modulus and damping degradation curves:

        Inputs:
            gam    An array of shear strains (decimal) of length n
            Gratio An array (matrix) of shear modulus degradation curves of
                   shape m X n where m is the number of layers
            damping An array of damping degradation curves, also m X n
        Outputs: Two arrays: G and D, each of length m.

    '''
    try:
        len(tau_avg)
    except TypeError:
        tau_avg = [tau_avg]
        Gmax = [Gmax]
    G_ = np.zeros([len(tau_avg)],dtype=float)
    D_ = np.zeros([len(tau_avg)],dtype=float)
    gam_ = np.zeros([len(tau_avg)],dtype=float)
    for j in xrange(len(tau_avg)):
        if len(tau_avg) == 1:
```

```
            Damp = np.ravel(damping)
            Grat = np.ravel(Gratio)
        else:
            Damp = np.ravel(damping[j,:])
            Grat = np.ravel(Gratio[j,:])
        gamma = gam[0]
        gamma0 = gamma + 0.0000001

        Gratiocurv = interp1d(np.log10(gam), Grat,
                              kind='linear')
        Dcurv = interp1d(np.log10(gam), Damp,
                         kind='linear')
        thiserror = 1
        count = 0
        try:
            while thiserror > error:
                count += 1
                if count > 50:
                    print('Layer %s did not converge!'%str(j))
                    print('Error = {:.5f}'.format(thiserror))
                    break
                G = Gmax[j] * Gratiocurv(np.log10(gamma))
                gamma = tau_avg[j] / G
                thiserror = abs(gamma - gamma0) / gamma0
                gamma0 = gamma
            D = Dcurv(np.log10(gamma))
            gam__ = gamma
        except ValueError:
            if gamma > gam[-1]:
                G = Gmax[j] * Grat[-1]
                D = Damp[-1]
                gam__ = gam[-1]
            else:
                G = Gmax[j] * Grat[0]
                D = Damp[0]
                gam__ = gam[0]
            print('Interp error!')

        G_[j] = G
        D_[j] = D
        gam_[j] = gam__
    if len(tau_avg) == 1:
        return G_[0], D_[0], gam_[0]
    else:
        return G_, D_, gam_

def TokimatsuSeedDirect(gam,Gratio,damping,Gmax, tau_avg, **kwargs):
    '''
    This function will estimate degraded damping and degraded modulus
    by iteration and using shear modulus and damping degradation curves:

        Inputs:
            gam    An array of shear strains (decimal) of length n
            Gratio An array (matrix) of shear modulus degradation curves of
                   shape m X n where m is the number of layers
            damping An array of damping degradation curves, also m X n
            Gmax   The small strain shear modulus
```

```
            tau_avg The average shear stress
        Outputs: Three arrays: G, D, and gamma each of length m.

    '''
    returnError = kwargs.get('returnError', False)
    G_ = np.zeros(Gratio.shape[0],dtype=float)
    D_ = np.zeros(Gratio.shape[0],dtype=float)
    gam_ = np.zeros(Gratio.shape[0],dtype=float)
    ErrorList = []
    if np.isscalar(Gmax):
        Gmax = [Gmax]
    if np.isscalar(tau_avg):
        tau_avg = [tau_avg]
    for i in xrange(len(Gmax)):
        #if i == 208: pdb.set_trace()
        Damp = np.ravel(damping[i,:])
        Grat = np.ravel(Gratio[i,:])
        gam_Grat = gam * Grat
        gamGratiocurv = interp1d(np.log10(gam_Grat), np.log10(gam),
                                 kind='linear')

        Gratiocurv = interp1d(np.log10(gam), Grat,
                                 kind='linear')
        Dcurv = interp1d(np.log10(gam), Damp,
                             kind='linear')
        try:
            newgam = 10 ** (gamGratiocurv(np.log10(tau_avg[i] / Gmax[i])))
        except ValueError:
            #if newgam > gam_[-1]:
            G_[i] = Gmax[i] * Grat[-1]
            D_[i] = Damp[-1]
            gam_[i] = gam[-1]
            print('Gamma exceeds range!, i={}'.format(i))
            ErrorList.append(i)
        else:
            switch=False
            G_[i] = Gmax[i] * Gratiocurv(np.log10(newgam))
            D_[i] = Dcurv(np.log10(newgam))
            gam_[i] = newgam
    if returnError:
        if len(G_) == 1:
            return G_[0], D_[0], gam_[0], ErrorList
        else:
            return G_, D_, gam_, ErrorList
    else:
        if len(G_) == 1:
            return G_[0], D_[0], gam_[0]
        else:
            return G_, D_, gam_

def Neq_sl(PGA, Mw, z, CEUS=False, Comb=False):
    '''
    Number of equivalent cycles using my relationship.
    '''
    CombinedWUS    = '  0.4208 |   -0.3989 |    0.2457 '
    CombinedCEUS   = '   1.209 |   -0.5448 |    0.1363 '
    SingleWUS      = ' -0.3643 |   -0.4105 |    0.2553 '
```

```
    SingleCEUS     = ' 0.4654 |   -0.5626 |    0.1423 '

    ErrorCombinedWUS   =0.4808
    ErrorCombinedCEUS  =0.5902
    ErrorSingleWUS     =0.5309
    ErrorSingleCEUS    =0.6366

    if Comb:
        if CEUS:
            a = [float(val) for val in CombinedCEUS.split('|')]
            sig = ErrorCombinedCEUS
        else:
            a = [float(val) for val in CombinedWUS.split('|')]
            sig = ErrorCombinedWUS
        return np.exp(a[0] + a[1] * np.log(PGA) + 0. * z + a[2] * Mw), sig
    else:
        if CEUS:
            a = [float(val) for val in SingleCEUS.split('|')]
            sig = ErrorSingleCEUS
        else:
            a = [float(val) for val in SingleWUS.split('|')]
            sig = ErrorSingleWUS
        # Multiply by two for both components of motion
        return np.exp(a[0] + a[1] * np.log(PGA) + 0. * z + a[2] * Mw)*2, sig

class CaseHists(object):
    '''
    This object holds the information and performs all the initial calcs
    on the case history database file. It prepares it for the limit-state
    fitting.
    '''
    def __init__(self, **kwargs):
        self.kwargs = kwargs
        self.unit_wt_below = kwargs.get('unit_wt_below', 19.5)
        self.unit_wt_above = kwargs.get('unit_wt_above', 17)

        self.sigeffvlimit = kwargs.get('sigeffvlimit', 40.) #kPa
        print('Lower limit of sigveff is {} kPa for Degrad Curves'.format(
            self.sigeffvlimit))

        self.altdWn= kwargs.get('altdWn', False)# Applies an exponential to the sigveff
        self.altCycles = kwargs.get('altCycles', False) # divides vals by 2
        self.N_Gmaxtype = kwargs.get('N_Gmaxtype', 'Seedetal')
        if 'df' in kwargs.keys():
            self.df = kwargs.get('df')
        else:
            datafile = kwargs.get('infile','./Boulanger er al 2012.csv')
            if 'FCrange' in kwargs.keys():
                FCs = kwargs.get('FCrange')
                df0 = pd.read_csv(datafile, converters = {'Liq' : strip})
                self.df = df0[(df0.FC >= FCs[0]) &
                        (df0.FC < FCs[1])].copy()
                self.df.index = np.arange(self.df.shape[0])
            else:
                self.df = pd.read_csv(datafile, converters = {'Liq' : strip})

        if 'sigv' not in self.df.columns:
```

```
282             self.df['sigv'], self.df['sigeffv'] = self.calcInsituStresses(**kwargs)
283         if kwargs.get('calcN160', True):
284             print('Calculating N160cs...')
285             Ncorrtype = kwargs.get('Ncorrtype', 'IB')
286             self.df['N160cs'], self.df['delN160'], Cr, Cn, Cs = calcN160cs(
287                    self.df.Nm.values,
288                    self.df.sigeffv.values,
289                    self.df.FC.values, self.df.z.values,
290                    Ce=self.df.CE.values, Cs=self.df.CS.values,
291                    Cb=self.df.CB.values,
292                    returnall=True, Ncorrtype=Ncorrtype)
293             self.df['N160'] = self.df['N160cs'] - self.df['delN160']
294             self.df['N60'] = self.df['N160'] / Cn
295             self.df['N160_Cetin'], Cr, Cm, Cs = calcN160cs(
296                    self.df.Nm.values,
297                    self.df.sigeffv.values,
298                    self.df.FC.values, self.df.z.values,
299                    Ce=self.df.CE.values, Cs=self.df.CS.values,
300                    Cb=self.df.CB.values,
301                    Cetin=True,
302                    returnall=True)
303             self.df['N60_Cetin'] = self.df['N160_Cetin'] / Cn
304         else:
305             if 'N160cs' in self.df.columns:
306                 self.df['delN160'] = self.df.N160cs - self.df.N160
307             else:
308                 self.df['delN160'] = calcDN160(self.df.FC.values)
309                 self.df['N160cs'] = self.df.delN160 + self.df.N160
310
311
312         self.calcDissEn(**kwargs)
313
314     def calcDissEn(self, **kwargs):
315         self.df['Neq'], self.df['Neqerror'] = Neq_sl(self.df.amax.values,
316                             self.df.Mw.values, self.df.z.values, CEUS=False,
317                             Comb=False)#Degrad=self.Degrad)
318         if self.altCycles:
319             print('Using altCycles..... Dividing by 2')
320             self.df['Neq'] = self.df['Neq'].values / 2.
321         self.Degrad = kwargs.get('Degrad', 'DS')
322
323         useVs = kwargs.get('useVs', False)
324         self.useVs = useVs
325
326         if kwargs.get('rd') == 'Lasley':
327             self.df['rd'] = self.calcrd()
328         elif kwargs.get('rd') == 'IB':
329             self.df['rd'] = self.calcrd(type='IB')
330         self.df['ko'], self.df['phi'] = self.calcKo()
331         self.df['tauavg'] = self.calctauavg()
332         self.df['sigmeff'] = self.calcsigmeff(limit=False)
333         self.df['sigmeff_alt'] = self.calcsigmeff(limit=True)
334         self.df['Gmax'] = self.calcGmax(limit=False)
335         if self.Degrad == 'IZ':
336             print('Ishibashi and Zhang Degrad Curves')
337             self.gam, self.Gratio, self.damping = getDegrad('IZ',
338                    np.zeros(self.df['tauavg'].values.shape),
```

```
                self.df['sigmeff_alt'].values)
        else:
            print('Darendeli and Stokoe Degrad Curves')
            FC = self.df.FC.values
            #Soiltype = np.zeros(FC.shape)
            Soiltype = np.ones(FC.shape)
            Soiltype[(FC>10) & (FC<50)] = 2
            Soiltype[FC >= 50] = 3
            self.df['DS_soil'] = Soiltype
            self.gam, self.Gratio, self.damping = getDegrad('DS',
                   np.zeros(self.df['tauavg'].values.shape),
                   self.df['sigmeff_alt'].values,
                   soil=Soiltype, N = 2. * self.df.Neq.values
                   )
        if useVs:
            self.calcGmaxVs()
        if kwargs.get('GDtype', 'direct') == 'direct':
            print('TokiSeedDirect')
            self.df['G'], self.df['D'], self.df['gamma'], self.TS_Errors =\
                   TokimatsuSeedDirect(
                   self.gam, self.Gratio, self.damping, self.df.Gmax.values,
                   self.df.tauavg.values, returnError=True)
        else:
            print('DobryIter')
            self.df['G'], self.df['D'], self.df['gamma'] = DobryIter(self.gam,
                   self.Gratio, self.damping, self.df.Gmax.values,
                   self.df.tauavg.values)
        self.calcGDerror()

        self.df['dW'] = (2 * np.pi * self.df['D'] * self.df['tauavg'] ** 2
                         * self.df['Neq'] / self.df['G'])

        if 'Liq' in self.df.columns:
            # You need 'Liq' for fitting the curve, but not for forward analysis
            Liq = self.df['Liq'].values
            self.gsign = np.ones(self.df.Liq.shape)
            self.ind1 = np.where(Liq == 'Yes')
            self.ind2 = np.where(Liq == 'No')
            self.ind3 = np.where(Liq == 'Marginal')
            self.Liq2 = np.zeros(Liq.shape, dtype=float)
            self.Liq2[self.ind1] = 1.
            self.Liq2[self.ind2] = 0.
            self.Liq2[self.ind3] = 0.5
            self.gsign[self.ind1] = -1
            self.gsign[self.ind3] = 0
            self.gsign[self.ind2] = 1
        self.altwt = np.ones(self.df.Neq.values.shape)
        if useVs:
            self.update_altwt(kwargs.get('altwt', 0.3))
            self.dfwVs = self.df[~np.isnan(self.df.Vs1.values)]
            self.dfwVs.index = range(len(self.dfwVs))
        self.calcVsError()
        if self.altdWn:
            print('using altdWn.......')
            self.df['dWn'] = self.df['dW'] / self.df['sigeffv'] ** 1.11
        else:
            self.df['dWn'] = self.df['dW'] / self.df['sigeffv']
```

```
            #self.df['logdWn'] = np.log(2 * np.pi * 9.81 * 0.65 ** 2 *
                   #self.df['D'] * self.df['amax'] ** 2 *
                   #self.df['sigv'] ** 2 * self.df['rd'] ** 2
                   #* self.df['Neq'] / ( np.exp(self.df['logVs']) ** 2 *
                       #self.unit_wt_below * self.df['G'] / self.df['Gmax']
                       #* self.df['sigeffv']))
            self.df['logdWn'] = (
                np.log(2 * np.pi * 0.65 ** 2 * 9.81)
                + np.log(self.df['D']) # log D
                + 2 * np.log(self.df.amax) # log amax
                + 2 * np.log(self.df['sigv']) # logsigv
                + 2 * np.log(self.df['rd']) # logrd
                + np.log(self.df['Neq']) #logNeq
                - 2 * self.df['logVs'] # logVs
                - np.log(self.unit_wt_below) # Unit wt
                - np.log(self.df['G'] / self.df['Gmax']) # Grat
                - np.log(self.df['sigeffv']) # logsigeffv
                )
        self.df = self.calcUncertainty()


    def calcInsituStresses(self, **kwargs):
        '''
        This function assumes that I am looking at a profile.
        Thus, it won't work for a case history database. If you have a
        case history database, you better
        calc your sigvs and sigeffvs before you input the dataframe into
        the CaseHists object.
        '''
        print('Calculating my total and effective overburden stresses...')
        if 'gwt' in kwargs.keys():
            gwt = kwargs['gwt']
        elif 'gwt' in self.df.columns:
            gwt = self.df.gwt.values[0]
        else: gwt = 0.
        unitwta = self.unit_wt_above
        unitwtb = self.unit_wt_below
        z = self.df.z.values
        sigv = (np.maximum(0.,(z-gwt)) * unitwtb +
                                np.minimum(z,gwt) * unitwta)
        sigeffv = sigv - np.maximum(0, z-gwt) * 9.81
        return sigv, sigeffv


    def update_altwt(self, wt):
        '''
        This method allows me to update my weighting for case histories with
        measured shear wave velocities.
        '''
        self.altwt[np.isnan(self.df.Vs1pm.values)] = wt

    def calcGDerror(self):
        '''
        This returns the errors from the degradation curves. The
        The D and S values are given in Chapter 11? of Darendeli's
        Dissertation.
        '''
```

```
        if self.Degrad != 'DS':
            self.df['Derror'] = 1.
            self.df['Graterror'] = 0.1
            # These values are placeholders, I&Z don't give any values in their paper
        else:
            gam = np.array([0.00001,0.000022,0.0000484,0.0001,0.00022,
                0.000484,0.001,0.0022,0.00484,0.01,0.022,0.0484,0.1,
                0.22,0.484,1])
            Gratio = np.array([0.01836,0.01979,0.02254,0.02553,0.03026,
                0.03683,0.04523,0.05699,0.07116,0.08438,0.09454,0.09556,
                0.08784,0.07407,0.05961,0.04818])
            Damp = np.array([0.70766,0.7093,0.71422,0.72232,0.74132,0.78058,
                0.8525,0.99402,1.23299,1.55606,2.00485,2.49242,2.89875,
                3.23658,3.45147,3.55137])
            Gratiocurv = interp1d(np.log10(gam), Gratio,
                                  kind='linear')
            Dcurv = interp1d(np.log10(gam), Damp,
                             kind='linear')
            try:
                Derror = np.zeros(len(self.df))
                Graterror = np.zeros(len(self.df))
                gamma = self.df['gamma'].values*100 # gamma now in percent
                Derror[gamma <= gam[0]] = Damp[0]
                Graterror[gamma <= gam[0]] = Gratio[0]
                Derror[gamma >= gam[-1]] = Damp[-1]
                Graterror[gamma >= gam[-1]] = Gratio[-1]
                Derror[(gamma > gam[0]) & (gamma < gam[-1])] = Dcurv(
                                  np.log10(gamma[(gamma > gam[0]) &
                                      (gamma < gam[-1])]))
                Graterror[(gamma > gam[0]) & (gamma < gam[-1])] = Gratiocurv(
                                  np.log10(gamma[(gamma > gam[0]) &
                                      (gamma < gam[-1])]))
            except ValueError:
                pdb.set_trace()
            self.df['Derror'] = Derror / 100. # Convert into decimal
            self.df['Graterror'] = Graterror


    def calcrd(self, **kwargs):
        '''
        Two options to calculate the stress reduction coefficient.
        You know you want to use mine. Just do it!
        '''
        Mw = self.df.Mw.values
        z = self.df.z.values
        if kwargs.get('type', 'Lasley') == 'IB':
            alpha = -1.012 - 1.126 * np.sin(np.minimum(z, 34.) / 11.73 + 5.133)
            beta = 0.106 + 0.118 * np.sin(np.minimum(z, 34.) / 11.28 + 5.142)
            return np.exp(alpha + beta * Mw)
        else:
            coeftext = '-4.031 |   0.364 |   -28.47 |    7.798'
            #coeftext = '-5.173 |  0.5094 |   -16.81 |    5.591'
            coefs = [float(val) for val in coeftext.split('|')]
            alpha = np.exp(coefs[0] + coefs[1] * Mw)
            beta = coefs[2] + coefs[3] * Mw
            return np.minimum((1 - alpha) * np.exp(-z / beta) + alpha, 1.)

```

```
510     def calctauavg(self):
511         '''
512         Average shear stress from PGA. Assumes a factor of 0.65
513         '''
514         df = self.df
515         amax = df['amax'].values
516         sigv = df['sigv'].values
517         rd = df['rd'].values
518         return 0.65 * amax * sigv * rd
519 
520     def calcKo(self, **kwargs):
521         which = kwargs.get('phitype', 'old')
522         PhiVar = kwargs.get('PhiVar', 'N160cs')
523         N = self.df[PhiVar].values
524         sigeffv = self.df.sigeffv.values
525         if which == 'old':
526             phi = np.sqrt(15.4 * N) + 20. #Hatanaka and Uchida, 1996, Sabatini et al. 2002
527                 # np.sqrt(20. * N1) + 20. originally#Hatanaka and Uchida, 1996
528                 # np.sqrt(15.4 * N) + 20. # Sabatini et al. 2002
529         else:
530             # Kulhawy and Mayne 1990, see Holtz et al. Eqn. 12.5, pg. 559
531             # After Schmertmann 1975
532             # I could never get this to work, plus, uses reg. N value, not corrected for
                    overburden pressures.
533             phi = (np.arctan((N / (12.2 + 20.3 * sigeffv / 101.325))
534                    * np.pi / 180.
535                    ) )** 0.34
536         Ko = 1 - np.sin(phi * np.pi / 180.) # Jaky 1944, 1948
537         return Ko, phi
538 
539     def calcsigmeff(self, **kwargs):
540         '''
541         Calculates mean effective confining stress.
542         Make sure your ko values are right.
543         '''
544         df = self.df
545         ko = df['ko'].values
546         if kwargs.get('limit', True):
547             sigveff = np.maximum(self.sigeffvlimit, df['sigeffv'].values)
548         else:
549             sigveff = df['sigeffv'].values
550         return (1 + 2 * ko) * sigveff / 3.
551 
552     def calcGmax(self, **kwargs):
553         '''
554         Two options to calculate Gmax from SPT blow count.
555         '''
556         limit = kwargs.get('limit', False)
557         if self.N_Gmaxtype == 'Seedetal':
558             df = self.df
559             GmaxVar = self.kwargs.get('GmaxVar', 'N160cs')
560             #N = df['N160cs'].values #Seed et al. 1986 use N160
561             #N = df['N160'].values
562             #N = df['N60'].values #per Wair et al, you shouldn't used overburden-corrected
                    N values
563             N = df[GmaxVar].values
564             print('Using {} for the Seed et al. Gmax Equation'.format(GmaxVar))
```

```
565             if limit:
566                 sigmeff = df['sigmeff_alt'].values
567             else:
568                 sigmeff = df['sigmeff'].values
569             return 440. * N ** (1/3.) * 101.325 * (sigmeff/101.325) ** 0.5 # Seed et al.
                    1986
570         if self.N_Gmaxtype == 'SeedetalALT':
571             print('Using Seed et al. Alt Gmax equation...')
572             beta = self.kwargs.get('SeedGmaxBeta', [249.6, 0.2779, 0.35198])
573             df = self.df
574             GmaxVar = self.kwargs.get('GmaxVar', 'N160')
575             N = df[GmaxVar].values
576             if limit:
577                 sigmeff = df['sigmeff_alt'].values
578             else:
579                 sigmeff = df['sigmeff'].values
580             return beta[0] * N ** beta[1] * 101.325 * (sigmeff/101.325) ** beta[2] # Seed
                    et al. 1986
581         else: #Wair et al. (2012), PEER 2012/08
582             df = self.df
583             unit_wt = self.unit_wt_below
584             if limit:
585                 sigeffv = np.maximum(df.sigeffv.values, self.sigeffvlimit)
586             else:
587                 sigeffv = df.sigeffv.values
588             N = df['N60'].values
589             #N = df['N160cs'].values
590             Vs = 30 * N ** 0.23 * sigeffv ** 0.23 # For Sand
591             #Vs = 30 * N ** 0.215 * sigeffv ** 0.275 # For All Soils
592             Gmax = Vs ** 2 * unit_wt / 9.81
593             return Gmax
594
595
596     def calcGmaxVs(self):
597         '''
598         Calculating Gmax from the measured shear wave velocity.
599         Notice that I have to have a value of unit weight.
600         I'm using the same one as in the __init__ function
601         '''
602         df = self.df
603         # From foundations 1:
604         # Getting the Gmax from Vs1, but first I have to get the unit weight
605         # of the soil.
606         Vs = df.Vs.values
607         N160cs = df.N160cs.values
608         Gmax2 = np.copy(df.Gmax.values)
609         sigv = df.sigv.values
610         z = df.z.values
611         if self.kwargs.get('myunitwt', 'assumed') == 'fromSPT':
612             for i, vs in enumerate(Vs):
613                 if np.isnan(vs):
614                     pass
615                 else:
616                     N = N160cs[i]
617                     if N < 4:
618                         pass
619                     elif N < 10:
```

```
                    unitwt = 117.4 + (N - 4.) * (127.4 - 117.4) / (10. - 4.)
                elif N < 30:
                    unitwt = 122.4 + (N - 10.) * (132.4 - 122.4) / (30. - 10.)
                elif N < 50:
                    unitwt = 127.4 + (N - 30.) * (147.4 - 127.4) / (50. - 30.)
                else: # If it is above the range, I'll just use the Gmax calculated from
                     N160cs
                    continue
                Gmax2[i] = vs ** 2 * unitwt / 62.4 * 9.81 / 9.81
            self.df['Gmax'] = Gmax2
        else:
            unit_wtabv = self.unit_wt_above
            unit_wtbelow = self.unit_wt_below#19.5 #Moss? See Green's email 6 April 2015
            df['sigv2'] = (df['gwt'] * unit_wtabv +
                    (df['z'] - df['gwt']) * unit_wtbelow)
            df['sigveff2'] = df['sigv2'] - 9.81 * (df['z'] - df['gwt'])
            df['Gmax2'] = df['Vs'] ** 2 * unit_wtbelow / 9.81
            self.df.loc[~np.isnan(Vs), ['Gmax']] = df.loc[~np.isnan(Vs),
                ['Gmax2']].values

    def calcVsError(self):
        '''
        This might be a little messed up: how to apply cov?
        to the logVs or to Vs?
        '''
        Gmax = self.df.Gmax.values
        if self.useVs:
            Vs = self.df['Vs1'].values
            Vs[np.isnan(Vs)] = np.sqrt(Gmax[np.isnan(Vs)] * 9.81 /
                                       self. unit_wt_below)
            #logVs = np.log(Vs)
            #logVsError = np.zeros(len(self.df))
            #logVsError[~np.isnan(self.df.Vs1pm.values)] = logVs[
                    #~np.isnan(self.df.Vs1pm.values)] * .25 #<<==========?????
            #logVsError[np.isnan(self.df.Vs1pm.values)] = (0.5 * logVs[np.isnan(
                                                    #self.df.Vs1pm.values)])
            VsError = np.zeros(len(self.df))
            VsError[~np.isnan(self.df.Vs1pm.values)] = Vs[
                    ~np.isnan(self.df.Vs1pm.values)] * .25 #<<==========?????
            VsError[np.isnan(self.df.Vs1pm.values)] = (0.5 * Vs[np.isnan(
                                                    self.df.Vs1pm.values)])
        else:
            #logVs = np.log(np.sqrt(Gmax * 9.81 / self.unit_wt_below))
            #logVsError = 0.5 * logVs # From Brady Cox
            Vs = np.sqrt(Gmax * 9.81 / self.unit_wt_below)
            VsError = 0.5 * Vs # From Brady Cox

        self.df['logVs'], self.df['logVserror'] = transformToLog(Vs, VsError)
        self.df['Vserror'] = VsError
        self.df['Vs'] = Vs


    def to_pickle(self, **kwargs):
        '''
        Allows me to send out my dataframe to a pickle.
        To read it in, use pandas.read_pickle(...)
        '''
```

```
676         if self.useVs:
677             fileout = kwargs.get('fileout',
678                 '/media/Storage/Documents/00-VtResearch/Thesis/Markdown/'+
679                 'Figures/LimitState/LimStDF_Vs.pickle')
680         else:
681             fileout = kwargs.get('fileout',
682                 '/media/Storage/Documents/00-VtResearch/Thesis/Markdown/'+
683                 'Figures/LimitState/LimStDF.pickle')
684         #import pickle
685         #limitstatedata = {'N160cs':df.N160cs.values,
686                           #'dW': df.dW.values,
687                           #'dWn': df.dWn.values,
688                           #'sigveff' : df.sigeffv.values,
689                           #'Liq': df.Liq.values}
690         #pickle.dump(limitstatedata, open('limStatedata.p', 'wb'))
691         self.df.to_pickle(fileout)
692 
693     def calcUncertainty(self, **kwargs):
694         '''
695         Estimates uncertainty in dWn and N160cs using the first order approx.
696         '''
697         df = self.df
698 
699         # N160cs first
700         FC = df['FC'].values
701         sigFC = kwargs.get('sigFC', 0.2 * FC) # Cetin (2000) pg. 65
702         N160 = df['N160'].values
703         delN160 = df['delN160'].values
704         sigN = kwargs.get('sigN', np.minimum(3, 0.2 * N160)) # Cetin (2000) pg. 64
705         sigN160cs = np.sqrt(sigN ** 2 +
706                (delN160 * (492.883 - 9.7 * FC) / (FC + 0.01) ** 3) ** 2 *
707                sigFC ** 2)
708         df['sigN160cs'] = sigN160cs
709 
710 
711         # Now look at dWn
712         unit_wt = kwargs.get('unit_wt', self.unit_wt_below)
713         rho = np.eye(9)
714         rho[0][1] = kwargs.get('rhoDamax', 0.)
715         rho[0][2] = kwargs.get('rhoDsigv', 0.) # should be non-zero?
716         rho[0][3] = kwargs.get('rhoDrd', 0.) # should be non-zero?
717         rho[0][4] = kwargs.get('rhoDNeq', 0.)
718         rho[0][5] = kwargs.get('rhoDVs', 0.)
719         rho[0][6] = kwargs.get('rhoDunit_wt', 0.) # should be non-zero?
720         rho[0][7] = kwargs.get('rhoDGrat', -0.5) # should be non-zero
721         rho[0][8] = kwargs.get('rhoDsigeffv', 0.) # should be non-zero
722 
723         rho[1][2] = kwargs.get('rhoAmaxsigv', 0.)
724         rho[1][3] = kwargs.get('rhoAmaxrd', 0.)
725         rho[1][4] = kwargs.get('rhoAmaxNeq', 0.)
726         rho[1][5] = kwargs.get('rhoAmaxVs', 0.)
727         rho[1][6] = kwargs.get('rhoAmaxunit_wt', 0.)
728         rho[1][7] = kwargs.get('rhoAmaxGrat', 0.)
729         rho[1][8] = kwargs.get('rhoAmaxsigeffv', 0.)
730 
731         rho[2][3] = kwargs.get('rhoSigvrd', 0.)
732         rho[2][4] = kwargs.get('rhoSigvNeq', 0.)
```

```
733         rho[2][5] = kwargs.get('rhoSigvVs', 0.)
734         rho[2][6] = kwargs.get('rhoSigvunit_wt', 1.) # Per Adrian's email 22 May 2015
735         rho[2][7] = kwargs.get('rhoSigvGrat', 0.) # should be non-zero
736         rho[2][8] = kwargs.get('rhoSigvsigeffv', -1.)# should be -1 if based on gwt alone
737
738         rho[3][4] = kwargs.get('rhordNeq', -0.434)#was -0.434, now 0.12 when logrderror is
                used
739         rho[3][5] = kwargs.get('rhordVs', 0.)
740         rho[3][6] = kwargs.get('rhordunit_wt', 0.)
741         rho[3][7] = kwargs.get('rhordGrat', 0.) # should be non-zero
742         rho[3][8] = kwargs.get('rhordsigeffv', 0.)
743
744         rho[4][5] = kwargs.get('rhoNeqVs', 0.)
745         rho[4][6] = kwargs.get('rhoNequnit_wt', 0.)
746         rho[4][7] = kwargs.get('rhoNeqGrat', 0.) # should be non-zero???? - both estimated
                from amax
747         rho[4][8] = kwargs.get('rhoNeqsigeffv', 0.)
748
749         rho[5][6] = kwargs.get('rhoVsunit_wt', 0.)
750         rho[5][7] = kwargs.get('rhoVsGrat', 0.)
751         rho[5][8] = kwargs.get('rhoVssigeffv', 0.)
752
753         rho[6][7] = kwargs.get('rhoUnitWtGrat', 0.)
754         rho[6][8] = kwargs.get('rhoUnitWtsigeffv', 1.) # Per Adrian's Email 22may 2015
755
756         rho[7][8] = kwargs.get('rhoGratsigeffv', 0.) # should be non-zero
757         Rho = rho + rho.T - np.eye(9) # To make it symmetric
758
759         # Now get my means and standard deviations
760         sig_amax = kwargs.get('sig_amax', 0.3*np.log(df.amax.values)) #Was 0.15, then 0.3
761         #df['nonlog_amax'], df['sig_amax'] = transformFromLog(np.log(df['amax'].values),
                sig_amax)
762         #df['nonlog_Neq'] , df['sig_Neq'] = transformFromLog(np.log(df['Neq'].values),
                df['logNeqerror'].values)
763         sig_rd = kwargs.get('sig_rd', 0.15) # From my rd functional form
764         sig_sigv = kwargs.get('sig_sigv', 0.1 * df.sigv.values) # See Uncertainties.ipynb
765         sig_unit_wt = kwargs.get('sig_unit_wt', 0.1 * unit_wt)# 0.1 per Moss diss, pg 57 #
                Changed to 0 per Adrian's email 22 may 2015
766         sig_sigeffv = kwargs.get('sig_sigeffv', 0.1 * df.sigeffv.values) # See
                Uncertainties.ipynb
767         df['logD'], df['logDerror'] = transformToLog(df['D'].values,
768                                             df['Derror'].values)
769         df['logGrat'], df['logGraterror'] = transformToLog(
770                                           df['G'].values/df['Gmax'].values,
771                                             df['Graterror'].values)
772         df['logsigv'], df['logsigverror'] = transformToLog(df['sigv'].values,
773                                             sig_sigv) # Need a sd for sigv here
774         df['logrd'], df['logrderror'] = transformToLog(df['rd'].values,
775                                             sig_rd) # from rdStats-out25Apr2015.txt,
                                                    assumes constant with depth, mag
776         #df['logVs'], df['logVserror'] = transformToLog(df['Vs'].values,
777                                             #df['Vserror'])
778         df['logunit_wt'], df['logunit_wterror'] =
                transformToLog(unit_wt,#df['unit_wt'].values,
779                                             sig_unit_wt)
780         df['logsigeffv'], df['logsigeffverror'] = transformToLog(df['sigeffv'].values,
781                                           sig_sigeffv) # Need a sd for sigeffv here
```

```
        rhosigsigeff = transformRhoToLog(
                                df['sigv'].values,
                                sig_sigv,
                                df['sigeffv'].values,
                                sig_sigeffv,
                                -1.)

        logdWnerror = np.zeros(len(df))
        for i, row in df.iterrows():
            dels = np.array([ 1./row['D'],
                              2.,
                              2./row['sigv'],
                              2./row['rd'],
                              1.,
                              -2.,#/row['logVs'], # Just added in this '/row['logVs']
                              -1./self.unit_wt_below,
                              -1./(row['G']/row['Gmax']),
                              -1./row['sigeffv']])
            # sigs is my array of standard deviations
            sigs = np.array([row['Derror'],
                              sig_amax[i],
                              sig_sigv[i],
                              sig_rd,
                              row['Neqerror'],
                              row['logVserror'],
                              sig_unit_wt,
                              row['Graterror'],
                              sig_sigeffv[i]])
            # Add in my correlation coefficients:
            Rho[2][8] = rhosigsigeff[i]
            Rho[8][2] = rhosigsigeff[i]
            # SIGMAdwn is a matrix with sig^2 on the diag, rhosigsig on off-diag
            SIGMAdWn = (Rho * sigs).T * sigs
            logdWnerror[i] = np.sqrt(np.dot(np.dot(dels, SIGMAdWn), dels))

            #This is an aside to check my calcs. See checkFORM()
            if len(df) == 1:
                self.SIGMA = SIGMAdWn
                #self.Rho = Rho
                self.MU = (
                np.log(2 * np.pi * 0.65 ** 2 * 9.81)
                + np.log(row['D']) # log D
                + 2 * np.log(row.amax) # log amax
                + 2 * np.log(row['sigv']) # logsigv
                + 2 * np.log(row['rd']) # logrd
                + np.log(row['Neq']) #logNeq
                - 2 * row['logVs'] # logVs
                - np.log(unit_wt)#row['unit_wt'] # Unit wt
                - np.log(row['G'] / row['Gmax']) # Grat
                - np.log(row['sigeffv']) # logsigeffv
                )
                self.MUs = np.array([
                np.log(row['D']), # log D
                np.log(row.amax), # log amax
                np.log(row['sigv']), # logsigv
                np.log(row['rd']), # logrd
                np.log(row['Neq']), #logNeq
```

```
                row['logVs'], # logVs
                np.log(unit_wt), # Unit wt
                np.log(row['G'] / row['Gmax']), # Grat
                np.log(row['sigeffv']) # logsigeffv
                ])

        df['logdWnerror'] = logdWnerror
        return df

def transformFromLog(mean, sig):
    '''
    For the mean and standard deviations of the distribution
    '''
    meanout = np.exp(mean + sig ** 2 / 2.)
    sigout = np.sqrt(np.exp(2 * mean + sig ** 2) * (np.exp(sig ** 2) - 1))
    return meanout, sigout # This is the stdev because I take the sqrt

def transformRhoFromLog(mean1, sig1, mean2, sig2, rho):
    return (np.exp(rho * sig1 * sig2) - 1) / (
        np.sqrt(np.exp(sig1 ** 2) - 1) * np.sqrt(np.exp(sig2 ** 2) - 1))

def transformToLog(mean, sig):
    meanout = np.log(mean ** 2 / np.sqrt(sig **2 + mean ** 2))
    sigout = np.sqrt(np.log(1 + sig ** 2 / mean ** 2))
    return meanout, sigout # This is the stdev because I take the sqrt

def transformRhoToLog(mean1, sig1, mean2, sig2, rho):
    return (1. / (sig1 * sig2) * np.log(1. + rho *
        np.abs((sig1 * sig2) / (mean1 * mean2))))

def calcDN160(FC, **kwargs):
    '''
    This is a hack for research purposes. Do not use for production;
    in the CaseHists object, don't give N160 but not N160cs in the
    input
    '''
    return np.exp(1.63 + 9.7 / (FC + 0.01) - (15.7 / (FC + 0.01)) **2 )


def calcN160cs(Nm, sigveff, FC, z, **kwargs):
    '''
    Calculates N160cs using Cetins or Idriss and Boulanger's procedures
    '''
    if kwargs.get('Cetin', False):
        Cetin = True
        #Crarr = np.array([0.7516507353,0.7840220588,0.8002095588,0.8384742647,
            #0.8708602941,0.8929485294,0.9165220588,0.9356875,0.9548566176,
            #0.9652132353,0.9770551471,0.9859926471,0.9905183824,0.9950441176,
            #0.9966176471,0.9967279412,0.9983345588,0.99955808824])
        #zarr_cr = np.array([3.675,4.05,4.275,4.875,5.55,6.15,7.05,8.025,9.075,
            #10.35,11.925,14.25,16.575,18.9,21,23.25,26.025,29.85])
        #f_cr = interp1d(zarr_cr, Crarr, bounds_error=False, fill_value=0.7)
        #Cr = f_cr(z+1.2) # Cetin assumes a rod stick up length of 1.2
        #Cr[z+1.2 >= 29.5] = 1.
        # From Cetin's spreadsheet SiteX_KOC.xlsx
        Cr = np.maximum(0.6, np.minimum(1.0, 0.48 + 0.225 * np.log(z + 1.2)))
    else:
```

```
        Cetin = False
        Cr = np.ones(Nm.shape)
        Cr[z+1.5<3] = 0.75
        Cr[(z+1.5 >= 3) & (z+1.5 < 4)] = 0.8
        Cr[(z+1.5 >= 4) & (z+1.5 < 6)] = 0.85
        Cr[(z+1.5 >= 6) & (z+1.5 < 10)] = 0.95
        # Assuming a rod stick up length of 1.5 m
    if 'Cb' in kwargs.keys():
        Cb = kwargs['Cb']
    else:
        Bore_diam = kwargs.get('Bore_diam', 100 * np.ones(Nm.shape))
        Cb = np.ones(Nm.shape)
        Cb[Bore_diam <= 115] = 1.
        Cb[Bore_diam <= 150] = 1.05
        Cb[Bore_diam > 150] = 1.15
    Ce = kwargs.get('Ce', 1.)
    Csopt = kwargs.get('Cs', 1.)

    error = 1
    n = Nm / 2.
    N160 = n
    i = 0
    while error > 0.0001:
        i += 1
        if i >25:
            pdb.set_trace()
        if Cetin:
            Cn = np.minimum(1.6, np.sqrt(101.325 / sigveff))
        else:
            alpha = 0.784 - 0.0768 * np.sqrt(np.minimum(n, 46))
            Cn = np.minimum(1.7, (101. / sigveff) ** alpha)
        if kwargs.get('calcCs', False):
            if Cetin:
                Cs = 1 + N160 / 100.
            else:
                Cs = np.maximum(1.1, np.minimum(1.3, 1 + n / 100.))
        else:
            Cs = Csopt

        if Cetin:
            N160 = Nm * Cn * Cs * Ce * Cb * Cr
            error = np.max(np.abs(n-N160)/N160)
            n = N160
        else:
            if kwargs.get('Ncorrtype', 'IB') == 'IB':
                DelN160 = np.exp(1.63 + 9.7 / (FC + 0.01) - (15.7 / (FC + 0.01)) **2 )
            else:
                N160 = Nm * Cn * Cs * Ce * Cb * Cr
                DelN160 = 0.025 * FC * N160
            N160cs = Nm * Cn * Cs * Ce * Cb * Cr + DelN160
            error = np.nanmax(np.abs(n-N160cs)/N160cs)
            n = N160cs
    if Cetin:
        if kwargs.get('returnall', False):
            return N160, Cr, Cn, Cs
        else:
            return N160
```

```
953     if kwargs.get('returnall', False):
954         return N160cs, DelN160, Cr, Cn, Cs
955     else:
956         return N160cs
957 
958 class CaseHistArtificial2(CaseHists):
959     '''
960     This provides an artificial case history object per Adrian's suggestions.
961     Notice that I create some fake data, then run it through the same
962     CaseHists object as above.
963     '''
964     def __init__(self, **kwargs):
965         #self.DetLiqAlt = kwargs.get('DetLiqAlt', False)
966         self.thetas = kwargs.get('thetas', [0.7, -15, 7])
967         self.N = kwargs.get('N', 1000)
968         self.g = kwargs.get('g', g_2)
969         gam_b = 19.5
970         gam_a = 17.0
971 
972 
973         # Populate Fake Data Frame
974         if 'InputVars' in kwargs:
975             pass
976         else:
977             amax = 10 ** np.random.uniform(-1, 0.01, self.N)
978             Nm = np.random.uniform(2,45, self.N)
979             z = np.random.lognormal(1.8, 0.5, self.N)
980             zw = np.min((np.random.uniform(0,3, self.N), z-0.5), axis=0)
981             FC = np.random.lognormal(1, 0.8, self.N)
982             Mw = np.random.normal(6.5, 0.6, self.N)
983         sigv = np.maximum(0.,(z-zw)) * gam_b + np.min((z,zw)) * gam_a
984         sigveff = sigv - np.maximum(0,(z-zw)) * 9.81
985         N160cs, DeltaN160cs, Cr, Cn, Cs = calcN160cs(Nm, sigveff, FC, z,
986                                                      returnall=True)
987 
988         Vs1 = np.ones(Nm.shape) * np.nan
989         Liq = 'Yes'
990         df = pd.DataFrame(dict(
991                       Mw = Mw,
992                       amax = amax,
993                       Liq=Liq,
994                       z = z,
995                       gwt = zw,
996                       sigv = sigv,
997                       sigeffv=sigveff,
998                       FC=FC,
999                       Nm=Nm,
1000                      N160cs = N160cs,
1001                      N160 = N160cs - DeltaN160cs,
1002                      delN160=DeltaN160cs,
1003                      Vs1=Vs1,
1004                      Vs1pm = Vs1))
1005        super(CaseHistArtificial2, self).__init__(df=df, rd='Lasley',
1006                                  Degrad = kwargs.get('Degrad','DS'),
1007                                  useVs=False,
1008                                  GDtype = kwargs.get('GDtype', 'direct'),
1009                                  calcN160=False)
```

```
        self.gkwargs = kwargs.get('gkwargs',
                  {'a': self.df.N160cs.values,
                  'b' : np.log(self.df.dWn.values),
                  'g' : self.g})
        self.alpha = np.random.uniform(0,1, self.N)
        Pliq = norm.cdf(-self.g(self.thetas, self.gkwargs) /self.thetas[-1])
        #Liq = np.empty(self.N, dtype=str)
        Liq = np.array(['Marginal' for i in range(self.N)])
        Liq[self.alpha <= Pliq] = 'Yes'
        Liq[self.alpha > Pliq] = 'No'
        self.df['Liq'] = Liq

        self.gsign = np.ones(self.df.Liq.shape)
        self.ind1 = np.where(Liq == 'Yes')
        self.ind2 = np.where(Liq == 'No')
        self.ind3 = np.where(Liq == 'Marginal')
        self.Liq2 = np.zeros(Liq.shape, dtype=float)
        self.Liq2[self.ind1] = 1.
        self.Liq2[self.ind2] = 0.
        self.Liq2[self.ind3] = 0.5
        self.gsign[self.ind1] = -1
        self.gsign[self.ind3] = 0
        self.gsign[self.ind2] = 1
        self.altwt = np.ones(self.N)

    def DetLiqAlt(self):
        '''
        This method incorporates input variable uncertainty into the
        determination of whether or not the synthetic point liquefied.
        '''
        like = OtherLikelihood(df=self.df)
        Pliq = like(self.thetas, self.gkwargs, self.df, ReturnProb=True)
        #Pliqalt = norm.cdf(-g(self.thetas, gkwargs) / self.thetas[-1])
        #Liq = np.array(['Marginal' for i in range(self.N)])
        Liq = np.array(['Marginal' for i in range(self.N)])
        Liq[self.alpha <= Pliq] = 'Yes'
        Liq[self.alpha > Pliq] = 'No'
        self.df['LiqOld'] = np.copy(self.df['Liq'].values)
        self.df['Liq'] = Liq

def plotFit2(beta, kwargs, df, **kwargs2):
    '''
    This plots the curve when I don't have a regression coefficient for logDwn.
    '''
    paramlist = 'acdefhijk'
    no_params = 0
    for key in kwargs:
        if key in paramlist:
            no_params += 1
    norm = kwargs2.get('norm',True)
    logDW = kwargs2.get('logDW', True)
    logN = kwargs2.get('logN', False)
    label = kwargs2.get('label')
    alpha = kwargs2.get('altwt',1.)
    invalues = kwargs2.get('invalues')
    xx = invalues.get('a', np.linspace(0, 28, 100))
    y = np.zeros(xx.shape)
```

```
    for i in range(no_params):
        if paramlist[i] == 'b':
            continue
        if invalues:
            y = y + beta[i] * invalues.get(paramlist[i])
        else:
            if i == 0:
                y = y + beta[i] * xx
            else:
                y = y + beta[i] * np.median(kwargs.get(paramlist[i]))
    y = y + np.ones(len(y)) * beta[i+1]
    #y = -y / beta[1]

    if logDW:
        y = np.exp(y)
    if norm:
        #y = y * 101.325#np.median(df.sigeffv.values)
        pass
    else:
        y = y / 50. #???

    if logN:
        xx = np.exp(xx)
    plt.figure(kwargs2.get('figname','limit-state'))
    plt.plot(xx,y, label=label, lw=3.5)

def plotBase(df, **kwargs):
    '''
    From a liqueraction case history dataset, this plots
    the points of liquefaction/non-liquefaction and marginal liquefaction
    on a plot of logdWn and N160cs
    '''

    Liq = df['Liq'].values
    ind1 = np.where(Liq == 'Yes')
    ind2 = np.where(Liq == 'No')
    ind3 = np.where(Liq == 'Marginal')
    plt.figure(kwargs.get('figname','limit-state'))
    x = kwargs.get('x', 'N160cs')
    xlabel = kwargs.get('xlabel', '$N_{1,60cs}$')
    plt.semilogy(df[x].values[ind1],df.dWn.values[ind1], 'ko')#, label='Liq.' )
    plt.plot(df[x].values[ind2],df.dWn.values[ind2], 'ws')#, label='No Liq.' )
    plt.plot(df[x].values[ind3],df.dWn.values[ind3], 'm*')#, label='Marginal')
    plt.xlabel(xlabel)
    plt.ylabel('$\Delta W / \sigma \'_v$')

def calcNashSutcliffe(Actual, Predicted):
    '''
    Returns the Nash-Sutcliffe model efficiency parameter thingy.
    Make sure to have Actual and Predicted in right order. It matters.
    '''
    E = (1 - (np.sum((Actual - Predicted) ** 2) /
           np.sum((Actual - np.mean(Actual)) ** 2))) #* 100 #%
    return E

def compareResults(g, inargs, gsign, **kwargs):
    '''
```

```
    This calculates a few stats on the fit of the liquefaction curve, and
    returns the results as a dictionary.

    inargs refers to the input arguments of the limit state curve, g.
    '''
    sigma = kwargs.get('sig')
    if sigma == None:
        try:
            sigma = inargs[0][-1]
        except:
            sigma = 1.
            print('Sigma =1 ')
    gout = g(*inargs)

    #Pliq = kwargs.get('Pliq')
    Pliq = 1 - norm.cdf(gout / sigma)
    Pact = np.ones(len(Pliq))
    Pact[gsign == 0] = 0.5
    Pact[gsign > 0] = 0.


    ind = np.where(np.sign(gout) != np.sign(gsign))[0]
    indfn = np.where((np.sign(gout) != np.sign(gsign)) &
                     (np.sign(gout) > 0))[0]
    indfp = np.where((np.sign(gout) != np.sign(gsign)) &
                     (np.sign(gout) <= 0))[0]

    try:
        no_incorrect = len(ind)
    except TypeError:
        pdb.set_trace()
        no_incorrect = 0
    no_correct = len(gout) - no_incorrect

    wrongs = gout[ind]

    falneg = len(np.where(wrongs > 0)[0]) # Predicted no liq, but liquefied
    falpos = len(np.where(wrongs < 0)[0]) # Predicted Liq, but did not liquefy
    #sumgwrong = np.sum(np.maximum(norm.cdf(wrongs/sigma)-0.5, 0))
    sumgwrong = np.sum(np.abs(norm.cdf(wrongs/sigma)-0.5))
    sumgfn = np.sum(np.abs(norm.cdf(gout[indfn]/sigma)-0.5))
    sumgfp = np.sum(np.abs(norm.cdf(gout[indfp]/sigma)-0.5))
    if kwargs.get('UseProb', False):
        print('Using Probs to calc E.')
        E = calcNashSutcliffe(Pact, Pliq)
    else:
        E = calcNashSutcliffe(gsign, np.sign(gout))
    results = { 'num_false_pos' : falpos,
                'num_false_neg' : falneg,
                'num_correct' : no_correct,
                'num_incorrect' : no_incorrect,
                'sum_g_err' : sumgwrong,
                'sum_g_fn' : sumgfn,
                'sum_g_fp' : sumgfp,
                'NashSutcliffe_E' : E
                }
    return results
```

```
1181
1182 def gIdrissBoul(df, **kwargs):
1183     '''
1184     Idriss and Boulanger limit state curve with input of a dataframe
1185     '''
1186
1187     x = df['N160cs'].values
1188     y = df['CSRm7.5'].values
1189     co = 2.67 #2.8
1190     g = (x / 14.1 + (x / 126) ** 2 - (x / 23.6) ** 3 + (x / 25.4) ** 4 - co) - np.log(y)
1191     # This form of g is from Boulanger and Idriss 2012. However, you can make another g
1192     # that takes the exp of the first parens and subtracts y. This would yield a smaller
1193     # sum_g_erro
1194     return g
1195
1196 def gIdrissBoul2(N160cs, CSR, **kwargs):
1197     '''
1198     Idriss and Boulanger limit state function with inputs of N160cs and CSR
1199     '''
1200     co = kwargs.get('co', 2.67) #2.8
1201     g = (N160cs / 14.1 + (N160cs / 126) ** 2 - (N160cs / 23.6) ** 3 +
1202            (N160cs / 25.4) ** 4 - co) - np.log(CSR)
1203     # This form of g is from Boulanger and Idriss 2012. However, you can make another g
1204     # that takes the exp of the first parens and subtracts y. This would yield a smaller
1205     # sum_g_erro
1206     return g
1207
1208 def calcLS_IdrissBoul2(Pliq, N160cs, **kwargs):
1209     '''
1210     This returns the CSR values corresponding to a prob of liquefaction and
1211     a [range] of N160cs values, using Boulanger and Idriss/idriss and boulanger
1212     curves
1213     '''
1214     co = kwargs.get('co', 2.67) #2.8
1215     siglnR = kwargs.get('siglnR', 0.17)
1216     out = np.exp(N160cs / 14.1 + (N160cs / 126) ** 2 - (N160cs / 23.6) ** 3 +
1217            (N160cs / 25.4) ** 4 - co + norm.ppf(Pliq) * siglnR)
1218     # This form of g is from Boulanger and Idriss 2012.
1219     return out
1220
1221 def gYoudEtal(df, **kwargs):
1222     '''
1223     Limit state given by Youd et al 2001
1224     '''
1225     x = df['N160cs'].values
1226     y = df['CSRm7.5'].values
1227     g = 1./(34 - x) + x/135 + 50 / (10*x + 45) ** 2 - 1./200. - y
1228
1229     return g
1230
1231 def calcLS_Youdetal(N160cs):
1232     x = N160cs
1233     CRR = 1./(34 - x) + x/135 + 50 / (10*x + 45) ** 2 - 1./200.
1234     return CRR
1235
1236 def calcYoudCSR(amax, sigv, sigveff, Mw, z, N160, FC):
1237     '''
```

```
    Calculates the corrected CSR according to the Youd et al. 2001 procedure.
    '''
    # Liao and Whitman (1986):
    rd = np.ones(len(z))
    rd[z<= 9.15] = 1.0 - 0.00765 * z[z <= 9.15]
    rd[(z > 9.15) & (z <= 23)] = 1.174 - 0.0267 * z[(z > 9.15) & (z <= 23)]
    rd[z> 23] = 1.174 - 0.0267 * 23 # This is a hack for deep stuff
    #FC = np.float(FC)

    alpha = np.zeros(N160.shape)
    alpha[FC > 5] = np.minimum(5., np.exp(1.76 - 190./ FC[FC > 5] **2))
    beta = np.ones(N160.shape)
    beta[FC > 5] = np.minimum(1.2, 0.99 + FC[FC>5] ** 1.5 / 1000.)
    N160cs = alpha + beta * N160

    MSF = 10 ** 2.24 / Mw ** 2.56
    Dr = 15 * np.sqrt(N160)

    f = np.minimum(0.8, np.maximum(0.6, 1 - 0.005 * Dr))
    Ksig = (sigveff / 101.325) ** (f - 1)
    #Ksig = np.minimum(1.0, (sigveff / 101.325) ** (0.7 - 1))
    CSR = 0.65 * amax * sigv * rd / (MSF * sigveff * Ksig)
    CRR = calcLS_Youdetal(N160cs)
    FS = np.minimum(2., CRR / CSR) # To agree with IB 2008

    return CSR, N160cs, FS

def gCetindf(df, **kwargs):
    '''
    Cetin's (2000 and maybe Cetin et al. 2004 or something) Limit-state curve
    '''
    beta = kwargs.get('beta',
           #[0.004, 13.79, 29.06, 3.82, 0.06, 44.29, 4.21]
           #[0.004, 13.32, 29.53, 3.70, 0.05, 44.97, 2.7] # Accountin for measurement
               errors
           [0.004, 13.32, 29.53, 3.70, 0.05, 16.85, 2.7] # From Cetin et al. 2004
           )
    try:
        N = kwargs.get('N160', df.N160_Cetin.values)
    except:
        pdb.set_trace()
        N = kwargs.get('N160', df.N160.values)
    amax=df.amax.values
    sigv = df.sigv.values
    z = df.z.values
    F = df.FC.values
    M = df.Mw.values
    T = df.sigeffv.values
    rd = rd_Cetin1(amax, M, z)
    S = 0.65 * amax * sigv * rd / T
    g = (N * (1 + beta[0] * F) - beta[1] * np.log(S) - beta[2] * np.log(M) -
        beta[3] * np.log(T/101.325) + beta[4] * F + beta[5])
    return g

def gCetin(CSR, N160, Mw, FC, sigeffv, **kwargs):
    '''
    Same as gCetindf, but not using a dataframe
```

```
    '''
    beta = kwargs.get('beta',
           #[0.004, 13.79, 29.06, 3.82, 0.06, 44.29, 4.21]
           #[0.004, 13.32, 29.53, 3.70, 0.05, 44.97, 2.7] # Accountin for measurement
               errors
           [0.004, 13.32, 29.53, 3.70, 0.05, 16.85, 2.7] # From Cetin et al. 2004
           )
    N = N160
    S = CSR
    F = FC
    M = Mw
    T = sigeffv
    g = (N * (1 + beta[0] * F) - beta[1] * np.log(S) - beta[2] * np.log(M) -
        beta[3] * np.log(T/100.) + beta[4] * F + beta[5])
    return g

def calcLS_Cetin(Pliq, N160, FC, Mw, sigveff):
    '''
    This is from Cetin et al. 2004

    Cetin, K. O., Seed, R. B., Der Kiureghian, A., Tokimatsu, K.,
        Harder, L. F., Kayen, R. E., and Moss, R. E. S. (2004).
        "Standard Penetration Test-Based Probabilistic and
        Deterministic Assessment of Seismic Soil Liquefaction
        Potential." Journal of Geotechnical and Geoenvironmental
        Engineering, 130(12), 1314-1340.

    Gives the CSR values of the limit state curve for the given input values.
    '''
    FC = np.minimum(35., FC)
    out = np.exp((N160 * (1 + 0.004 * FC) - 29.53 * np.log(Mw) - 3.70 *
           np.log(sigveff / 100.) + 0.05 * FC + 16.85 + 2.70 *
           norm.ppf(Pliq)) / 13.32)
    return out

def g_1(beta, **kwargs):
    '''
    This does not assume the regression coefficient for dWn is -1
    '''
    #no_params = len(beta) - 1
    paramlist = 'abcdefhijk'
    no_params = 0
    for key in kwargs:
        if key in paramlist:
            no_params += 1
    a = kwargs.get('a')
    myones = np.ones(a.shape, dtype=float)
    res = np.array([beta[i] * kwargs.get(paramlist[i]) if i < (no_params)
        else beta[i]*myones for i in range(no_params+1)])
    return np.sum(res, axis=0)

def g_2(beta, kwargs):
    '''
    Assumes the 'b' coefficient is -1
    '''
    #no_params = len(beta) - 1
    paramlist = 'acdefhijk'
```

```
    no_params = 0
    for key in kwargs:
        if key in paramlist:
            no_params += 1
    a = kwargs.get('a')
    myones = np.ones(a.shape, dtype=float)
    res = np.array([beta[i] * kwargs.get(paramlist[i]) if i < (no_params)
        else beta[i]*myones for i in range(no_params+1)])
    return np.sum(res, axis=0) - kwargs.get('b')

def calcLimStCurves(Prob, beta, kwargs, **this):
    '''
    Returns log dWn (or whatever) for a given limit state curve
    (see g_2). Assumes that regression coefficient 'b' is equal to -1.
    '''
    paramlist = 'acdefhijk'
    no_params = 0
    for key in kwargs:
        if key in paramlist:
            no_params += 1
    a = kwargs.get('a')
    myones = np.ones(a.shape, dtype=float)
    res = np.array([beta[i] * kwargs.get(paramlist[i]) if i < (no_params)
        else beta[i]*myones for i in range(no_params+1)])
    out = np.sum(res, axis=0) + norm.ppf(Prob) * beta[-1]
    cutoff = this.get('cutoff', 100.)
    out[a>cutoff] = 100.
    return out

def log_like(beta, kwargs, df, wt=1.):
    '''
    Log-likelihood function for minimization. Deterministic.
    '''
    #df = kwargs.get('df')
    g = kwargs.get('g', g_1)
    #wt2 = kwargs.get('wt2',0.)
    sig = beta[-1]
    Liq = df['Liq'].values
    ind1 = np.where(Liq == 'Yes')
    ind2 = np.where(Liq == 'No')
    ind3 = np.where(Liq == 'Marginal')


    eps = g(beta, kwargs) / sig #- wt2

    returnval = np.ones(Liq.shape, dtype=float) #* 0.#5
    #returnval[ind1] = norm.cdf(0, loc=eps[ind1])
    #returnval[ind2] = (1. - norm.cdf(0, loc=eps[ind2]))
    returnval[ind1] = norm.cdf(-eps[ind1])
    returnval[ind2] = norm.cdf(eps[ind2])
    #returnval[np.abs(eps) > 2.5] = 1
    try:
        retval = -np.sum(np.log(np.maximum(returnval, 1e-10) )* wt)
    except:
        pdb.set_trace()
    #return -np.sum(np.log(returnval))
    return retval
```

```
1407
1408 class OtherLikelihood(object):
1409     '''
1410     Following Adrian's guidance. This examines uncertainties
1411     in the input parameters.
1412     '''
1413     def __init__(self, **kwargs):
1414         self.df = kwargs.get('df')
1415         self.verbose = kwargs.get('verbose', False)
1416         self.voltype = kwargs.get('voltype', 'other')
1417
1418         # Assign uncertainties if I have a dataset, otherwise...
1419         if 'df' in kwargs.keys():
1420             self.isdf = True
1421             #self.sigy = kwargs.get('sigy', np.abs(np.log(self.df.dWn.values)*0.5) ) # in
                      ln space
1422             self.sigy = kwargs.get('sigy') # in ln space
1423             if np.any(self.sigy) == None:
1424                 self.sigy = self.df.logdWnerror.values
1425             self.sigx = kwargs.get('sigx') # Boulanger and Idriss used 0.15 and 0.2
1426             if np.any(self.sigx) == None:
1427                 #self.sigx = np.minimum(self.df.N160.values * 0.2,3)# Per Cetin (2000), pg.
                        64
1428                 self.sigx = self.df.sigN160cs.values
1429             self.liq = np.ones(len(self.df))
1430             self.liq[self.df.Liq.values == 'Yes'] = -1.
1431             self.speedup = kwargs.get('speedup', True)
1432         else:
1433             self.speedup = False
1434             self.isdf = False
1435             self.sigy = kwargs.get('sigy', 1.5) #* np.ones(len(self.df))
1436             self.sigx = kwargs.get('sigx', 7) # SPT blowcount error
1437
1438         # This will ignore dx and dy keyword args, as well as ranges
1439         # Here I am creating a separate grid for each data point, instead of
1440         # one large grid for all datapoints.
1441         self.altgrid = kwargs.get('altgrid', True)
1442         if self.altgrid & self.isdf:
1443             # grid_x_sds is the number of standard deviations to extend in
1444             # in each direction from the mean values.
1445             # The count is the number of spacings in grid
1446             self.grid_x_sds = kwargs.get('grid_x_sds', 4.5)
1447             self.grid_y_sds = kwargs.get('grid_y_sds', 4.5)
1448             self.grid_x_count = kwargs.get('grid_x_count', 35)
1449             self.grid_y_count = kwargs.get('grid_y_count', 35)
1450             if self.isdf:
1451                 # I'm assuming the variables here of logdWn and N160cs
1452                 # Will need to be modified in other cases, CPT, for example
1453                 xs = self.df.N160cs.values
1454                 ys = self.df.logdWn.values
1455                 if np.isscalar(self.sigy):
1456                     sigy = np.ones(len(self.df)) * self.sigy
1457                 else:
1458                     sigy = self.sigy
1459                 if np.isscalar(self.sigx):
1460                     sigx = np.ones(len(self.df)) * self.sigx
1461                 else:
```

```
                sigx = self.sigx
            self.x_range = np.zeros((len(self.df),2))
            self.y_range = np.zeros((len(self.df),2))
            self.dx = np.zeros(len(self.df))
            self.dy = np.zeros(len(self.df))
            for i, (x, y) in enumerate(zip(xs, ys)):
                self.x_range[i,:] = [
                       x - self.grid_x_sds * sigx[i],
                       x + self.grid_x_sds * sigx[i]]
                self.dx[i] = (2 * self.grid_x_sds * sigx[i] /
                              self.grid_x_count)
                self.y_range[i,:] = [
                       y - self.grid_y_sds * sigy[i],
                       y + self.grid_y_sds * sigy[i]]
                self.dy[i] = (2 * self.grid_y_sds * sigy[i] /
                              self.grid_y_count)
                grid = np.mgrid[
                       self.x_range[i][0]:self.x_range[i][1]:self.dx[i],
                       self.y_range[i][0]:self.y_range[i][1]:self.dy[i]]
                xgrid, ygrid = grid[0], grid[1]
                if i == 0:
                    self.xgrid = np.zeros((len(self.df), xgrid.shape[0],
                                           xgrid.shape[1]))
                    self.ygrid = np.zeros((len(self.df), ygrid.shape[0],
                                           ygrid.shape[1]))
                try:
                    self.xgrid[i,
                           :self.grid_x_count,
                           :self.grid_x_count] = xgrid[:self.grid_x_count,
                                           :self.grid_x_count]
                    self.ygrid[i,
                           :self.grid_y_count,
                           :self.grid_y_count] = ygrid[:self.grid_y_count,
                                           :self.grid_y_count]
                except ValueError:
                    pdb.set_trace()
        else:
            # Grid spacing
            self.dx = kwargs.get('dx', .12)
            self.dy = kwargs.get('dy', .7)
            # Range of grid:
            self.x_range = kwargs.get('x_range', (0, 70))
            self.y_range = kwargs.get('y_range', (-15, 6))
            # Construct my grid
            self.grid = np.mgrid[self.x_range[0]:self.x_range[1]:self.dx,
                                 self.y_range[0]:self.y_range[1]:self.dy]
            self.xgrid, self.ygrid = self.grid[0], self.grid[1]

        # This is assuming x= N160cs, and y = DWn, to speed things up
        # If self.speedup is true, I calculate fxfy once at the beginning
        # This will take a bit more memory, though.
        if self.isdf & self.speedup:
            if np.isscalar(self.sigy):
                self.sigy = np.ones(len(self.df)) * self.sigy
            if np.isscalar(self.sigx):
                self.sigx = np.ones(len(self.df)) * self.sigx
            y = self.df.logdWn.values
```

```
            sigx = self.sigx
            sigy = self.sigy
            for i, x in enumerate(self.df.N160cs.values):
                if len(self.xgrid.shape) > 2:
                    xgrid = self.xgrid[i]
                    ygrid = self.ygrid[i]
                else:
                    xgrid = self.xgrid
                    ygrid = self.ygrid
                #self.fxfy[i] = self.calcfxfy(x, y[i], sigx[i], sigy[i],
                       #xgrid=xgrid, ygrid=ygrid)
                fxfy = self.calcfxfy(x, y[i], sigx[i], sigy[i],
                       xgrid=xgrid, ygrid=ygrid)
                if i == 0:
                    self.fxfy = np.zeros((self.sigx.shape[0], xgrid.shape[0],
                                  xgrid.shape[1]))
                Vol = self.calcVol(fxfy, xgrid=xgrid, ygrid=ygrid,
                       dx=self.dx[i], dy=self.dy[i])
                #Vol = 1.
                try:
                    self.fxfy[i] = fxfy / Vol
                except ValueError:
                    pdb.set_trace()
                # This is one of my diagnostics: check if the vol == 1
                #print('Vol: {}'.format(Vol))
                #pdb.set_trace()


    def __call__(self, beta, kwargs, df, wt=1., **otherkwargs):
        '''
        This is meant to mimic the other likelihood functions,
        and is supplied to the minimize function.
        By default it returns the negative sum of probabilities for a
        given set of regression coefficients.
        '''
        g = kwargs.get('g', g_2)
        EPS = g(beta, {'a': self.xgrid,
                       'b' : self.ygrid})
        sigy = self.sigy
        sigx = self.sigx
        x = kwargs['a']
        y = kwargs['b']
        if self.isdf:
            liq = self.liq
        else:
            if np.isscalar(sigy):
                sigy = np.ones(len(df)) * self.sigy
            if np.isscalar(sigx):
                sigx = np.ones(len(df)) * self.sigx
            Liq = df['Liq'].values
            liq = np.ones(x.shape)
            liq[Liq == 'Yes'] = -1.
        out = np.zeros(x.shape)
        #plot = False
        plot = otherkwargs.get('plot', False)
        for i,xmean in enumerate(x):
            if self.verbose:
```

```
                if i %10 == 0:
                    sys.stdout.write('-')
                if i % 50 == 0:
                    sys.stdout.write('{}'.format(i))
                sys.stdout.flush()
            if len(EPS.shape) > 2:
                eps = EPS[i]
                xgrid = self.xgrid[i]
                dx = self.dx[i]
                ygrid = self.ygrid[i]
                dy = self.dy[i]
            else:
                eps = EPS
                xgrid = self.xgrid
                dx = self.dx
                ygrid = self.ygrid
                dy = self.dy
            if self.speedup:
                fxfy = self.fxfy[i]
            else:
                fxfy = self.calcfxfy(xmean, y[i], sigx[i], sigy[i],
                        xgrid=xgrid, ygrid=ygrid)
                Vol = self.calcVol(fxfy, xgrid=xgrid, ygrid=ygrid,
                        dx=dx, dy=dy)
            if otherkwargs.get('ReturnProb', False):
                probsurf = norm.cdf(-eps / beta[-1]) * fxfy
            else:
                probsurf = norm.cdf(liq[i] * eps/beta[-1]) * fxfy

            if i == 0:
                probsurfall = probsurf
            else:
                probsurfall += probsurf

            out[i] = self.calcVol(probsurf, xgrid=xgrid, ygrid=ygrid,
                    dx=dx, dy=dy)
        if otherkwargs.get('ReturnProb', False):
            return out
        if plot:
            fig = plt.figure()
            ax = fig.add_subplot(111, projection='3d')
            ax.plot_surface(self.xgrid, self.ygrid, probsurfall)
            plt.show()
            pdb.set_trace()
            return fig
        return -np.sum(np.log(np.maximum(out, 1e-10)) * wt)


    def calcVol(self, surf, **kwargs):
        '''
        Calculates the volume under a two-dimensional surface.
        Surf is the n x m array of surface values.

        Optional keyword arguments allow you to specify the x and y grid.
        Otherwise it defaults to the single grid that was created in the
        __init__ function. However, if self.altgrid == True, that won't work.

```

```
        Three options for self.voltype: 'bvs', 'quickndirty', and 'other'.
        See below for details.
        '''
        self.voltype = kwargs.get('voltype', self.voltype)
        xgrid = kwargs.get('xgrid', self.xgrid)
        ygrid = kwargs.get('ygrid', self.ygrid)
        dx = kwargs.get('dx', self.dx)
        dy = kwargs.get('dy', self.dy)
        x_range = kwargs.get('x_range', self.x_range)
        y_range = kwargs.get('y_range', self.y_range)
        if self.voltype == 'bvs':
            bvs = RectBivariateSpline(xgrid[:,0],
                                      ygrid[0,:],
                                      surf)
            #out = bvs.integral(x_range[0], x_range[1],
                                      #y_range[0], y_range[1])
            out = bvs.integral(np.min(xgrid), np.max(xgrid),
                               np.min(ygrid), np.max(ygrid))
        elif self.voltype =='quickndirty':
            out = np.sum(surf[:-1, :-1]) * dx * dy

        elif self.voltype == 'other':
            Ny = len(ygrid[0,:])
            interm = np.zeros(Ny)
            for i in range(Ny):
                try:
                    interm[i] = np.trapz(surf[:,i], xgrid[:,0])
                except IndexError:
                    pdb.set_trace()
            out = np.trapz(interm, ygrid[0,:])
        return out

    def calcfxfy(self, xmean, ymean, xsig, ysig, *args, **kwargs):
        '''
        This returns a two dimension array of the probability densify function
        in both the x and y direction.

        Notice that you need to specify the xgrid and ygrid
        if self.altgrid == True
        '''
        xgrid = kwargs.get('xgrid', self.xgrid)
        ygrid = kwargs.get('ygrid', self.ygrid)

        fx = norm.pdf(xgrid, loc=xmean, scale=xsig)
        fy = norm.pdf(ygrid, loc=ymean, scale=ysig)
        if np.min(xgrid) < 0:
            #pdb.set_trace()
            pass
        out = fx * fy
        return out

def MLEall(beta, kwargs, df, gsign, likefunc, method='Powell', wt=1, wt2=1,
           ):
    '''
    This is a small helper function to run the minimization of likelihood,
    and send the results to the compareResults function for summary
    '''
```

```
    results = minimize(likefunc, beta, args=(kwargs,df, wt),
                                method=method,
                                options={'maxiter': 5000, 'disp': True})
    print(results.x)
    print(results.x/np.abs(results.x[1]))
    comres = compareResults(g1MLEalt,[results.x,kwargs], gsign)
    print(comres)
    return (results, comres)

def MLEmeta(mydict, plot=True):
    '''
    This is an attempt to make the minimization more straightforward.
    It is still pretty convoluted with dictionaries within dictionaries,
    but it works for now.
    '''
    def countargs(kwargs):
        count = 0
        for key in kwargs:
            if key in 'abcdefhijk':
                count += 1
        return count + 2

    kwargs = mydict['kwargs']
    beta = mydict.get('beta', [1. for i in range(countargs(kwargs))])
    UseProb = mydict.get('UseProb', True)
    print(beta)
    #method=mydict.get('method','Powell')
    method=mydict.get('method','TNC')
    bounds = mydict.get('bounds',((None, None), (None, None), (0,None)))
    #bounds = ((None, None), (-1, -1), (None, None), (0,None))
    CH = mydict.get('CaseHist')
    df = CH.df
    wt = CH.altwt
    g = kwargs.get('g', g_1)
    likefunc = mydict.get('likefunc', log_like)
    gsign = CH.gsign
    results = minimize(likefunc, beta, args=(kwargs,df, wt),
                           method=method,
                           #bounds=bounds,
                           options={'maxiter': 10000, 'disp': True})
    print(results.x)
    comres = compareResults(g,[results.x,kwargs], gsign, UseProb=UseProb)

    if plot:
        plotfunc = kwargs.get('plotfunc', plotFit2)
        plotfunc(results.x,kwargs, df, **mydict['plotkwargs'])
    print(comres)
    return (results, comres)

def run5(**kwargs):
    method = 'TNC'
    CS1 = CaseHists(Degrad='DS', useVs=False, GDtype='direct', altwt=1.0,
                      myunitwt='assumed', rd='Lasley',
                      #altdWn=False, #altCycles=False, #False is default for both
                      calcN160=True,# True is Default
                      #Ncorrtype='Lasley',
                      #N_Gmaxtype='SeedetalALT'
```

```
                        )
    #pdb.set_trace()
    CS1.to_pickle()
    #CSW = CaseHists(Degrad='DS', useVs=False, GDtype='direct', altwt=1.0,
                    #myunitwt='assumed', rd='Lasley', altdWn=False,
                    #altCycles=False, N_Gmaxtype='Wairetal')
    CSVs = CaseHists(Degrad='DS', useVs=True, GDtype='direct', altwt=0.25,
                    myunitwt='assumed', rd='Lasley',
                    #N_Gmaxtype='SeedetalALT'
                    )
    CSVs.to_pickle()
    plotBase(CS1.df)
    plotBase(CSVs.df, figname='Vs')
    df = CS1.df
    gsign = CS1.gsign

    #Rtype = 'Rrup (km)'
    #Jafdf = df[~np.isnan(df[Rtype].values)].copy()
    #Jaf = LiqEvalJafarianetal(Jafdf, Rtype=Rtype)
    #JafResults = compareResults(Jaf.calc_g, [0.5],
           #gsign[~np.isnan(df[Rtype].values)], sig=1)
    #print('Jafarian et al. 2014')
    #print(JafResults)
    #Jaf.plot()
    #plt.show()

    #pdb.set_trace()

    beta = kwargs.get('beta', [0.4,-10.,1.])
    dict1 = dict(CaseHist=CS1,
           beta=beta,
           method=method,
           bounds = ((None, None), (None, None), (0,None)),
           kwargs = {'a': CS1.df.N160cs.values,
               'b' : np.log(CS1.df.dWn.values),
              'g' : g_2},

           plotkwargs ={
               'logDW' : True,
               'norm' : True,
               'label' : 'DS-Regular',
               'invalues' : {'a' : np.linspace(0, 28, 60)}})
    result1 = MLEmeta(dict1)
    print('Regular^')

    dictVs = dict(CaseHist=CSVs,
           beta=[0.4,-10.,1.],
           method=method,
           bounds = ((None, None), (None, None), (0,None)),
           kwargs = {'a': CSVs.df.N160cs.values,
               'b' : np.log(CSVs.df.dWn.values),
              'g' : g_2},

           plotkwargs ={
               'logDW' : True,
               'norm' : True,
               'label' : 'Vs',
```

```
            'figname' :'Vs',
            'invalues' : {'a' : np.linspace(0, 28, 60)}})
resultVs = MLEmeta(dictVs)
print('With Vs and altwt = 0.25^')
print('\n\nIdriss and Boulanger')
IBResults = compareResults(gIdrissBoul2, [df.N160cs.values,
    df['CSRm7.5'].values], gsign, sig=0.13, UseProb=True) #Idriss and Boulanger 2010,
        pg. 100 (105)
print(IBResults)
print('\n\nYoud et al.')
YoudResults = compareResults(gYoudEtal, [df], gsign)
print(YoudResults)


print('\n\nCetin')
CetResults = compareResults(gCetindf, [df], gsign, sig=2.7, UseProb=True)
print(CetResults)



like2 = OtherLikelihood(df=df)

dict2 = dict(CaseHist=CS1,
       likefunc = like2,
       beta = result1[0].x,
       kwargs = {'a': CS1.df.N160cs.values,
           'b' : np.log(CS1.df.dWn.values),
          'g' : g_2},
       method=method,
       plotkwargs ={
           'logDW' : True,
           'norm' : True,
           'label' : 'No Vs, Uncert',
           'invalues' : {'a' : np.linspace(0, 28, 60)}})
result2 = MLEmeta(dict2)

likeVs = OtherLikelihood(df=CSVs.df)

dict2Vs = dict(CaseHist=CSVs,
       likefunc = likeVs,
       beta = resultVs[0].x,
       method=method,
       kwargs = {'a': CSVs.df.N160cs.values,
           'b' : np.log(CSVs.df.dWn.values),
          'g' : g_2},
       plotkwargs ={
           'logDW' : True,
           'norm' : True,
           'figname' : 'Vs',
           'label' : 'Vs \& Uncert',
           'invalues' : {'a' : np.linspace(0, 28, 60)}})
result2Vs = MLEmeta(dict2Vs)
#labels=['$G_{max} from $N_{1,60cs}$',
    #'$G_{max} from $N_{1,60cs} & measured $V_{s,\\text{ measured}}$',
    #'$G_{max} from $N_{1,60cs}$, with Uncertainties',
    #'$G_{max} from $N_{1,60cs} & measured $V_{s,\\text{ measured}}$, with
        Uncertainties',
```

```
    labels = [ '#1 - $G_{max} = f(N_{1,60cs})$',
               '#2 - $G_{max} = f(V_{s,\\text{ meas.}}, N_{1,60cs})$',
               '#3 - $G_{max} = f(N_{1,60cs})$, with Uncertainties',
               '#4 - $G_{max} = f(V_{s,\\text{ meas.}}, N_{1,60cs})$, with Uncert.'
                ]

    mixedbeta = np.copy(result2[0].x)
    mixedbeta[-1] = result1[0].x[-1]
    mixedkwargs = {'a': CS1.df.N160cs.values,
           'b' : np.log(CS1.df.dWn.values),
          'g' : g_2}
    mixed = compareResults(g_2, [mixedbeta, mixedkwargs], gsign, UseProb=True)
    print(mixedbeta)
    print(mixed)



    printCoefTable(result1[0].x, resultVs[0].x,
                   result2[0].x, result2Vs[0].x,
                   labels=labels
                )

    printCompareTable(result1[1], resultVs[1], result2[1], result2Vs[1],
           mixed,
           #resultW[1], resultW2[1],
           IBResults, CetResults,
           #JafResults,
           labels=labels + [
           'Mixed coefficients',
           'Boulanger and Idriss 2012',
           'Cetin et al. 2004',
           #'Jafarian et al. 2014'
           ])

    plt.legend(loc='lower right')

    plt.show()

    pdb.set_trace()


def run4():
    method = 'Nelder-Mead'
    g=g_2
    CS1 = CaseHistArtificial2(N=200, thetas=[0.4, -14, 1.2], g=g)
    #pdb.set_trace()
    plotBase(CS1.df)
    print(CS1.thetas)
    df = CS1.df
    gsign = CS1.gsign
    dict1 = dict(CaseHist=CS1,
           beta=[1.,-10.,1.],
           method=method,
           kwargs = {'a': CS1.df.N160cs.values,
               'b' : np.log(CS1.df.dWn.values),
              'g' : g},
           plotkwargs ={
```

```
            'logDW' : True,
            'norm' : True,
            'label' : 'Deterministic-Old',
            'invalues' : {'a' : np.linspace(0, 35, 60)}})
result1 = MLEmeta(dict1)

like = OtherLikelihood(df=df)
#outval = like([0.7, -1, 15, 3.], dict1['kwargs'], df)

dict2 = dict(CaseHist=CS1,
       likefunc = like,
       beta = result1[0].x,
       method=method,
       kwargs = {'a': CS1.df.N160cs.values,
           'b' : np.log(CS1.df.dWn.values),
          'g' : g},
       plotkwargs ={
           'logDW' : True,
           'norm' : True,
           'label' : 'Probabilistic-Old',
           'invalues' : {'a' : np.linspace(0, 35, 60)}})
result2 = MLEmeta(dict2)

plt.figure('limit-state')
plt.legend(loc='lower right')
plt.savefig('Limit-state-old.pdf')
plt.close('all')

# Alt method
CS1.DetLiqAlt()
#CS1a = CaseHistArtificial2(N=1000, thetas=[0.4, -15, 1.2], g=g,
       #DetLiqAlt=True)
CS1a = CS1
plotBase(CS1a.df)#,figname='Limit-State-New' )
gsign = CS1a.gsign
dict1a = dict(CaseHist=CS1a,
       beta=[1.,-10.,1.],
       method=method,
       kwargs = {'a': CS1a.df.N160cs.values,
           'b' : np.log(CS1a.df.dWn.values),
          'g' : g},
       plotkwargs ={
           'logDW' : True,
           'norm' : True,
           #'figname' : 'Limit-State-New',
           'label' : 'Deterministic-New',
           'invalues' : {'a' : np.linspace(0, 35, 60)}})
result1a = MLEmeta(dict1a)
#plotFit2(results1a.x,{'a': CS1a.df.N160cs.values, 'b' : np.log(CS1a.df.dWn.values),
     'g' : g}, df, figname='Limit-State-New')

likea = OtherLikelihood(df=df)
#pdb.set_trace()
#outval = like([0.7, -1, 15, 3.], dict1['kwargs'], df)

dict2a = dict(CaseHist=CS1a,
       likefunc = likea,
```

```
            beta = result1a[0].x,
            method=method,
            kwargs = {'a': CS1a.df.N160cs.values,
                'b' : np.log(CS1a.df.dWn.values),
               'g' : g},
            plotkwargs ={
                'logDW' : True,
                'norm' : True,
                #'figname' : 'Limit-State-New',
                'label' : 'Probabilitistic-New',
                'invalues' : {'a' : np.linspace(0, 35, 60)}})
    result2a = MLEmeta(dict2a)

    printCompareTable(result1[1], result2[1],
            result1a[1],
            result2a[1],
            labels=['Det.- Old', 'Prob. - Old',
                'Det. - New',
                'Prob. - New'])

    printCoefTable(result1[0].x, result2[0].x,
                   result1a[0].x,
                   result2a[0].x,
            labels=['Det.- Old', 'Prob. - Old',
                'Det. - New',
                'Prob. - New'])
    #plt.figure('Limit-State-New')
    plt.legend(loc='lower right')
    plt.savefig('Limit-state-new.pdf')

    plt.show()
    pdb.set_trace()

def LiqEvalEnergy(df, **kwargs):
    '''
    This function will calculate the probability of liquefaction
    '''
    g = kwargs.get('g', g_2)
    gkwargs = kwargs.get('gkwargs', {'a': df.N160cs.values,
                'b' : np.log(df.dWn.values)})
    gbeta = kwargs.get('gbeta',
           #[0.2897, -1., -9.3443, 0.9188]
           #[  0.91626033, -4.15771179, -45.22387576, 4.1210432 ]
           #[0.4754, -13.471, 6.277e-4] #1 May 15
           [0.2912, -11.18, 1.272]
           )
    gout = g(gbeta, gkwargs)
    Pliq = norm.cdf(-gout / gbeta[-1])

    LSC = calcLimStCurves(0.15, gbeta, gkwargs)
    #FSscale = np.minimum(np.min(np.log(df.dWn.values)),
           #kwargs.get('FSscale', gbeta[1]))
    FSscale = kwargs.get('FSscale', gbeta[1])
    if kwargs.get('FSalt', False):
           FS = np.minimum(2.,
               np.maximum(np.exp(LSC) / df.dWn.values, 0.1))
    else:
```

```
        FS = np.minimum(2.,
        np.maximum((LSC - FSscale) / (np.log(df.dWn.values) - FSscale), 0.1))
    FS[df.gamma.values * 1.543 < 1e-4] = 2. # Won't liq. if doesn't exceed threshold. The
         1.534 converts from the average strain to a max value.
    FS[np.log(df.dWn.values) < FSscale] = 2.
    #otherL = calcLimStCurves(0.15, gbeta,
           #{'a':np.linspace(0,30, 100)})
    #plt.figure('mine')
    #plt.semilogy(np.linspace(0, 30, 100), np.exp(otherL), '-k')
    #plt.plot(df.N160cs.values[Pliq>=0.15], df.dWn.values[Pliq>=0.15], 'ro')
    #plt.plot(df.N160cs.values[Pliq<0.15], df.dWn.values[Pliq<0.15], 'bo')
    #plt.plot(df.N160cs.values[FS>=1], df.dWn.values[FS>=1], 'gx')
    #plt.plot(df.N160cs.values[FS<1], df.dWn.values[FS<1], 'rx')
    #print('LS-log(dWn) values')
    #print(LSC)
    #print('Demand dWn vals (log of)')
    #print(np.log(df.dWn.values))
    ##f_ls = interp1d(np.linspace(0, 30,100), np.exp(otherL))
    ##print(np.log(f_ls(df.N160cs.values)))
    #plt.figure('FS')
    #plt.plot(FS, df.z.values)
    #plt.show()
    #pdb.set_trace()

    if kwargs.get('plot', False):
        indLiq = np.where(gout<= 0)
        indNo = np.where(gout> 0)
        print(len(indLiq[0]))
        print(len(indNo[0]))

        plt.semilogy(N160cs[indLiq], dWn[indLiq], 'ko')
        plt.plot(N160cs[indNo], dWn[indNo], 'ws')
        plt.show()

        pdb.set_trace()
    return Pliq, gout, FS

def LiqEvalBoulangerIdriss(df, **kwargs):
    returnall = kwargs.get('returnall', False)
    calcN160 = kwargs.get('calcN160', True)
    Mw = df.Mw.values
    N160cs = df.N160cs.values
    z = df.z.values
    PGA = df.amax.values
    sigveff = df.sigeffv.values
    sigv = df.sigv.values
    MSF = np.minimum( 6.9 * np.exp(-Mw / 4.) - 0.058, 1.8)
    if calcN160:
        N160cs, DN160, Cr, Cn, Cs = calcN160cs(df.Nm.values, df.sigeffv.values,
               df.FC.values, df.z.values, Ce=df.CE.values, Cs=df.CS.values,
               returnall=True)
        N160 = N160cs - DN160
    else:
        N160 = df.N160.values
    if 'Ksig' in kwargs.keys():
        Ksig = kwargs['Ksig']
    else:
```

```
        Csig = np.minimum(1 / (18.9 - 2.55 * np.sqrt(np.minimum(37,N160))), 0.3)
        Ksig = np.minimum(1 - Csig * np.log(sigveff / 101.325), 1.1)


    # Stress reduction coefficient. Notice that I am cheating here with
    # the depth; it says not to use below 34 meters (or even 20 m)
    alpha = -1.012 - 1.126 * np.sin(np.minimum(z, 34.) / 11.73 + 5.133)
    beta = 0.106 + 0.118 * np.sin(np.minimum(z, 34.) / 11.28 + 5.142)
    rd = np.exp(alpha + beta * Mw)

    #CSR M=7.5, sigveff = 1 atm
    CSR = 0.65 * sigv * PGA * rd / (sigveff * MSF * Ksig)
    co = kwargs.get('co', 2.67)
    siglnR = kwargs.get('siglnR', 0.13)
    CRR = calcLS_IdrissBoul2(0.15, N160cs,#Boulanger and Idriss 2012
           co=co, siglnR=siglnR) #Deterministic curve at Pl=15%???

    gout = gIdrissBoul2(N160cs, CSR, co=co)
    FS = np.minimum(2, CRR / CSR)
    Pliq = norm.cdf(-gout / siglnR)

    if kwargs.get('plot', False):
        indLiq = np.where(gout<= 0)
        indNo = np.where(gout> 0)
        print(len(indLiq[0]))
        print(len(indNo[0]))

        plt.plot(N160cs[indLiq], CSR[indLiq], 'ko')
        plt.plot(N160cs[indNo], CSR[indNo], 'ws')
        plt.show()

        pdb.set_trace()
    if returnall:
        return Pliq, gout, FS, CSR*Ksig, CRR*Ksig, rd, N160cs
    return Pliq, gout, FS

def rd_Cetin1(amax, Mw, z, **kwargs):
    beta = [-9.147, -4.173, 0.652, 10.567, 0.089,
              0.089, -7.76, 78.576, 0.0014, 19.8, 0.1, 0.0072]
    d = z
    g = np.empty(d.shape, dtype=float)
    numerator1 = ( 1 + (beta[0] + beta[1] * amax + beta[2] * Mw) /
                (beta[3] + beta[4] * np.exp(beta[5] *
                    (-d * 3.28 + beta[6] * amax + beta[7]))))
    numerator2 = ( 1 + (beta[0] + beta[1] * amax + beta[2] * Mw) /
                (beta[3] + beta[4] * np.exp(beta[5] *
                    (-d * 3.28 + beta[6] * amax + beta[7]))))
    denominator = ( 1 + (beta[0] + beta[1] * amax + beta[2] * Mw) /
                (beta[3] + beta[4] * np.exp(beta[5] * (beta[6] * amax + beta[7]))))
    g[d<beta[9]] = (numerator1 / denominator )[d<beta[9]]
    g[d>=beta[9]] = (numerator2 / denominator - beta[8] * (d - beta[9]))[d>=beta[9]]
    return g

def rd_Cetin2(amax, Mw, z, Vs12):
    d=z
    beta = [-23.013, -2.949, 0.999, 0.0525, 16.258,
              0.201, 0.341, 0.0785, 7.586, 0.0046,
```

```
2142                20,  0.85, 0.0072]
2143     g = np.empty(d.shape, dtype=float)
2144     numerator1 = (1 + (beta[0] + beta[1] * amax + beta[2] * Mw + beta[3] * Vs12) /
2145                   (beta[4] + beta[5] * np.exp(beta[6] *(-d + beta[7] * Vs12 + beta[8]))))
2146     numerator2 = (1 + (beta[0] + beta[1] * amax + beta[2] * Mw + beta[3] * Vs12) /
2147                   (beta[4] + beta[5] * np.exp(beta[6] * (-beta[10] + beta[7] * Vs12 +
                          beta[8]))))
2148     denominator = (1 + (beta[0] + beta[1] * amax + beta[2] * Mw + beta[3] * Vs12) /
2149                   (beta[4] + beta[5] * np.exp(beta[6] * (beta[7] * Vs12 + beta[8]))))
2150     g[d<beta[10]] = (numerator1 / denominator )[d<beta[10]]
2151     g[d>=beta[10]] = (numerator2 / denominator - beta[9] * (d - beta[10]))[d>=beta[10]]
2152     return g
2153
2154 def LiqEvalCetin(df, **kwargs):
2155     '''
2156     Evaluates liquefaction using Cetin's procedure
2157     '''
2158     returnall = kwargs.get('returnall', False)
2159     calcN160 = kwargs.get('calcN160', True)
2160
2161     def calcDWF(Mw):
2162         #CetinM = [5.50,6.00,6.50,7.00,7.51,8.00,8.50]
2163         #CetinMSF = [1.61,1.41,1.25,1.12,1.00,0.91,0.83]
2164         #f_MSF = interp1d(CetinM, CetinMSF)
2165         #MSF = f_MSF(Mw)
2166         MSF = (7.5 / Mw) ** 2.217 #From Cetin's Spreadsheet SIteX_KOC.xlsx
2167         return MSF
2168
2169     def calcKsig(sigveff, N160cs):
2170         #f = 0.6 + 0.005714 * (40 - (np.maximum(5, np.minimum(40, N160cs))))
2171         #ksig = np.minimum(1.5, (sigveff/101.325) ** (f-1))
2172         ksig = (100. / sigveff) ** 0.278 #From Cetin's Spreadsheet SIteX_KOC.xlsx
2173         return ksig
2174
2175     FC = df.FC.values
2176     z = df.z.values
2177     amax = df.amax.values
2178     Mw = df.Mw.values
2179     if 'Vs12' in df.columns:
2180         rd = rd_Cetin2(amax, Mw, z, df.Vs12.values)
2181     else:
2182         rd = rd_Cetin1(amax, Mw, z)
2183     sigv = df.sigv.values
2184     sigeffv = df.sigeffv.values
2185
2186     # N Correction
2187     N = df.Nm.values
2188     if calcN160:
2189         N160, Cr, Cn, Cs = calcN160cs(N, sigeffv, FC, z,
2190                    Ce=df.CE.values, #Cs=df.CS.values,
2191                    Cb=df.CB.values,
2192                    Cetin=True,
2193                    calcCs=False,
2194                    returnall=True)
2195     N160 = df.N160.values
2196
2197     FC_lim = np.minimum(35, FC)
```

```
    FC_lim[FC<5] = 0.
    C_fines = (1 + 0.004 * FC_lim) + 0.05 * (FC_lim / N160)
    N160cs = N160 * C_fines
    CSR_eq = 0.65 * amax * sigv * rd / sigeffv
    DWF = calcDWF(Mw)
    Ksig = calcKsig(sigeffv, N160)
    CSR_eq_75 = CSR_eq / DWF
    CSR_eq_str = CSR_eq_75 / Ksig

    gout = gCetin(CSR_eq, N160, Mw, FC_lim, sigeffv)
    Pliq = norm.cdf(-gout / 2.7)
    CRR = calcLS_Cetin(0.15, N160, FC_lim, Mw, sigeffv) # Uses 0.5 for deterministic? See
        SiteX_KOC.xlsx

    FS = np.minimum(2., CRR / CSR_eq) # From SiteX_KOC.xlsx
    #FS = np.minimum(2., CRR / CSR_eq_str) #Idriss and Boul 2008 use this max val
    if returnall:
        return Pliq, gout, FS, CSR_eq, CRR, rd, N160
    return Pliq, gout, FS

class LiqEvalJafarianetal(object):
    '''
    Jafarian, Y., Vakili, R., Abdollahi, A. S., and Baziar, M. H. (2014).
    "Simplified Soil Liquefaction Assessment Based on Cumulative Kinetic
    Energy Density: Attenuation Law and Probabilistic Analysis."
    International Journal of Geomechanics, 14(2), 267-281.
    '''
    def __init__(self, df, **kwargs):
        self.df = df
        if 'rhoeff' in kwargs.keys():
            self.df['rhoeff'] = kwargs['rhoeff']
        elif 'rhoeff' in df.columns:
            pass
        else:
            self.df['rhoeff'] = 1400. #kg/m^3, This is a guess, maybe up to 1800

        self.Rtype = kwargs.get('Rtype', 'Rrup (km)')
        self.df['rsed'] = self.calc_rSED(**kwargs)
        self.df['SEDsurf'] = self.calc_SED_surf()
        self.df['CKED'] = self.calc_CKED()
        self.df['g'] = self.calc_g()

    def calc_CKED(self, **kwargs):
        rhoeff = self.df.rhoeff.values
        SEDsurf = self.df.SEDsurf.values
        rsed = self.df.rsed.values
        return rhoeff * rsed * SEDsurf * 0.0001# in N*s/M^2 or kg / (s *m)

    def calc_SED_surf(self, **kwargs):
        M = self.df.Mw.values
        R = self.df[self.Rtype].values
        if 'Vs30' in self.df.columns:
            Vs30 = self.df.Vs30
        elif 'Vs30' in kwargs.keys():
            Vs30 = kwargs['Vs30']
        else:
            Vs30 = 273.* np.ones(self.df.shape[0])
```

```
        if 'faulttype' in self.df.columns:
            faulttype = self.df.faulttype
        elif 'faulttype' in kwargs.keys():
            faulttype = kwargs['faulttype']
        else:
            faulttype = ['unspecified' for val in
                xrange(self.df.shape[0])]

        f1 = (76.37e-7 * (R + 24.83) ** 2 * (1 + 0.026348 * M *
                (0.040161 * R + np.log(Vs30) + 1.45626)) +
                1.17971 * M - 0.01443 * R - 0.82029 * np.log(Vs30)
                - 0.39071)
        f2 = np.zeros(self.df.shape[0])

        f2[faulttype == 'normal'] = (97.3e-5 * M[faulttype == 'normal'] ** 2
                + 0.10408 * M[faulttype == 'normal'] - 0.89195)
        f2[faulttype == 'strikeslip'] = ((0.12595 *
            M[faulttype == 'strikeslip'] - 0.64) * (-0.14572 *
                M[faulttype == 'strikeslip'] ** 2 +
                M[faulttype == 'strikeslip'] + 0.91072))
        f2[faulttype == 'reverse'] = (-5.857e-6 *
                M[faulttype == 'reverse'] ** 3 *
                (M[faulttype == 'reverse'] - 3.4631) *
                (M[faulttype == 'reverse'] ** 3 - 8.56458 *
                    M[faulttype == 'reverse'] ** 2 + 8.19733 *
                    M[faulttype == 'reverse'] + 5.93354))

        fb = -0.0024 * R + 0.0005 * Vs30 + 0.0054
        fb[faulttype == 'normal'] = (-0.125 * M[faulttype == 'normal'] -
                0.0054 * R[faulttype == 'normal'] +
                0.0002 * Vs30[faulttype == 'normal'] + 0.9772)
        fb[faulttype == 'strikeslip'] = (0.0003 *
                              Vs30[faulttype == 'strikeslip'] - 0.1067)
        fb[faulttype == 'reverse'] = (0.0934 * M[faulttype == 'reverse'] -
                0.0043 * R[faulttype == 'reverse'] + 0.0006 *
                Vs30[faulttype == 'reverse'] - 0.6259)
        SEDsurf = 10 ** (f1 + f2 + fb) # in cm^2 /s?
        return SEDsurf


    def calc_rSED(self, **kwargs):
        z = self.df.z.values
        M = self.df.Mw.values
        rsed =( 1 - ((-0.2361663 * M + 2.263352) * z ** 1.958192) /
                ((-0.2361663 * M + 2.263352) * z ** 1.958192 +
                    (0.4487404 * M + 1.959488) * z ** 1.316766 +
                    (14.39358 * M - 56.22128)))
        #self.df.rsed = rsed
        return rsed

    def calc_g(self, Pl=0.5, sig=1.):
        sigeffv = self.df.sigeffv.values
        N160 = self.df.N160.values
        FC = self.df.FC.values
        CKED = self.df.CKED.values
        # not sure, but may need to take negative of g
        g = (np.log(1/Pl - 1) - 0.80908 * np.log(sigeffv / 101.325) -
```

```
2311                0.38610 * N160 - 0.0236 * FC - 0.16946 +
2312                1.40241 + np.log(CKED))
2313        self.df.g = -g
2314        return -g
2315
2316    def calc_CKED_liq(self, N160, **kwargs):
2317        Pl = kwargs.get('Pl', 0.5)
2318        FC = kwargs.get('FC', 0.)
2319        sigeffv = kwargs.get('sigeffv', 50.)
2320        return np.exp((np.log(1./Pl - 1) - 0.80908 * np.log(sigeffv / 101.325) -
2321                0.38610 * N160 - 0.0236 * FC - 0.16946) / -1.40241)
2322
2323    def plot(self, **kwargs):
2324        fig, ax = plt.subplots(num=kwargs.get('figname', 'Jafarian'))
2325        ax.plot(self.df.N160[self.df.Liq =='Yes'],
2326                self.df.CKED[self.df.Liq =='Yes'], 'ok')
2327        ax.plot(self.df.N160[self.df.Liq =='No'],
2328                self.df.CKED[self.df.Liq =='No'], 'sw')
2329        N160 = np.linspace(2, 40, 50)
2330        CKED = self.calc_CKED_liq(N160)
2331        ax.plot(N160, CKED, 'b', lw=3)
2332        ax.set_xlabel('$N_{1,60}$')
2333        ax.set_ylabel('CKED ($N s/m^2$)')
2334        ax.set_yscale('log')
2335        return fig
2336
2337 def calcKsigFactory(**kwargs):
2338     '''
2339     Calcs Ksigs and compares
2340     '''
2341
2342     def calcdWone(sigveff, Degrad, PI, tau, N160, **kwargs):
2343         '''
2344         Calculates the dissipated energy in one equivalent cycle
2345         '''
2346         soil = kwargs.get('soil', 1)
2347         phi = np.sqrt(20 * N160) + 20
2348         ko = 1. - np.sin(phi * np.pi / 180.)
2349         sigme = (1 + 2 * ko) * sigveff / 3.
2350         Gmax = 440. * N160 ** (1./3.) * 101.325 * (sigme/ 101.325) ** 0.5
2351         gam, Gratio, damping = getDegrad('DS', PI, sigme, N=10, soil=soil)
2352         G, D, gamma = TokimatsuSeedDirect(gam, Gratio, damping, Gmax, tau)
2353         dw1 = 2 * np.pi * D * tau ** 2 / G
2354         return dw1, G, D, gamma
2355
2356
2357     def calcKsigEn(sigveff, dW1, tau1, **kwargs):
2358         '''
2359         Calculates Ksig using the Darendeli and Stokoe curves
2360         and energy principles
2361         '''
2362         verbose = kwargs.get('verbose', False)
2363         sigveff1 = kwargs.get('sigveff1', 101.325)
2364         Degrad = kwargs.get('Degrad', 'DS')
2365         PI = kwargs.get('PI', [0.])
2366         soil = kwargs.get('soil', 0)
2367         normd = kwargs.get('normd', True)
```

```
        dW1n = dW1 / sigveff1
        numtrials = kwargs.get('numtrials', 200)
        #pdb.set_trace()

        for i, sigve in enumerate(sigveff):
            #print(i)
            tau = CSR * sigve
            dw, G, D, gamma = calcdWone(sigve, Degrad, PI, tau, N160,
                   soil=soil)
            dwn = dw / sigve
            mytau = tau


            trialtaus = kwargs.get('trialtaus', np.linspace(0.35 * tau,
                tau1*np.maximum(3, 1.7 * sigve/101.325), numtrials))
            theseerrors = np.zeros(trialtaus.shape)
            theseDWs = np.zeros(trialtaus.shape)

            count = -1
            if verbose:
                print('\n\n\n Sigveff: {}'.format(sigve))
            if normd:
                error = np.abs(dW1n - dwn)/ dW1n
            else:
                error = np.abs(dW1 - dw)/ dW1
            while np.abs(error) > 0.01:
                count += 1

                if error < -10:
                    break

                if count > (numtrials-1):
                    dw = np.nan
                    mytau = np.nan
                    break
                if normd:
                    #mytau = np.sqrt(dW1n * G * sigve /(2 * np.pi * D))
                    mytau = trialtaus[count]
                else:
                    #mytau = np.sqrt(dW1 * G /(2 * np.pi * D))
                    mytau = trialtaus[count]
                dw, G, D, gamma = calcdWone(sigve, Degrad, PI, mytau, N160,
                       soil=soil)
                if normd:
                    dwn = dw / sigve
                    #error = np.abs(dW1n - dwn)/ dW1n
                    theseerrors[count] = (dW1n - dwn)/ dW1n
                    theseDWs[count] = dwn
                    error = theseerrors[count]
                else:
                    error = np.abs(dW1 - dw)/ dW1
                if verbose:
                    print('Count: {}, Error: {:.4f}, tau: {:.2f}'.format(
                        count, error, mytau))
            if np.abs(error) < 0.01:
                dWone[i] = dwn
                taus[i] = mytau
```

```
            else:
                f_tau = interp1d(theseerrors[:count-1], trialtaus[:count-1])
                f_dW = interp1d(theseerrors[:count-1], theseDWs[:count-1])
                try:
                    dWone[i] = f_dW(0.)
                    taus[i] = f_tau(0.)
                except ValueError:
                    pdb.set_trace()
            #errors[i] = error
            #dWone[i] = dw
            #taus[i] = mytau
        Ksig = (taus / sigveff) / (tau1 / sigveff1)
        return Ksig, taus, dWone

    #ko=0.5
    normd = True
    verbose = True
    PI = [0.]
    Drs = np.array([30., 40., 50.])#, 50., 70.]
    N160s = Drs ** 2 / 15 ** 2
    x = np.linspace(0, 50 ,100)
    IBCSR = np.exp(x / 14.1 + (x / 126) ** 2 -
           (x / 23.6) ** 3 + (x / 25.4) ** 4 - 2.67)
    f_csr = interp1d(x, IBCSR)
    CSRs = f_csr(N160s)

    lws = [1.5, 2.3, 3.5]
    colors = ['0.3', '0.1', '0.0']
    Degrad = 'DS'
    labels = [
           '$D_r = {:.0f}\%$, $CSR={:.2f}$:'.format(Drs[0], CSRs[0]),
           #'  Darendeli and Stokoe (2001)',
           '  Boulanger and Idriss (2004)',
           #'  Hynes and Olsen (1999)',
           #'',
           '$D_r = {:.0f}\%$, $CSR={:.2f}$:'.format(Drs[1], CSRs[1]),
           #'  Darendeli and Stokoe (2001)',
           '  Boulanger and Idriss (2004)',
           '  Hynes and Olsen (1999)',
           #'',
           '$D_r = {:.0f}\%$, $CSR={:.2f}$:'.format(Drs[2], CSRs[2]),
           #'  Darendeli and Stokoe (2001)',
           '  Boulanger and Idriss (2004)',
           '  Hynes and Olsen (1999)',
           #'',
           'Dissipated-Energy-Derived:',
           '  Darendeli and Stokoe (2001) - Clean Sand',
           '  Darendeli and Stokoe (2001) - Sand with High Fines',
           ]
    HOdashes = (15,5)

    LegLines = [
        plt.Line2D([], [], color='0.0', linestyle='-', lw=lws[0], alpha=0),
        #plt.Line2D([], [], color=colors[0], linestyle='-', lw=lws[0]),
        plt.Line2D([], [], color=colors[1], linestyle='--', lw=lws[0]),
        #plt.Line2D([], [], color=colors[2], linestyle='-', lw=lws[0], dashes=HOdashes),
        #plt.Line2D([], [], color='0.0', linestyle='-', lw=lws[0], alpha=0),
```

```
        plt.Line2D([], [], color='0.0', linestyle='-', lw=lws[0], alpha=0),
        #plt.Line2D([], [], color=colors[0], linestyle='-', lw=lws[1]),
        plt.Line2D([], [], color=colors[1], linestyle='--', lw=lws[1]),
        plt.Line2D([], [], color=colors[2], linestyle='-', lw=lws[1], dashes=HOdashes),
        #plt.Line2D([], [], color='0.0', linestyle='-', lw=lws[0], alpha=0),
        plt.Line2D([], [], color='0.0', linestyle='-', lw=lws[0], alpha=0),
        #plt.Line2D([], [], color=colors[0], linestyle='-', lw=lws[2]),
        plt.Line2D([], [], color=colors[1], linestyle='--', lw=lws[2]),
        plt.Line2D([], [], color=colors[2], linestyle='-', lw=lws[2], dashes=HOdashes),
        #plt.Line2D([], [], color='0.0', linestyle='-', lw=lws[0], alpha=0),
        plt.Line2D([], [], color='0.0', linestyle='-', lw=lws[0], alpha=0),
        plt.Line2D([], [], color=colors[0], linestyle='-', lw=lws[1]),
        plt.Line2D([], [], color=colors[0], linestyle='-', lw=lws[2]),

        ]

    sigveff1 = np.array([101.325])
    sigveff = np.linspace(40, 900, 45)#, 50, 200, 300, 400])
    errors = np.zeros(sigveff.shape)

    plt.figure('Main Event', figsize=(8.5,5.))

    for Dr, CSR, lw in zip(Drs, CSRs, lws):
        print('Dr = {}\n'.format(Dr))
        N160 = Dr ** 2 / 15. ** 2
        dWone = np.zeros(sigveff.shape)
        taus = np.zeros(sigveff.shape)
        soil = 0
        tau1 = CSR * 101.325
        dW1, G1, D1, gam1 = calcdWone(sigveff1, Degrad, PI, tau1, N160,
                soil=soil)
        dW1n = dW1 / sigveff1[0]


        Ksig, tau, dW = calcKsigEn(sigveff, dW1, tau1, PI=PI, soil=soil,
                Degrad='DS', verbose=verbose, normd=normd)
        #Ksig2 = (taus / sigveff) / (tau1 / sigveff1)
        #pdb.set_trace()

        Csig = np.minimum(1 / (18.9 - 17.3 * Dr/100.), 0.3)
        KsigIB = np.minimum(1 - Csig * np.log(sigveff / 101.325), 1.1)

        plt.figure('Main Event')
        #plt.plot(sigveff/101.325, Ksig, '-', lw=lw, color=colors[0],
                #label=labels[1])

        plt.plot(sigveff/101.325, KsigIB, '--', lw=lw, color=colors[1],
                label=labels[2])
        if Dr >= 40:
            f = np.minimum(0.8, np.maximum(0.6, 1 - 0.005 * Dr))
            plt.plot(sigveff/101.325, np.minimum(1.,
                    (sigveff / 101.325) ** (f - 1)), '-',
                   lw=lw, label=labels[3], color=colors[2], dashes=HOdashes)
        plt.figure('CSR')
        plt.plot(sigveff, taus/sigveff, '-', lw=2, label='$D_r = {}$'.format(Dr))

    plt.figure('Main Event')
```

```
    plt.grid('on', alpha=0.5)
    plt.xlabel('$\sigma\'_{v0} / P_a$')
    plt.ylabel('$K_{\sigma}$')
    leg1 = plt.legend(LegLines, labels,
        bbox_to_anchor=(0.5, 1.05), loc='lower center',
        #bbox_to_anchor=(1.05, 1.), loc='upper left',
          ncol=1,# mode="expand",
            fontsize=13,
          borderaxespad=0., handlelength=4.5)
    #plt.legend(loc='upper right')
    [xmin, xmax, ymin, ymax] = plt.axis()
    if normd:
        plt.axis([xmin, xmax, 0.4, 1.2])
    else:
        plt.axis([xmin, xmax, 0, 1.2])


    fig, ax = plt.subplots(num='Soil types')
    dashes = [(1000,1), (35,5), (20,7.5), (10,10)]
    colors2 = ['0.0', '0.1', '0.2', '0.3']
    labels = ['Clean Sand',
              'General',
              'Sands with High Fines Contents',
              'Silts']

    for soil, dash, label, color in zip([1, 0, 2, 3], dashes, labels,
                                        colors2):
        print('Soil Type: {}'.format(soil))
        N160 = Drs[1] ** 2 / 15. ** 2
        dWone = np.zeros(sigveff.shape)
        taus = np.zeros(sigveff.shape)

        tau1 = CSRs[1] * 101.325
        dW1, G1, D1, gam1 = calcdWone(sigveff1, Degrad, PI, tau1, N160,
        soil=soil)
        dW1n = dW1 / sigveff1[0]

        Ksig, tau, dW = calcKsigEn(sigveff, dW1, tau1, PI=PI, soil=soil,
               Degrad='DS', verbose=verbose, normd=normd)
        ax.plot(sigveff, Ksig, dashes=dash, lw=2.5, color=color,
               label=label)
        if soil == 1:
            plt.figure('Main Event')
            plt.plot(sigveff/101.325, Ksig, '-', lw=lws[1], color=colors[0],
                   label=labels[1])
        if soil == 2:
            plt.figure('Main Event')
            plt.plot(sigveff/101.325, Ksig, '-', lw=lws[2], color=colors[0],
                   label=labels[1])
    ax.grid('on', alpha=0.5)
    ax.set_xlabel('$\sigma\'_v / P_a$')
    ax.set_ylabel('$K_{\sigma}$')
    leg2 = ax.legend(
        #bbox_to_anchor=(1.05, 1.), loc='upper left',
        fontsize=12,
        bbox_to_anchor=(0.5, 1.02), loc='lower center',
          ncol=1,# mode="expand",
```

```
          borderaxespad=0., handlelength=4.5)
    #plt.legend(loc='upper right')
    [xmin, xmax, ymin, ymax] = ax.axis()
    if normd:
        ax.axis([xmin, xmax, 0.4, 1.2])
    else:
        ax.axis([xmin, xmax, 0, 1.2])

    savefig = True
    if savefig:
        if normd:
            plt.figure('Main Event')
            dirout =
                 '/media/Storage/Documents/00-VtResearch/Thesis/Markdown/Figures/LimitState/'
            plt.savefig(dirout + 'KsigmaNormd.png', dpi=80, bbox_extra_artists=(leg1,),
                 bbox_inches='tight')
            plt.savefig(dirout + 'KsigmaNormd.pdf', dpi=100, bbox_extra_artists=(leg1,),
                 bbox_inches='tight')
            fig.savefig(dirout + 'KsigmaNormdcompare.pdf', dpi=100,
                 bbox_extra_artists=(leg2,), bbox_inches='tight')
            fig.savefig(dirout + 'KsigmaNormdcompare.png', dpi=100,
                 bbox_extra_artists=(leg2,), bbox_inches='tight')
        else:
            plt.figure('Main Event')
            dirout =
                 '/media/Storage/Documents/00-VtResearch/Thesis/Markdown/Figures/LimitState/'
            plt.savefig(dirout + 'Ksigma.png', dpi=80, bbox_extra_artists=(leg1,),
                 bbox_inches='tight')
            plt.savefig(dirout + 'Ksigma.pdf', dpi=100, bbox_extra_artists=(leg1,),
                 bbox_inches='tight')
            fig.savefig(dirout + 'Ksigmacompare.pdf', dpi=100, bbox_extra_artists=(leg2,),
                 bbox_inches='tight')
            fig.savefig(dirout + 'Ksigmacompare.png', dpi=100, bbox_extra_artists=(leg2,),
                 bbox_inches='tight')
    plt.show()
    pdb.set_trace()
```

J.8 Data Reduction of CSS Lab Data

The file `gcts-cv-css.py` takes the output files from CV-CSS tests on the GCTS equipment and reduces the data.

```
,
import os
import sys
if os.name == 'posix':
    sys.path.append('/media/Storage/Documents/Python/Modules')
    sys.path.append('/media/Storage/Documents/Python/CaseHistories/')
else:
    sys.path.append('/../../Documents/Python/Modules')
    sys.path.append('/../../Documents/Python/CaseHistories/')
try:
```

```
    import ipdb as pdb
except:
    import pdb
import glob
import csv
import numpy as np
import matplotlib
#matplotlib.rcParams.update({'font.size': 18, 'text.usetex': True,
#'legend.fontsize': 16})
matplotlib.rcParams.update({'font.size': 18, 'font.family': 'STIXGeneral',
       'mathtext.fontset': 'stix',
       'legend.fontsize': 16})
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
import statsmodels.api as sm

import pandas as pd
import pickle as p
import qtfiledialog as fd

import tables as pt
from time import strftime
import datetime as dt
from scipy.interpolate import interp1d
import degrcurv as dc
from CaseHistoriesv3 import getDegrad, DobryIter, TokimatsuSeedDirect


try: # Makes the code okay to use in Python3, not fully tested.
    xrange(10)
#    raw_input('This is Python 2, press "Enter"')
    basestring
except NameError:
    xrange = range
    raw_input = input
    basestring = str

class SumTable(pt.IsDescription):
    '''
    PyTables summary table format for stress-controlled soil tests.
    '''
    File = pt.StringCol(itemsize=32,pos=1)
    eo = pt.Float32Col(pos=2)
    Dr = pt.Float32Col(pos=3)
    CSR = pt.Float32Col(pos=4)
    Sigveff = pt.Float32Col(pos=5)
    Cycles = pt.Float32Col(pos=6)
    DE = pt.Float32Col(pos=7)
    DE_corr = pt.Float32Col(pos=8)
    DE_b = pt.Float32Col(pos=9)
    DE_m = pt.Float32Col(pos=10)
    DE_1 = pt.Float32Col(pos=11)
    Gs = pt.Float32Col(pos=12)
    mass = pt.Float32Col(pos=13)
    hi = pt.Float32Col(pos=14)
    di = pt.Float32Col(pos=15)
    PEC = pt.Float32Col(pos=16)
```

```
    LoadCorr = pt.BoolCol(pos=17)
    LoadCorrType = pt.StringCol(itemsize=32, pos=18)
    WaterTestName = pt.StringCol(itemsize=32,pos=19)
    Modified = pt.StringCol(itemsize=32,pos=20)
    Date_of_Test = pt.Int64Col(pos=21)
    failuretype = pt.StringCol(itemsize=32, pos=22)
    DE_corralt = pt.Float32Col(pos=8)
    MaxStrain = pt.Float32Col(pos=23)
    MaxStress = pt.Float32Col(pos=24)
    r_u_fin = pt.Float32Col(pos=25)
    r_u_fin_zero_stress = pt.Float32Col(pos=26)

class StrainSumTable(pt.IsDescription):
    '''
    PyTables summary table format for strain-controlled
    soil tests.
    '''
    File = pt.StringCol(itemsize=32,pos=1)
    eo = pt.Float32Col(pos=2)
    Dr = pt.Float32Col(pos=3)
    CSR = pt.Float32Col(pos=4)
    Gamma = pt.Float32Col(pos=4)
    Sigveff = pt.Float32Col(pos=5)
    Cycles = pt.Float32Col(pos=6)
    DE = pt.Float32Col(pos=7)
    DE_corr = pt.Float32Col(pos=8)
    DE_b = pt.Float32Col(pos=9)
    DE_m = pt.Float32Col(pos=10)
    DE_1 = pt.Float32Col(pos=11)
    Gs = pt.Float32Col(pos=12)
    mass = pt.Float32Col(pos=13)
    hi = pt.Float32Col(pos=14)
    di = pt.Float32Col(pos=15)
    PEC = pt.Float32Col(pos=16)
    LoadCorr = pt.BoolCol(pos=17)
    LoadCorrType = pt.StringCol(itemsize=32, pos=18)
    WaterTestName = pt.StringCol(itemsize=32,pos=19)
    Modified = pt.StringCol(itemsize=32,pos=20)
    Date_of_Test = pt.Int64Col(pos=21)
    failuretype = pt.StringCol(itemsize=32, pos=22)
    DE_corralt = pt.Float32Col(pos=8)
    MaxStrain = pt.Float32Col(pos=23)
    MaxStress = pt.Float32Col(pos=24)
    r_u_fin = pt.Float32Col(pos=25)
    r_u_fin_zero_stress = pt.Float32Col(pos=26)

class EQSumTable(pt.IsDescription):
    '''
    PyTables summary table format for soil tests with
    earthquake loading functions.
    '''
    File = pt.StringCol(itemsize=32,pos=1)
    eo = pt.Float32Col(pos=2)
    Dr = pt.Float32Col(pos=3)
    CSR = pt.Float32Col(pos=4)
    Sigveff = pt.Float32Col(pos=5)
    Sigveff_fin = pt.Float32Col(pos=6)
```

```
    r_u_fin = pt.Float32Col(pos=7)
    DE = pt.Float32Col(pos=7)
    DE_corr = pt.Float32Col(pos=8)
    DE_b = pt.Float32Col(pos=9)
    DE_m = pt.Float32Col(pos=10)
    DE_1 = pt.Float32Col(pos=11)
    Ar = pt.Float32Col()
    PTratio = pt.Float32Col()
    ElapsedTime = pt.Float32Col()
    Gs = pt.Float32Col(pos=12)
    mass = pt.Float32Col(pos=13)
    hi = pt.Float32Col(pos=14)
    di = pt.Float32Col(pos=15)
    PEC = pt.Float32Col(pos=16)
    LoadCorr = pt.BoolCol(pos=17)
    LoadCorrType = pt.StringCol(itemsize=32, pos=18)
    WaterTestName = pt.StringCol(itemsize=32,pos=19)
    Modified = pt.StringCol(itemsize=32,pos=20)
    Date_of_Test = pt.Int64Col(pos=21)
    failuretype = pt.StringCol(itemsize=32, pos=22)
    Cycles = pt.Float32Col(pos=23)
    Total_Pts = pt.Float32Col(pos=24)
    CSR_calcd = pt.Float32Col(pos=25)
    EQ = pt.StringCol(itemsize=32, pos=26)
    Neq = pt.Float32Col(pos=27)
    Gmax = pt.Float32Col(pos=28)
    DE_corralt = pt.Float32Col(pos=8)


class EQCSR_stuff(pt.IsDescription):
    '''
    PyTable formats for a summary table of eqs
    '''
    File = pt.StringCol(itemsize=32, pos=1)
    CSR_nom = pt.Float32Col(pos=2)
    EQ = pt.StringCol(itemsize=32, pos=3)

class WaterSumTable(pt.IsDescription):
    '''
    Format of output table for friction tests without a surcharge
    '''
    File = pt.StringCol(itemsize=32,pos=1)
    Modified = pt.StringCol(itemsize=32,pos=2)
    posm = pt.Float32Col(pos=3)
    posb = pt.Float32Col(pos=4)
    negm = pt.Float32Col(pos=5)
    negb = pt.Float32Col(pos=6)
    Load_unit = pt.StringCol(itemsize=16,pos=7)
    Disp_unit = pt.StringCol(itemsize=16,pos=8)
    Date_of_Test = pt.Int64Col(pos=9)
    Gmax = pt.Float32Col(pos=10)
    G2 = pt.Float32Col(pos=11)
    gamr = pt.Float32Col(pos=12)
    voffset = pt.Float32Col(pos=13)
    hypa = pt.Float32Col(pos=14)
    hypb = pt.Float32Col(pos=15)

```

```
class SurchargeSumTable(pt.IsDescription):
    '''
    Format of output table for friction tests with a surcharge
    '''
    File = pt.StringCol(itemsize=32,pos=1)
    Modified = pt.StringCol(itemsize=32,pos=2)
    posm = pt.Float32Col(pos=3)
    posb = pt.Float32Col(pos=4)
    negm = pt.Float32Col(pos=5)
    negb = pt.Float32Col(pos=6)
    surcharge = pt.Float32Col(pos=7)
    DE = pt.Float32Col(pos=7)
    DE_corr = pt.Float32Col(pos=8)
    Load_unit = pt.StringCol(itemsize=16,pos=9)
    Disp_unit = pt.StringCol(itemsize=16,pos=10)
    Date_of_Test = pt.Int64Col(pos=11)

def moviemaker((t,x,y), **kwargs):
    '''
    This will create a movie of the friction test hysteresis loops.
    Deprecated because 'mencoder' isn't available anymore
    '''
    numy = kwargs.get('num_y', 1)
    syms = kwargs.get('syms',['ro','b', 'g', '.k'])
    no_frames = kwargs.get('no_frames', 100)
    frames_stay = kwargs.get('frames_stay', 1)
    xlabel = kwargs.get('xlabel','x')
    ylabel = kwargs.get('ylabel','y')
    tlabel = kwargs.get('tlabel','Time')
    moviename = kwargs.get('moviename', 'Movie1')
    fixedaxes = kwargs.get('fixedaxes', False)
    if fixedaxes == True:
        xmax = np.max(x) + 0.05 * (np.max(x) - np.min(x))
        xmin = np.min(x) - 0.05 * (np.max(x) - np.min(x))
        ymax = np.max(y) + 0.05 * (np.max(y) - np.min(y))
        ymin = np.min(y) - 0.05 * (np.max(y) - np.min(y))
    fps = kwargs.get('fps',5)
    step = len(t) / no_frames
    files = []

    fig, ax = plt.subplots(figsize=(5,5))
    for i in range(no_frames):
        im1 = max(0,i-frames_stay)
        ind = step * (i + 1)
        indm1 = step * (im1)
        plt.cla()
        if numy == 1:
            plt.plot(x[indm1:ind], y[indm1:ind], syms[0])
        else:
            for j,yy in enumerate(y):
                plt.plot(x[indm1:ind], yy[indm1:ind], syms[j])
        if fixedaxes != True:
            try:
                xmin = np.min(x[indm1:ind])
                xmax = np.max(x[indm1:ind])
                ymin = np.min(yy[indm1:ind])
                ymax = np.max(yy[indm1:ind])
```

```
238             except ValueError:
239                 pdb.set_trace()
240         plt.xlabel(xlabel)
241         plt.ylabel(ylabel)
242         plt.title('{}: {}-{}'.format(tlabel,t[indm1], t[ind]))
243         plt.axis([xmin,xmax,ymin,ymax])
244         fname = '_tmp%03d.png'%i
245         print('Saving frame', fname)
246         plt.savefig(fname)
247         files.append(fname)
248 
249     print('Making movie animation.mpg - this make take a while')
250     os.system("mencoder 'mf://_tmp*.png' -mf type=png:fps={} -ovc lavc -lavcopts
            vcodec=wmv2 -oac copy -o {}.mpg".format(fps,moviename))
251     #os.system("convert _tmp*.png animation.mng")
252 
253     # cleanup
254     for fname in files: os.remove(fname)
255 
256 def nextpow2(i):
257     '''
258     Returns the the log-base-2 of the next power of 2 greater than the input
259     value.
260 
261     Example:
262     >>> y = nextpow2(5)
263     >>> print y
264     3
265     >>>2 **nextpow2(5)
266     8
267     '''
268     n = 2
269     while n < i:
270         n = n * 2
271     return int(round(np.log2(n)))
272 
273 def datetimeconvert(datestr,timestr):
274     '''
275     Convenience function for working with dates
276     '''
277     date = dt.datetime.strptime(datestr+timestr,
278                                 '%m/%d/%y%H:%M:%S')
279     return int(date.strftime('%y%m%d%H%M%S'))
280 
281 def getMotionName(basename):
282     '''
283     Helps me keep the eq motion tests organized
284     '''
285     motionpool = ['NGA_no_87_SAD273',
286                   'NGA_no_45_DCF090',
287                   'NGA_no_283_A-ARI000',
288                   'NGA_no_2658_TCU129-N',
289                   'NGA_no_1060_CUC090',
290                   'NGA_no_1060_CUC180',
291                   'NGA_no_23_GGP010',
292                   'NGA_no_45_DCF090_inv',
293                   'NGA_no_87_SAD273_inv',
```

```
                'NGA_no_1060_CUC090_inv',
                'NGA_no_283_A-ARI000_inv']
    print('What is the ground motion for {}:\n'.format(basename))
    for i, val in enumerate(motionpool):
        print('   [{}]. {}'.format(i+1, val))
    print('   [0]: Input the motion yourself.')

    choice = int(raw_input('Choose: >>> '))
    if choice == 0:
        name_of_motion = raw_input('Motion Name (Ex. NGA_no_87_SAD273) >>> ')
    else:
        name_of_motion = motionpool[choice-1]
    return name_of_motion

def unit_lookup(header1,header2,Label):
    '''
    Returns the unit of a given variable (Label).
    header1 is an array of labels: Shear Load, Shear Displacement, etc
    header2 is an array of units corresponding to the labels: kN, mm, etc
    Label is label for which a unit is desired.
    '''
    for i,label in enumerate(header1):
        if Label == label:
            return header2[i]
            break

def getsign(sign):
    '''
    Should use np.sign() instead of this
    '''
    i = 0
    mys = sign[i]
    if mys != 0:
        return mys
    else:
        while mys == 0:
            i += 1
            if i > 30:
                pdb.set_trace()
            mys = np.sign(np.mean(sign[:i]))
    return mys

def calcDissEnergy(stress, strain, **kwargs):
    '''
    Calculates the dissipated energy from a stress and strain time history.
    Output is the same length as the input time history.
    '''
    area = kwargs.get('area', np.ones(stress.shape, dtype=float))
    ht = kwargs.get('ht', np.ones(stress.shape, dtype=float))
    stress = stress / area
    strain = strain / ht
    dissEn = np.zeros(stress.shape, dtype=float)
    dissEn = np.cumsum((stress[1:] + stress[:-1]) *
                            (strain[1:] - strain[:-1])) * 0.5
    return dissEn
```

```
def PeakFromP2P(th, **kwargs):
    '''
    Returns the index of peaks using the peak to peak method.
    This is the Simple-Range Counting Method from Section 5.3 of ASTM
    1049 - 85 (2011) where both positive and negative ranges are counted.
    Optional Keyword arguments:
    thrshld = 0 [default]. Threshold load under which no cycles are
    counted.
    '''
    thrshld = kwargs.get('thrshld', 0.0005)
    if thrshld == None:
        print('You\'ve got to specify the threshold!!!')
        thrshold = float(input('What will it be? >>> '))
    zero = 0.
    load = th
    dLoad = np.diff(load)
    ind = np.where(((dLoad[:-1] >= zero) & (dLoad[1:] < zero)) |
           ((dLoad[:-1] < zero) & (dLoad[1:] >= zero)) )
    ind2 = ind[0] + 1
    peaks = load[(ind2)]
    Amp = np.abs(peaks[1:] - peaks[:-1]) / 2
    indout = ind2[np.where((Amp!=0) & (Amp >= thrshld))[0]+1]
    if indout[0] == 0 or indout[0] == 1:
        return indout[1:]
    else:
        return indout

def PeakFromZC(th, **kwargs):
    '''
    Returns the index of the peak using zero crossings.
    '''
    calcZero = kwargs.get('calcZero', True)
    if calcZero:
        lenth = len(th)
        zero = np.mean(th[:lenth/2])
    else:
        zero = kwargs.get('zero', 0.)
    ind = np.where(((th[:-1] >= zero) & (th[1:] < zero)) |
           ((th[:-1] < zero) & (th[1:] >= zero)) )

    indout = np.zeros(len(ind[0]), dtype=int)
    for i,ind2 in enumerate(ind[0]):
        if i == 0:
            if ind2 == 0:
                indout[i] = 0.
                continue
            else:
                indout[i] = np.abs(th[:ind2] - zero).argmax(axis=0)

        else:
            indout[i] = (ind[0][i-1] +
                       np.abs(th[ind[0][i-1]:ind2] - zero).argmax(axis=0))

            #    plt.figure('yeahyeah')
            #    plt.plot(th)
            #    plt.plot(indout, th[indout], 'o')
            #    plt.show()
```

```
            #    pdb.set_trace()
    if indout[0] == 0:
        return indout[1:]
    else:
        return indout

def calcAriasRatio(motion,dt, **kwargs):
    '''
    Calculates the following:
    a_rms / (max(abs(motion)) )

    It should be unitless.

    '''
    if 1 in motion.shape or len(motion.shape) == 1:
        NumMotions = 1
    else:
        NumMotions = np.min(motion.shape)
    Ar = np.zeros([NumMotions,], dtype=float)
    if NumMotions == 1:
        Ar = (calcRmsAccel(motion, dt, **kwargs) /
                    np.max(np.abs(motion))) # Assume data is in rows
    else:
        for i in xrange(NumMotions):
            Ar[i] = (calcRmsAccel(motion[i,:], dt, **kwargs) /
                    np.max(np.abs(motion[i,:]))) # Assume data is in rows

    return Ar


def calcRmsAccel(motion, dt, **kwargs):
    '''
    Calculates the rms acceleration.

    Keyword Arguments:
        duration       5to95 [default] Uses the duration from 5-95% of
                          the area under the motion squared
                       total       Uses the entire time of the input motion

    '''
    motion = np.ravel(motion)
    durationtype = kwargs.get('duration','5to95')
    NPTS = len(motion)
    if len(dt) > 1:
        time = np.cumsum(dt)
        asqr = np.cumsum(motion ** 2) * np.max(dt)
    else:
        time = np.linspace(0,(NPTS-1) * dt,NPTS)
        asqr = np.cumsum(motion ** 2) * dt

    if durationtype == '5to95':
        int1 = interp1d( asqr, time, kind='linear')
        T = int1(0.95 * asqr[-1]) - int1(0.05 * asqr[-1])
    else:
        T = time[-1]

    return np.sqrt(asqr[-1] / T)
```

```
465
466
467 class hyperbolic(object):
468     '''
469     To handle hyperbolic stress-strain relationships.
470     Equations taken from Hardin, B. O., and Drnevich, V. P. (1972).
471     "Shear modulus and damping in soils." Journal of the Soil Mechanics and Foundations
          Division, 98(7), 667-692.
472     This wasn't really working last time I tried it.
473     '''
474     def __init__(self, Gmax, taumax, **kwargs):
475         '''
476         Initializes the object to prepare for calculating values.
477         '''
478         self.Gmax = Gmax
479         self.taumax = taumax
480         self.gamr = taumax / Gmax
481         self.a = kwargs.get('a', -0.5)
482         self.b = kwargs.get('b', 0.16)
483    self.voffset = kwargs.get('voffset', 0.)
484
485
486     def __call__(self, gam, **kwargs):
487         '''
488         Calculate your tau from your gam
489         '''
490         Gmax = kwargs.get('Gmax',self.Gmax)
491         taumax = self.taumax
492         gamr = self.gamr
493         a = kwargs.get('a', self.a)
494         b = kwargs.get('b', self.b)
495         mottype = kwargs.get('mottype','sine')
496         window_len = kwargs.get('window_len',11)
497         if kwargs.get('smooth', True) == True:
498
499             from smooth import smooth
500             gam_sm = np.zeros(gam.shape, dtype=float)
501             gam_sm[1:] = smooth(gam, window='flat',
502                         window_len=window_len)[window_len/2:-window_len/2]
503         else:
504             gam_sm = gam
505         # find the stress reverses:
506         if mottype == 'sine':
507             ind = PeakFromZC(gam_sm)
508         else:
509             ind = PeakFromP2P(gam_sm)
510         tau = np.zeros(gam.shape, dtype=float)
511         printer = False
512         # Do my calcs
513         gam_r = gamr
514         tau_max = taumax
515         for i in xrange(len(ind)):
516             if i == 0:
517                 gambase = gam_sm[0]
518                 taubase = 0.
519                 mysign = np.sign(np.mean(np.diff(gam_sm[:ind[i]])))
520                 val = gam_sm[:ind[i]]
```

```
521                 tau[:ind[i]] = (taubase + mysign*np.abs(val - gambase) /
522                        ((1. / Gmax) + np.abs(val - gambase) *
523                            (1. + a * np.exp(-b * (np.abs(val - gambase)
524                                / gam_r))) / tau_max))
525                 gam_r = 2 * gamr
526       tau_max = 2 * taumax
527                 gambase = gam_sm[ind[i]-1]
528                 taubase = tau[ind[i]-1]
529             else:
530                 val = gam_sm[ind[i-1]:ind[i]]
531                 mysign = np.sign(np.mean(np.diff(gam_sm[ind[i-1]:ind[i]])))
532                 tau[ind[i-1]:ind[i]] = (taubase +
533                           mysign*np.abs(val - gambase) / ((1. / Gmax)
534                           + np.abs(val - gambase) *
535                           (1. + a * np.exp(-b * (np.abs(val - gambase) /
536             gam_r))) / tau_max))
537                 gambase = gam_sm[ind[i]-1]
538                 taubase = tau[ind[i]-1]
539         val = gam_sm[ind[i]:]
540         mysign = np.sign(np.mean(np.diff(gam_sm[ind[i]:])))
541         tau[ind[i]:] = (taubase +
542                    mysign*np.abs(val - gambase) / ((1. / Gmax)
543                    + np.abs(val - gambase) *
544                    (1. + a * np.exp(-b * (np.abs(val - gambase) /
545                        gam_r))) / tau_max))
546         if printer:
547             plt.plot(gam_sm,tau)
548             plt.show()
549             pdb.set_trace()
550         return tau + self.voffset
551
552
553     def findab(self, gam, tau, **kwargs):
554         '''
555         Tries to calculate a and b from the data
556         '''
557         from scipy.optimize import curve_fit
558         returns = kwargs.get('True', True)
559
560         def finder1(x,a,b,Gmax):
561                return self.__call__(gam, a=a, b=b, Gmax=Gmax)
562
563         def finder2(x,a,b):
564                return self.__call__(gam, a=a, b=b)
565
566         pdb.set_trace()
567         if kwargs.get('Gmax', True) == True:
568             p0 = kwargs.get('p0', [self.a, self.b, self.Gmax])
569             self.popt, self.pov = curve_fit(finder1,gam, tau, p0=p0)
570             self.a, self.b, self.Gmax = self.popt[0], self.popt[1], self.popt[2]
571             if returns:
572                 return self.a, self.b, self.Gmax
573         else:
574             p0 = kwargs.get('p0', [self.a, self.b])
575             self.popt, self.pov = curve_fit(finder2,gam, tau, p0=p0)
576             self.a, self.b = self.popt[0], self.popt[1]
577             if returns:
```

```
578                 return self.a, self.b
579
580 class bilinear(object):
581     '''
582     To handle bi-linear stress-strain relationships.
583     Also not really working last time I tried it.
584     Needs some serious love. "Will you adopt me?!?"
585     '''
586     def __init__(self, Gmax, G2, gamr, **kwargs):
587         '''
588         Initializes the object to prepare for calculating values.
589         '''
590         self.Gmax = Gmax
591         self.G2 = G2
592         self.gamr = gamr
593         self.voffset = kwargs.get('voffset', 0.)
594
595     def __call__(self, gam, **kwargs):
596         '''
597         Calculate your tau from your gam
598         '''
599         Gmax = kwargs.get('Gmax',self.Gmax)
600         m = self.G2
601         gamr = self.gamr
602         voffset = self.voffset
603         mottype = kwargs.get('mottype','sine')
604         window_len = kwargs.get('window_len',11)
605         if kwargs.get('smooth', True) == True:
606
607             from smooth import smooth
608             gam_sm = np.zeros(gam.shape, dtype=float)
609             gam_sm[1:] = smooth(gam, window='flat',
610                         window_len=window_len)[window_len/2:-window_len/2]
611         else:
612             gam_sm = gam
613         # find the stress reverses:
614         if mottype == 'sine':
615             ind = PeakFromZC(gam_sm)
616         else:
617             ind = PeakFromP2P(gam_sm)
618         tau = np.zeros(gam.shape, dtype=float)
619         printer = False
620         # Do my calcs
621         gam_r = gamr
622         for i in xrange(len(ind)):
623             if i == 0:
624                 mysign = np.sign(np.mean(np.diff(gam_sm[:ind[i]])))
625                 if np.isnan(mysign):
626                     mysign = 0.
627                 gambase = gam_sm[0]
628                 taubase = 0.
629                 val = gam_sm[:ind[i]]
630                 diff = np.abs(val - gambase)
631                 pt1 = np.min([diff, np.ones(diff.shape) * gam_r], axis=0)
632                 pt2 = np.max([diff - gam_r, np.zeros(diff.shape)], axis=0)
633                 tau[:ind[i]] = taubase + mysign * (pt1 * Gmax + pt2 * m)
634                 gam_r = 2 * gamr
```

```
                gambase = gam_sm[ind[i]-1]
                taubase = tau[ind[i]-1]
            else:
                mysign = np.sign(np.mean(np.diff(gam_sm[ind[i-1]:ind[i]])))
                if np.isnan(mysign):
                    mysign = 0.
                val = gam_sm[ind[i-1]:ind[i]]
                diff = np.abs(val - gambase)
                pt1 = np.min([diff, np.ones(diff.shape) * gam_r], axis=0)
                pt2 = np.max([diff - gam_r, np.zeros(diff.shape)], axis=0)
                tau[ind[i-1]:ind[i]] = taubase + mysign * (pt1 * Gmax + pt2 * m)
                gambase = gam_sm[ind[i]-1]
                taubase = tau[ind[i]-1]
        mysign = np.sign(np.mean(np.diff(gam_sm[ind[i]:])))
        val = gam_sm[ind[i]:]
        diff = np.abs(val - gambase)
        pt1 = np.min([diff, np.ones(diff.shape) * gam_r], axis=0)
        pt2 = np.max([diff - gam_r, np.zeros(diff.shape)], axis=0)
        tau[ind[i]:] = taubase + mysign * (pt1 * Gmax + pt2 * m)
        if printer:
            plt.plot(gam_sm,tau)
            plt.show()
            pdb.set_trace()
        return tau + voffset

class HDF5saver(object):
    '''
    This object will save the data to a pytable HDF5 file.
    Setup to accept all test types.
    '''
    def __init__(self,**kwargs):
        testtype = kwargs.get('testtype','sand')
        self.strain = kwargs.get('strain', False)
        if testtype == 'sand':
            filename = kwargs.get('filename','SummaryResults.h5')

            # Check to see if the output table exist, if not, create it.
            self.outfile = pt.openFile(filename,mode='a',
                               title='CV-CSS Lab Testing Results')
            # Check if the internal summary table exists
            if self.outfile.__contains__('/Summary'):
                self.summtable = self.outfile.root.Summary
            else:
                if self.strain:
                    self.summtable = self.outfile.createTable('/','Summary',
                                                              StrainSumTable)
                else:
                    self.summtable = self.outfile.createTable('/','Summary',
                                                              SumTable)
        elif testtype == 'EQ':
            filename = kwargs.get('filename','EQSummaryResults.h5')

            # Check to see if the output table exist, if not, create it.
            self.outfile = pt.openFile(filename,mode='a',
                               title='EQ CV-CSS Lab Testing Results')
            # Check if the internal summary table exists
            if self.outfile.__contains__('/Summary'):
```

```
692                 self.summtable = self.outfile.root.Summary
693             else:
694                 self.summtable = self.outfile.createTable('/','Summary',
695                                                           EQSumTable)
696         elif testtype == 'water':
697             filename = kwargs.get('filename','WaterSummary.h5')
698 
699             # Check to see if the output table exist, if not, create it.
700             self.outfile = pt.openFile(filename,mode='a',
701                                title='CV-CSS Lab Testing Results')
702             # Check if the internal summary table exists
703             if self.outfile.__contains__('/WaterSummary'):
704                 self.summtable = self.outfile.root.WaterSummary
705             else:
706                 self.summtable = self.outfile.createTable('/',
707                                                           'WaterSummary',
708                                                           WaterSumTable)
709         elif testtype == 'surcharge':
710             filename = kwargs.get('filename','SurchargeSummary.h5')
711             # Check to see if the output table exist, if not, create it.
712             self.outfile = pt.openFile(filename,mode='a',
713                                title='CV-CSS Surcharge Lab Testing Results')
714             # Check if the internal summary table exists
715             if self.outfile.__contains__('/SurchargeSummary'):
716                 self.summtable = self.outfile.root.SurchargeSummary
717             else:
718                 self.summtable = self.outfile.createTable('/',
719                                                           'SurchargeSummary',
720                                                           SurchargeSumTable)
721         self.filename = filename
722         self.testtype = testtype
723 
724 
725 
726     def input(self, ResultObj, **kwargs):
727         '''
728         This is the mechanism whereby data is input into the table.
729         '''
730         # Check to see if an entry already exists
731         name = ResultObj.name
732         name = str(os.path.basename(name))
733         if self.testtype == 'sand':
734             self.delete(name)
735             entry = self.summtable.row
736             entry['File'] = name
737             entry['eo'] = ResultObj.e_o
738             entry['Dr'] = ResultObj.Dr
739             entry['CSR'] = ResultObj.CSR
740             if self.strain:
741                 entry['Gamma'] = ResultObj.gamma
742             entry['Sigveff'] = ResultObj.sigveff
743             entry['Cycles'] = ResultObj.cycles
744             entry['DE'] = ResultObj.DE
745             entry['DE_corr'] = ResultObj.DE_corr
746             entry['DE_b'] = ResultObj.DE_b
747             entry['DE_m'] = ResultObj.DE_m
748             entry['DE_1'] = ResultObj.DE_1
```

```
749             entry['Gs'] = ResultObj.Gs
750             entry['mass'] = ResultObj.mass
751             entry['hi'] = ResultObj.hi
752             entry['di'] = ResultObj.di
753             entry['PEC'] = ResultObj.PEC
754             entry['LoadCorr'] = ResultObj.loadcorr
755             if ResultObj.loadcorr:
756                 entry['LoadCorrType'] = ResultObj.loadcorr_type
757                 entry['WaterTestName'] = ResultObj.WaterTestName
758             entry['Modified'] = strftime("%Y-%m-%d %H:%M:%S")
759             entry['Date_of_Test'] = ResultObj.date_of_test
760             entry['failuretype'] = ResultObj.cycletype
761             entry['DE_corralt'] = ResultObj.dW_corralt
762             entry.append()
763             self.summtable.flush()
764 
765         if self.testtype == 'EQ':
766             rowdata = self.delete(name, returns=True)
767 #            if rowdata == []:
768 #
769 #                try:
770 #                    CSR, EQ = self.EQExtradata(name)
771 #                except:
772 #                    CSR, EQ = eqCSR(ResultObj.name, table=self.EQExtraTable,
773 #                          returns=True)
774 #            else:
775             try:
776                 CSR, EQ = self.EQExtradata(name)
777             except:
778                 CSR, EQ = eqCSR(ResultObj.name, table=self.EQExtraTable,
779                       returns=True)
780 
781 #                ResultObj.CSR =
782                 # extract the old CSR and EQ data
783             entry = self.summtable.row
784             entry['File'] = name
785             entry['eo'] = ResultObj.e_o
786             entry['Dr'] = ResultObj.Dr
787             entry['CSR'] = CSR
788             entry['Sigveff'] = ResultObj.sigveff
789             entry['Sigveff_fin'] = ResultObj.sigveff_fin
790             entry['r_u_fin'] = ResultObj.r_u_fin
791             entry['DE'] = ResultObj.DE
792             entry['DE_corr'] = ResultObj.DE_corr
793             entry['DE_b'] = ResultObj.DE_b
794             entry['DE_m'] = ResultObj.DE_m
795             entry['DE_1'] = ResultObj.DE_1
796             entry['Ar'] = ResultObj.Ar
797             entry['PTratio'] = ResultObj.PTratio
798             entry['ElapsedTime'] = ResultObj.cyc['Time'].values[-1]
799             entry['Gs'] = ResultObj.Gs
800             entry['mass'] = ResultObj.mass
801             entry['hi'] = ResultObj.hi
802             entry['di'] = ResultObj.di
803             entry['PEC'] = ResultObj.PEC
804             entry['LoadCorr'] = ResultObj.loadcorr
805             if ResultObj.loadcorr:
```

```
806                 entry['LoadCorrType'] = ResultObj.loadcorr_type
807                 entry['WaterTestName'] = ResultObj.WaterTestName
808             entry['Modified'] = strftime("%Y-%m-%d %H:%M:%S")
809             entry['Date_of_Test'] = ResultObj.date_of_test
810             entry['failuretype'] = ResultObj.cycletype
811             entry['Cycles'] = ResultObj.cycles
812             entry['Total_Pts'] = ResultObj.total_pts
813             entry['CSR_calcd'] = ResultObj.CSR
814             entry['EQ'] = EQ
815             entry['Neq'] = ResultObj.Neq
816             entry['Gmax'] = ResultObj.Gmax
817             entry['DE_corralt'] = ResultObj.dW_corralt
818             entry.append()
819             self.summtable.flush()
820 
821         if self.testtype == 'water':
822 #            pdb.set_trace()
823             entry = self.summtable.row
824             entry['File'] = name
825             entry['Modified'] = strftime("%Y-%m-%d %H:%M:%S")
826             entry['posm'] = ResultObj.poscoeff[0]
827             entry['posb'] = ResultObj.poscoeff[1]
828             entry['negm'] = ResultObj.negcoeff[0]
829             entry['negb'] = ResultObj.negcoeff[1]
830             entry['Load_unit'] = unit_lookup(ResultObj.headerlabel,
831                                              ResultObj.units,
832                                              'Internal Shear Load')
833             entry['Disp_unit'] = unit_lookup(ResultObj.headerlabel,
834                                              ResultObj.units,
835                                              'Internal Shear LVDT')
836             entry['Date_of_Test'] = ResultObj.date_of_test
837             entry['Gmax'] = ResultObj.Gmax
838             entry['G2'] = ResultObj.G2
839             entry['gamr'] = ResultObj.gamr
840             entry['voffset'] = ResultObj.voffset
841             entry['hypa'] = ResultObj.hypa
842             entry['hypb'] = ResultObj.hypb
843             entry.append()
844             self.summtable.flush()
845 
846         if self.testtype == 'surcharge':
847             entry = self.summtable.row
848             entry['File'] = name
849             entry['Modified'] = strftime("%Y-%m-%d %H:%M:%S")
850             entry['posm'] = ResultObj.poscoeff[0]
851             entry['posb'] = ResultObj.poscoeff[1]
852             entry['negm'] = ResultObj.negcoeff[0]
853             entry['negb'] = ResultObj.negcoeff[1]
854             entry['surcharge'] = ResultObj.surcharge
855             entry['DE'] = ResultObj.PDE
856             entry['DE_corr'] = ResultObj.PDE_corr
857             entry['Load_unit'] = unit_lookup(ResultObj.headerlabel,
858                                              ResultObj.units,
859                                              'Shear Load')
860             entry['Disp_unit'] = unit_lookup(ResultObj.headerlabel,
861                                              ResultObj.units,
862                                              'Shear Displacement')
```

```
            entry['Date_of_Test'] = ResultObj.date_of_test
            entry.append()
            self.summtable.flush()


    def read(self):
        '''
        Reads in the result of the Summary Table.
        '''
        results = self.summtable.read()
        return results

    def plot(self, **kwargs):
        if kwargs.get('latex', True):
            fs = kwargs.get('fs',18)
            import matplotlib
            #matplotlib.rcParams.update({'font.size': 18, 'text.usetex': True,
            #'legend.fontsize': 16})
            matplotlib.rcParams.update({'font.size': 18, 'font.family': 'STIXGeneral',
                   'mathtext.fontset': 'stix',
                   'legend.fontsize': 16, 'text.usetex':False})
        else:
            pdb.set_trace()
            import matplotlib
            matplotlib.rcParams.update({'font.size': 12, 'font.family': 'sans',
                 'text.usetex': False})
        cmap = kwargs.get('cmap', 'Set1')
        interval = kwargs.get('interval',24.)
        from mpl_toolkits.mplot3d import axes3d
        CSR = self.summtable.read(field='CSR')
        Dr = self.summtable.read(field='Dr')
        N = self.summtable.read(field='Cycles')
        DE = self.summtable.read(field='DE')
        DE_corr = self.summtable.read(field='DE_corr')
        Sigveff = self.summtable.read(field='Sigveff')
        datefield = kwargs.get('datefield','Modified')
        Date = self.summtable.read(field=datefield)
        delta = np.ones(Date.shape,dtype=float) * 1000.
#       separator = kwargs.get('separator','delta<interval')
#       separator = kwargs.get('separator','Sigveff<75.')
        separator = kwargs.get('separator','sigvcolormap')
        for i in xrange(len(Date)):
            t1 = dt.datetime.now()
            if datefield == 'Modified':
                t2 = dt.datetime.strptime(Date[i].decode("utf-8"),
                                          '%Y-%m-%d %H:%M:%S')
            elif datefield == 'Date_of_Test':
                t2 = dt.datetime.strptime(str(Date[i].decode("utf-8")),
                                          '%y%m%d%H%M%S')
            delt = t1-t2
            delta[i] = delt.total_seconds() / 3600.
        if separator == 'sigvcolormap':

            fig1 = plt.figure(1)
            ax1 = fig1.add_subplot(111, projection='3d')
            st1 = ax1.scatter(CSR,Dr,zs=np.log10(N), c=Sigveff, cmap=cmap)
            ax1.set_zlabel(r'$\log10(No. of cycles to liq)$')
```

```
            ax1.set_ylabel('Dr (%)')
            ax1.set_xlabel('CSR')
            plt.colorbar(st1)

            fig2 = plt.figure(2)
            ax2 = fig2.add_subplot(111, projection='3d')
            st2 = ax2.scatter(CSR,Dr,zs=DE_corr, c=Sigveff, cmap=cmap)
            plt.colorbar(st2)
            ax2.set_zlabel(r'$\Delta W_{corrected}$')
            ax2.set_ylabel('Dr (%)')
            ax2.set_xlabel('CSR')

            fig4 = plt.figure(4)
            ax4 = fig4.add_subplot(111, projection='3d')
            st4 = ax4.scatter(CSR,Dr,zs=np.log10(DE), c=Sigveff, cmap=cmap)
            plt.colorbar(st4)
            ax4.set_zlabel(r'$log(\Delta W)$')
            ax4.set_ylabel('Dr (%)')
            ax4.set_xlabel('CSR')

            fig5 = plt.figure(5)
            ax5 = fig5.add_subplot(111, projection='3d')
            st5 = ax5.scatter(CSR,Dr,zs=(DE / N), c=Sigveff, cmap=cmap)
            plt.colorbar(st5)
            ax5.set_zlabel(r'$\Delta W / N$')
            ax5.set_ylabel('Dr (%)')
            ax5.set_xlabel('CSR')

            fig6 = plt.figure(6)
            ax6 = fig6.add_subplot(111, projection='3d')
            DEn = DE / Sigveff
            st6 = ax6.scatter(CSR,Dr,zs=DEn, c=Sigveff, cmap=cmap)
            plt.colorbar(st6)
            ax6.set_zlabel(r'$\Delta W / \sigma _v$')
            ax6.set_ylabel('Dr (\%)')
            ax6.set_xlabel('CSR')

            plt.figure(3)
            st3 = plt.scatter(N, CSR, c=Sigveff, cmap=cmap)
            plt.xlabel(r'$N$')
            plt.ylabel('CSR')
            plt.colorbar(st3)
            ax = plt.gca()
            ax.set_xscale('log')

        else:
            ind2 = np.where(eval(separator))
            ind1 = np.where(~eval(separator))
            fig = plt.figure(1)
            ax = fig.add_subplot(111, projection='3d')
            ax.scatter(CSR[ind1],Dr[ind1],zs=np.log10(N[ind1]), facecolor='b')
            ax.scatter(CSR[ind2],Dr[ind2],zs=np.log10(N[ind2]), facecolor='r')
            ax.set_zlabel(r'$\log10(No. of cycles to liq)$')
            ax.set_ylabel('Dr (%)')
            ax.set_xlabel('CSR')

            fig = plt.figure(2)
```

```
976             ax = fig.add_subplot(111, projection='3d')
977             ax.scatter(CSR[ind1], Dr[ind1], zs=np.log10(DE_corr[ind1]), facecolor='b')
978             ax.scatter(CSR[ind2], Dr[ind2], zs=np.log10(DE_corr[ind2]), facecolor='r')
979             ax.set_zlabel(r'$log(\Delta W_{corrected})$')
980             ax.set_ylabel('Dr (%)')
981             ax.set_xlabel('CSR')
982 
983             fig = plt.figure(4)
984             ax = fig.add_subplot(111, projection='3d')
985             ax.scatter(CSR[ind1],Dr[ind1],zs=np.log10(DE[ind1]), facecolor='b')
986             ax.scatter(CSR[ind2],Dr[ind2],zs=np.log10(DE[ind2]), facecolor='r')
987             ax.set_zlabel(r'$log(\Delta W)$')
988             ax.set_ylabel('Dr (%)')
989             ax.set_xlabel('CSR')
990 
991             fig = plt.figure(5)
992             ax = fig.add_subplot(111, projection='3d')
993             ax.scatter(CSR[ind1],Dr[ind1],zs=(DE[ind1] / N[ind1]), facecolor='b')
994             ax.scatter(CSR[ind2],Dr[ind2],zs=(DE[ind2] / N[ind2]), facecolor='r')
995             ax.set_zlabel(r'$\Delta W / N$')
996             ax.set_ylabel('Dr (%)')
997             ax.set_xlabel('CSR')
998             plt.figure(3)
999             syms = ['o','x','+']
1000    #        color=['r','b']
1001            Drs = [0,60, 85, 100]
1002            for i in xrange(len(Drs)-1):
1003                ind1 = np.where((delta>interval) & (Dr >= Drs[i]) & (Dr < Drs[i+1]))
1004                plt.semilogx(N[ind1], CSR[ind1],'b'+syms[i],
1005                        label=r'${}<=D_r (\%) <{}$'.format(Drs[i],Drs[i+1]))
1006                ind2 = np.where((delta<=interval) & (Dr >= Drs[i]) & (Dr < Drs[i+1]))
1007                plt.semilogx(N[ind2], CSR[ind2],'r'+syms[i])
1008 
1009            plt.xlabel(r'$N$')
1010            plt.ylabel('CSR')
1011            plt.legend(loc='upper right')
1012 
1013        more = True
1014        if more:
1015            ind100 = np.where((Sigveff >90) & (Sigveff<150))
1016            ind50 = np.where(Sigveff <90)
1017            ind250 = np.where(Sigveff > 150)
1018 
1019            plt.figure('Dr-DE')
1020            plt.plot(Dr[ind100], DE[ind100], 'bo', label=r'$\sigma \prime _v =$ 100 kPa')
1021            plt.plot(Dr[ind50], DE[ind50], 'ro', label=r'$\sigma \prime _v =$ 60 kPa')
1022            plt.plot(Dr[ind250], DE[ind250], 'go', label=r'$\sigma \prime _v =$250 kPa')
1023            plt.xlabel('Relative Density (%)')
1024            plt.ylabel('Dissipated Energy (kPa)')
1025            plt.legend(loc='best')
1026 
1027            plt.figure('Dr-nDE')
1028            plt.plot(Dr[ind100], DE[ind100]/Sigveff[ind100], 'bo', label=r'$\sigma \prime
                    _v =$ 100 kPa')
1029            plt.plot(Dr[ind50], DE[ind50]/Sigveff[ind50], 'ro', label=r'$\sigma \prime _v
                    =$ 60 kPa')
1030            plt.plot(Dr[ind250], DE[ind250]/Sigveff[ind250], 'go', label=r'$\sigma \prime
```

```
                _v =$250 kPa')
            plt.xlabel('Relative Density (%)')
            plt.ylabel('Normalized Dissipated Energy')
            plt.legend(loc='best')


            plt.figure('N-CSR')
            plt.semilogx(N[ind100], CSR[ind100], 'bo', label=r'$\sigma \prime _v =$ 100
                kPa')
            plt.plot(N[ind50],CSR[ind50], 'ro', label=r'$\sigma \prime _v =$ 60 kPa')
            plt.plot(N[ind250], CSR[ind250], 'go', label=r'$\sigma \prime _v =$250 kPa')
            plt.xlabel('Cycles to Liquefaction')
            plt.ylabel('Cyclic Stress Ratio')
            plt.legend(loc='best')

            plt.figure('DE-CSR')
            plt.plot(CSR[ind100], DE[ind100], 'bo', label=r'$\sigma \prime _v =$ 100 kPa')
            plt.plot(CSR[ind50], DE[ind50], 'ro', label=r'$\sigma \prime _v =$ 60 kPa')
            plt.plot(CSR[ind250], DE[ind250], 'go', label=r'$\sigma \prime _v =$250 kPa')
            plt.ylabel('Dissipated Energy (kPa)')
            plt.xlabel('Cyclic Stress Ratio')
            plt.legend(loc='best')
        plt.show()



    def delete(self,*arg, **kwargs):
        '''
        Deletes an entry from an H5 Table.
        '''
        filters = 'csv (*.csv)' # for qtfiledialog
#       filters = 'csv |*.csv' # for wxfiledialog
        returns = kwargs.get('returns', False)
        infilelist = []
        if len(arg) == 0:
#       if kwargs.get('filelist',None) == None:
            filelist=fd.fileopen(
                   message='Select the files you want to remove from the table:',
                   filters=filters,
                   defaultDir=(u'/media/Storage/Documents/Python'+
                            '/GCTS/ConstantVolume'))

            print('Files to remove from {}:\n\n'.format(self.summtable))

            for file1 in filelist:
                infilelist.append(file1)
                print(file1 + '\n')
            del filelist
            print('\n')
            selection = raw_input('Are you sure you want to remove these ' +
            'files from "{}"? [y]/n: '.format(self.filename))

            if selection.lower() == 'n':
                return
            elif selection.lower() == 'y':
                pass
            elif selection == '':
```

```
                pass
            else:
                return
        else:
            for val in arg:
                infilelist.append(val)

        rowdata = []
        for name in infilelist:
            name = str(os.path.basename(name)).split('.')[0]
            try:
                row1 = self.summtable.getWhereList('File == name')
            except NameError:
                row1 = self.summtable.get_where_list('File == name')
            except AttributeError:
                pdb.set_trace()
            if len(row1) == 0:
                pass
            else:
                for i,val in enumerate(row1):
                    if kwargs.get('returns') == True:
                        rowdata.append(self.summtable.read(val,val+1))
                    try:
                        self.summtable.remove_row(val)
                    except NameError:
                        self.summtable.removeRows(val-i)
                    except IndexError, NotImplementedError:
                        pdb.set_trace()
        if returns == True:
            return rowdata


    def EQExtradata(self, name, **kwargs):
        '''
        Gets the input CSR and Earthquake motion name from EQExtra.h5.
        '''
        try:
            self.EQExtraTable
        except AttributeError:
            filename = 'EqExtra.h5'
            outfile = pt.openFile(filename,mode='a')
           # Check if the internal summary table exists
            self.EQExtraTable = outfile.root.Data
    #       testnames = table.read(field='File')
        try:
            row1 = self.EQExtraTable.getWhereList('File == name')
        except NameError:
            row1 = self.EQExtraTable.get_where_list('File == name')

        CSR = self.EQExtraTable.read(row1,row1+1, field='CSR_nom')[0]
        EQ = self.EQExtraTable.read(row1,row1+1, field='EQ')[0]
        if EQ == 'None' or EQ == None:
            CSR, EQ = eqCSR(name, table=self.EQExtraTable,
                            returns=True)
    #           EQ = getMotionName(name)
        return CSR, EQ

```

```
1143
1144     def table2Latex(self, **kwargs):
1145         '''
1146         Prints out a table for use in Latex.
1147         keyword:
1148         cols = ['File', 'Dr', ...]
1149         The tables library would probably work better.
1150         '''
1151         defaultcols = ['File', 'Dr', 'CSR', 'Sigveff', 'Cycles', 'DE']
1152         cols = kwargs.get('cols',defaultcols)
1153
1154         line = r''
1155         for i,val in enumerate(cols):
1156
1157             line += r' {} &'.format(val)
1158         line = line[:-1] + r'\\'
1159         print(line)
1160
1161
1162         for row in self.summtable:
1163             line = r''
1164             for val in cols:
1165                 if val == 'Dr':
1166                     line += r' {:.1f} &'.format(row[val])
1167                 elif val == 'CSR':
1168                     line += r' {:.2f} &'.format(row[val])
1169                 elif val == 'Sigveff':
1170                     line += r' {:.0f} &'.format(row[val])
1171                 elif val == 'Cycles':
1172                     line += r' {:.1f} &'.format(row[val])
1173                 elif val == 'DE':
1174                     line += r' {:.2f} &'.format(row[val])
1175                 elif val == 'CSR_calcd':
1176                     line += r' {:.2f} &'.format(row[val])
1177                 else:
1178                     line += r' {} &'.format(row[val])
1179             line = line[:-1] + r'\\' + '\n'
1180             print(line)
1181
1182     def close(self):
1183         self.outfile.close()
1184
1185 class SoilTest(object):
1186     '''
1187     This object controls all the calcs and i/o for each test.
1188     Keyword          Definition
1189     file             name/path of the GCTS data file for the test.
1190     '''
1191
1192     def __init__(self,**kwargs):
1193         self.Error = False
1194         infile = kwargs.get('infile')
1195         self.emin = kwargs.get('emin',0.53)
1196         self.emax = kwargs.get('emax',0.845)
1197         self.strainfailval = kwargs.get('strainfailval', '03')
1198         self.rufailval = kwargs.get('rufailval', '98')
1199         self.loadcorr = kwargs.get('loadcorr', True)
```

```
        if self.loadcorr:
            self.loadcorr_type = kwargs.get('loadcorr_type', 'bilinear')
        self.waterfile = kwargs.get('waterfile')
        verbose = kwargs.get('verbose',True)
        if verbose: print('\n\n\n')
        self.testtype = kwargs.get('testtype','sine')
        self.extrafigs = kwargs.get('extrafigs', False)
        self.cycletype = kwargs.get('cycletype','wu_etal')
        self.strain = kwargs.get('strain', False)
        self.PaperFigs = kwargs.get('PaperFigs', False)
        if self.testtype == 'EQ':
            self.path = './EQ/'
        else:
            if self.strain:
                self.path = './StrainControlled/'
            else:
                self.path = './Sine/'
        if not os.path.exists(self.path):
            os.makedirs(self.path)

        if 'infile' == None:
            infile = raw_input('Nothing to do. Specify an input file:')
            self.csvimport(infile)
        else:
            self.csvimport(infile)

        self.calcs1(verbose=verbose, **kwargs)
        self.DiscreteCycles()
        errcode = self.dissen(verbose=verbose, **kwargs)
        if self.Error:
            return
        if self.testtype == 'EQ':
            self.EQcalcs()
        self.calcdW_Alt()
        self.plotter()
        if kwargs.get('THout', False):
            self.fileout(**kwargs)

        #if self.Error:
            #return 1
        #else:
            #return 0


    def csvimport(self, infile):
        '''
        This function pulls in the data from the GCTS output file.
        The GCTS file should not have the data separated into stages.
        '''
        self.data=[]
        self.label=[]
        self.name=[]
        self.Gs=[]
        self.mass=[]
        self.cyc_header=[]
        self.pist_area=[]
        datasection=0
```

```
1257            dimsec=0
1258            datafile = csv.reader(open(infile,'rU'))
1259            rowcount = -1
1260            for row in datafile:
1261                rowcount += 1
1262
1263                if datasection==0:
1264                    if self.Gs:
1265                        pass
1266                    elif 'Specific Gravity: ' in row:
1267                        self.Gs=float(row[1])
1268                        self.Gs=2.66 #Because sometimes I forget to change the value in the GCTS
                                software
1269                    if self.name:
1270                        pass
1271                    elif 'Specimen: ' in row:
1272                        self.name=row[1]
1273
1274                    if self.mass:
1275                        pass
1276                    elif 'Dry Mass of Specimen: ' in row:
1277                        self.mass=float(row[1]); self.mass_unit=row[2]
1278                        if self.mass==0:
1279                            self.mass=float(raw_input('Input the dry mass (g) for specimen %s:'
1280                            %self.name))
1281                    if 'Type:' and 'Universal' in row:
1282                        dimsec=1
1283                    if 'Starting Date: ' in row:
1284                        testdate = row[1]
1285                    if 'Starting Time: ' in row:
1286                        testtime = row[1]
1287
1288                if dimsec==1:
1289                    if 'Height: ' in row:
1290                        self.hi=float(row[3])*25.4 ;
1291                        if self.hi > 40:
1292                            self.hi = self.hi - 25.4 * 3.337
1293                            print('Changed the height to {} mm !!!!'.format(
1294                                        self.hi))
1295                        self.hi_unit= 'mm' #Convert to mm
1296                    if 'Diameter: ' in row:
1297                        self.di=float(row[3])*25.4;
1298                        if self.di/25.4 == 4.04:
1299                            print('Changed diameter from {:.3f} mm to 102.36 mm'.format(self.di))
1300                            self.di = 102.362
1301                        self.di_unit= 'mm' #Convert to mm
1302                        self.area = (self.di / 1000 / 2) ** 2 * np.pi # in m^2
1303                        dimsec=0
1304
1305                if len(row) > 17:
1306                    if row[0] == 'Time':
1307                        header = rowcount
1308                        self.headerlabel = row
1309                    if row[0] == 'sec':
1310                        self.units = row
1311                        break
1312
```

```
        df = pd.read_csv(infile,names=self.headerlabel,skiprows=header+2)
        df = df.convert_objects(convert_numeric=True)
        select = df.apply(lambda r : any([isinstance(e, basestring)
                        for e in r ]),axis=1)
        df = df[~select]
        df = df.dropna()

        df['dt'] = df['Time'].diff()
        if self.strain:
            sheardisp = df['Shear Displacement'].values
            sheardisp = sheardisp - sheardisp[0]
            for i,disp in enumerate(sheardisp):
                if np.abs(disp) >=0.02:
                    if i > 35:
                        ind = i - 35
                    else:
                        ind = i
                    break
        else:
            shearload = df['Shear Load'].values
            shearload = shearload - shearload[0]
            for i,load in enumerate(shearload):
                if np.abs(load) >=0.02:
                    if i > 35:
                        ind = i - 35
                    else:
                        ind = i
                    break
        dt = df['dt'][ind:].values
        for i, d in enumerate(dt):
            if d < 0.1:
                ind2 = i + ind
                break
        # Create a new dataframe that contains only the cyclic portion
        columns = ['Time', 'Cycles','Shear Load','Shear Displacement',
                   'Normal Load','Normal Displacement',
                   'Internal Shear Load','Internal Shear LVDT',
                   'Internal Normal LVDT','dt']
        self.cyc = df.loc[ind2:,columns].reset_index(drop=True)
        self.date_of_test = datetimeconvert(testdate,testtime)

    def calcs1(self,**kwargs):
        '''
        Performs the first calculations on the data obtained from the csv file.
        '''
        verbose = kwargs.get('verbose',True)
        self.cycletype = kwargs.get('cycletype','strain01')


        self.cyc['ndisp_ext'] = (self.cyc['Normal Displacement'] -
                            self.cyc['Normal Displacement'][0])
        self.cyc['ndisp_int'] = (self.cyc['Internal Normal LVDT'] -
                            self.cyc['Internal Normal LVDT'][0])


        # Calculate strains
        self.h_cyc = self.hi - self.cyc['Internal Normal LVDT'][0]
```

```
        self.cyc['nstrain_int'] = self.cyc['ndisp_int'] / self.h_cyc
        self.cyc['nstrain_ext'] = self.cyc['ndisp_ext'] / self.h_cyc
        self.cyc['sstrain_ext'] = (self.cyc['Shear Displacement'] /
                                   self.h_cyc)
        self.cyc['sstrain_int'] = (self.cyc['Internal Shear LVDT']
                                   / self.h_cyc)

        # Calculate stresses
        if 'lbf' in self.units:
            for i,unit in enumerate(self.units):
                if unit == 'lbf':
                    if self.headerlabel[i] == 'Normal Load':
                        self.cyc['nload'] = (self.cyc['Normal Load'] *
                                             0.0044482216) # Convert from lbf to kN
                    else:
                        self.cyc['nload'] = self.cyc['Normal Load']

                    break
        self.cyc['nstress'] = self.cyc['nload'] / self.area # in kPa

        # Correct the loads by subtracting out the friction
        if self.loadcorr == True:
            loadcorrf = self.loadcorrection()
            #friction = loadcorrf(self.cyc['Internal Shear LVDT'].values)
            friction = loadcorrf(self.cyc['Shear Displacement'].values,
                                 mottype=self.testtype)
        else:
            friction = np.zeros(self.cyc['nload'].values.shape,
                                dtype=float)
        frictionplots = kwargs.get('frictionplots', False)
        if frictionplots:
            plt.figure(1)
            plt.plot(self.cyc['Internal Shear LVDT'].values, label='LVDT')
            for typ in ['old', 'hyperbolic','bilinear']:
                lc = self.loadcorrection(type=typ,
                    mottype=self.testtype)
                fric = lc(self.cyc['Internal Shear LVDT'].values)
                DE = calcDissEnergy(fric,
                           self.cyc['Internal Shear LVDT'].values,
                           area=self.area,
                           ht=self.h_cyc)
                plt.figure(1)
                plt.plot(fric*100, label='{}Frict*100'.format(typ[:3]))
                plt.legend(loc='best')
                plt.figure(typ)
                plt.plot(self.cyc['Internal Shear LVDT'].values, fric)
                plt.figure('DE')
                plt.plot(DE, label='{}'.format(typ))
                plt.legend(loc='best')
            plt.show()
        self.cyc['sstress_ext'] = ((self.cyc['Shear Load'] - friction)
                                   / self.area) # COnvert to kPa
        self.cyc['sstress_int'] = ((self.cyc['Internal Shear Load'] -
                                   friction) / self.area) # convert to kPa
        self.cyc['G'] = self.cyc['sstress_int'] / self.cyc['sstrain_int']

        # Calculate Volumes, Void Ratios and Relative Densities
```

```
        self.V_cyc = self.area * self.h_cyc / 1000. #cu m
        V_s = self.mass / 1000. / (self.Gs * 1000) #cu m
        self.e_o = (self.V_cyc - V_s) / V_s
        self.Dr = (self.emax - self.e_o) / (self.emax - self.emin) * 100.

        # Calculate the pore pressures
        self.sigveff = self.cyc['nstress'].values[0]
        self.sigveff_fin = self.cyc['nstress'].values[-1]
        self.cyc['Pore Pressure'] = self.sigveff - self.cyc['nstress']
         # Calculate the apparent Residual Excess Pore Pressure, r_u
        self.cyc['Ru'] = self.cyc['Pore Pressure'] / self.sigveff

        # Get the index of zero stress
        sstress_int = self.cyc['sstress_int'].values
        self.ZeroStressInd = np.where(np.sign(sstress_int[:-1]) !=
                      np.sign(sstress_int[1:]))[0]

        # Calculate the number of cycles, determine the point of liquefaction
        if self.cycletype == 'modulus': # Not working?
            self.cycles, self.cycind = self.mod_liq_crit()
        elif self.cycletype == 'strain01':
            ss = self.cyc['sstrain_int'].values
            ind = np.where((ss > 0.01) | (ss < -0.01))
            try:
                self.cycles = self.cyc['Cycles'][ind[0][0]]
                self.cycind = ind[0][0]
            except:
                self.cycles = -1 #self.cyc['Cycles'].values[-1]
                self.cycind = -1 #len(ns)
                print('Shear strain criteria didn\'t work!')
                pdb.set_trace()
        elif self.cycletype == 'strain03SA':
            ss = self.cyc['sstrain_int'].values
            ind = np.where((ss > 0.03) | (ss < -0.03))
            try:
                self.cycles = self.cyc['Cycles'][ind[0][0]]
                self.cycind = ind[0][0]
            except:
                self.cycles = -1 #self.cyc['Cycles'].values[-1]
                self.cycind = -1 #len(ns)
                print('Shear strain criteria didn\'t work!')
        elif self.cycletype == 'strainSA':
            self.cycletype = 'strain{}SA'.format(self.strainfailval)
            failval = float('0.' + self.strainfailval)
            ss = self.cyc['sstrain_int'].values
            ind = np.where((ss > failval) | (ss < -failval))
            try:
                self.cycles = self.cyc['Cycles'][ind[0][0]]
                self.cycind = ind[0][0]
            except:
                self.cycles = -1000 #self.cyc['Cycles'].values[-1]
                self.cycind = -1 #len(ns)
                print('Shear strain criteria didn\'t work!')

        elif self.cycletype == 'strainSAZS':
            self.cycletype = 'strain{}SAZS'.format(self.strainfailval)
            failval = float('0.' + self.strainfailval)
```

```
            sszs = self.cyc['sstrain_int'].values[self.ZeroStressInd]

            ind = np.where((sszs > failval) | (sszs < -failval))
            ind2 = self.ZeroStressInd[ind[0]]
            try:
                self.cycles = self.cyc['Cycles'][ind2[0]]
                self.cycind = ind2[0]
            except:
                self.cycles = -1000 #self.cyc['Cycles'].values[-1]
                self.cycind = -1 #len(ns)
                print('Shear strain criteria didn\'t work!')
        elif self.cycletype == 'normstress':
            ns = self.cyc['nstress'].values
            minval = 1. #Adjust the add value as needed
            ind = np.where(ns<(minval)) #old way
            ss = self.cyc['sstress_int'].values
            stressind = np.where((np.sign(ss[:-1]) != np.sign(ss[1:])) &
                    (ns[:-1] < minval))
            try:
               #self.cycles = self.cyc['Cycles'][ind[0][0]] # old way
               self.cycles = self.cyc['Cycles'][stressind[0][0]]
               self.cycind = ind[0][0]
            except:
                self.cycles = -1 #self.cyc['Cycles'].values[-1]
                self.cycind = -1 #len(ns)
                print('Normal Stress criteria didn\'t work!\n'+
                        ' Or the sample did not liquefy!')
        elif self.cycletype == 'ru':
            #self.rufailval = '99'
            self.cycletype = 'ru' + self.rufailval
            failval = float('0.'+self.rufailval)
            ruzs = self.cyc['Ru'].values[self.ZeroStressInd]
            ind = np.where(ruzs >= failval)
            ind2 = self.ZeroStressInd[ind[0]]
            try:
                self.cycles = self.cyc['Cycles'][ind2[0]]
                self.cycind = ind2[0]
            except:
                self.cycles = -1000 #self.cyc['Cycles'].values[-1]
                self.cycind = -1 #len(ns)
                print('r_u failure criteria didn\'t work!')
        elif self.cycletype == 'wu_etal':
            self.cycles, self.cycind = self.wu_etal_liq_crit()
        elif self.cycletype == 'strainDAZS':
            self.cycletype = 'strain{}DAZS'.format(self.strainfailval)
            failval = float('0.' + self.strainfailval)
            strain = self.cyc['sstrain_int'].values[self.ZeroStressInd]
            stress = self.cyc['sstress_int'].values[self.ZeroStressInd]
            #ind = np.where(np.sign(stress[1:]) != np.sign(stress[:-1]))[0]
            cycles = -1000
            cycind = -1
            for i, val in enumerate(self.ZeroStressInd):
                if (i == 0) or (i == 1):
                    continue
                else:
                    try:
                        val0 = self.ZeroStressInd[i-2]
```

```
                    diff = np.abs(strain[val] - strain[val0])
                except IndexError:
                    pdb.set_trace()
                if diff >= failval:
                    cycles = self.cyc['Cycles'].values[val]
                    cycind = val
                    break
        self.cycles = cycles
        self.cycind = cycind
        if self.cycles < 0.:
            print('Failure criteria didn\'t work: DA {}%'.format(failval))
    elif self.cycletype == 'strainDA':
        self.cycletype = 'strain{}DA'.format(self.strainfailval)
        failval = float('0.' + self.strainfailval)
        strain = self.cyc['sstrain_int'].values
        stress = self.cyc['sstress_int'].values
        ind = np.where(np.sign(stress[1:]) != np.sign(stress[:-1]))[0]
        cycles = -1000
        cycind = -1
        for i, val in enumerate(ind):
            if (ind[i] == 0) or (i == 0) or (i == 1):
                continue
            else:
                try:

                    mycycle = strain[ind[i-2]:ind[i]]
                    diff = np.max(mycycle) - np.min(mycycle)
                except IndexError:
                    pdb.set_trace()
                if diff >= failval:
                    cycles = self.cyc['Cycles'].values[ind[i]]
                    cycind = np.argmax(np.abs(mycycle)) + ind[i-2]
                    #plt.plot(self.cyc['Time'].values, strain)
                    #plt.plot(self.cyc['Time'].values[ind[i-2]:ind[i]],
                        #mycycle)
                    #pdb.set_trace()
                    break
        self.cycles = cycles
        self.cycind = cycind
        if self.cycles < 0.:
            print('Failure criteria didn\'t work: DA {}%'.format(failval))
    elif self.cycletype == 'backcalc':
        gbeta = [ 0.2747 , -11.34 , 1.303 ]
        print(gbeta)
        self.DWliqbc = self.sigveff * np.exp(gbeta[0] *
                   (self.Dr / 15.) ** 2 + gbeta[1])
        self.cycles = self.cyc['Cycles'].max() #May need to be adjusted
        self.cycind = len(self.cyc.index) - 2

    else:
        self.cycles = self.cyc['Cycles'].max() #May need to be adjusted
        self.cycind = len(self.cyc.index) - 2

    self.r_u_fin = self.cyc['Ru'].values[self.cycind]
    self.total_pts = self.cyc['Cycles'].values[-1]

    # Calculate the CSR
```

```
        if self.strain:
            self.CSRcalc(strain=True)
        self.CSRcalc()


        if verbose:
            print('Test: %s, Dr=%.2f%%, CSR: %.3f, Cycles: %.1f, sigv: %.0f kPa'
                %(self.name,self.Dr, self.CSR, self.cycles, self.sigveff))
            if self.strain:
                print('Gamma (SA): {:.4f}'.format(self.gamma))

    def EQcalcs(self, **kwargs):
        '''
        This will calculate Neq_computed for the earthquake tests.
        '''
        from smooth import smooth
        # Get my gam, G and D degradation curves:
        sigmeff = (1. + 2. * 0.45) * self.sigveff / 3. # <==== Assuming a ko here.
        gam, Gratio, damping = getDegrad('DS', 0., sigmeff, N=self.cycles,
                                soil=1., freq=0.1)
        tau_avg = np.max(np.abs(self.cyc['sstress_int'].values)) * 0.65

        strain = smooth(self.cyc['sstrain_int'].values)
        stress = smooth(self.cyc['sstress_int'].values)
        runningG = (np.abs(stress - stress[0]) /
                    np.abs(strain - strain[0]))
        self.Gmax = (440 * (self.Dr / 15.) ** (2/3.) * 101.325 *
                np.sqrt(sigmeff/101.325))

        G, D, gamout = TokimatsuSeedDirect(gam, Gratio, damping, self.Gmax, tau_avg)
        Gstr = G * (1 - D ** 2 + 1j * 2 * D) #<=== Simplified

        NPTS = len(stress)
        NFFT = 2 ** nextpow2(NPTS)

        newstrain = np.fft.irfft( np.fft.rfft(stress, NFFT) / Gstr)[:NPTS]

        DissEntotalstress = np.cumsum((stress[1:] + stress[:-1]) *
                (newstrain[1:] - newstrain[:-1])) * 0.5
        dWone = 2 * np.pi * D * tau_avg ** 2 / G
        self.Neq = DissEntotalstress[self.maxind] / dWone

    def calcdW_Alt(self, **kwargs):
        '''
        This will calculate a dissipated energy in a manner more consistent
        with equivalent-linear results.
        '''
        N160 = (self.Dr / 15.) ** 2
        phi = np.sqrt(20 * N160) + 20
        ko = 1- np.sin(phi * np.pi / 180.)
        sigmeff = (1. + 2. * ko) * self.sigveff / 3. # <==== Assuming a ko here.
        gam, Gratio, damping = getDegrad('DS', 0., sigmeff, N=self.cycles,
                                soil=0, freq=0.1)
        if self.testtype == 'EQ':
            tau_avg = np.max(np.abs(self.cyc['sstress_int'].values)) * 0.65
        else:
            tau_avg = self.CSR * self.sigveff
```

```
1655
1656            self.Gmax = (440. * N160 ** (1./3.) * 101.325 *
1657                   np.sqrt(sigmeff/101.325))
1658
1659            G, D, gamout = TokimatsuSeedDirect(gam, Gratio, damping,
1660                                               self.Gmax, tau_avg)
1661            self.dWone = 2 * np.pi * D * tau_avg ** 2. / G
1662            if self.testtype == 'EQ':
1663                self.dW_corralt = self.dWone * self.Neq
1664            else:
1665                self.dW_corralt = self.dWone * self.cycles
1666            print('    G/Gmax: {:.3f}, D: {:.3f}, gam: {:.4g}, DE_alt: {}'.format(
1667                G/self.Gmax, D, gamout, self.dW_corralt ))
1668            print('Gmax: {}, tau/Gmax: {}, tau: {}, dWone: {}'.format(
1669                self.Gmax, tau_avg/self.Gmax, tau_avg, self.dWone))
1670            print('dWone/Gmax: {}, sigmeff: {}'.format(self.dWone/self.Gmax,sigmeff))
1671
1672        def DiscreteCycles(self, **kwargs):
1673            '''
1674            Modifies the cycle count so that partial cycles are given, and saved
1675            as 'CyclesD' in the dataframe.
1676            '''
1677            cycles = self.cyc['Cycles'].values
1678            time = self.cyc['Time'].values
1679            ind = np.where((cycles[:-1] != cycles[1:]))
1680            cyclesD = np.zeros(cycles.shape, dtype=float)
1681
1682            for i,val in enumerate(ind[0]):
1683                if i == 0:
1684                    in1 = 0
1685                    in2 = val + 1
1686                    prevcyc = 0.
1687                else:
1688                    in1 = ind[0][i-1] + 1
1689                    in2 = val + 1
1690                    prevcyc = cycles[in1] - 1.
1691                time2 = time[in1:in2] - time[in1]
1692                cyclesD[in1:in2] = prevcyc + time2 / time2[-1]
1693            in1 = ind[0][i-1] + 1
1694            in2 = len(time)
1695            prevcyc = cycles[in1] - 1.
1696            time2 = time[in1:in2] - time[in1]
1697            cyclesD[in1:in2] = prevcyc + time2 / time2[-1]
1698            self.cyc['CyclesD'] = cyclesD
1699            self.cyc['CycleRatio'] = cyclesD / self.cycles
1700
1701        def CSRcalc(self, **kwargs):
1702            '''
1703            Calculates the average CSR over the majority of the cyclic portion,
1704            or, for a strain controlled tests, the SA shear strain.
1705            '''
1706            strain = kwargs.get('strain', False)
1707            if self.testtype == 'sine':
1708                cycrange = kwargs.get('cycrange',75)
1709                # Get the zero crossings
1710                if strain:
1711                    stress = self.cyc['sstrain_int'].values
```

```
            else:
                stress = self.cyc['sstress_int'].values
            cycles = int(self.cyc['Cycles'].values[-1] * cycrange / 100)
#           cycles = int(self.cycind * cycrange / 100)
            # Calc the max and min in each cycle
            maxs = np.zeros([cycles,],dtype=float)
            mins = np.zeros([cycles,],dtype=float)
            for i in xrange(cycles):
                ind2 = np.where(self.cyc['Cycles'].values == i+1)
                maxs[i] = np.max(stress[ind2])
                mins[i] = np.min(stress[ind2])

            sigd = np.mean(maxs - mins) / 2
            if strain:
                self.gamma = sigd
            else:
                self.CSR = sigd / self.sigveff
                if self.strain:
                    self.CSR = self.CSR * -1.
        else:
            self.ElapsedTime = self.cyc['Time'].values[-1]
            stress = self.cyc['sstress_int'].values
            peakval = np.amax(np.abs(stress[:self.cycind]))
            self.CSR = 0.65 * peakval / self.sigveff
            ind = np.argmax(np.abs(stress))
            peaktime = self.cyc['Time'].values[ind]
            ind2 = np.where(self.cyc['dt'].values<0.1)
            dt1 = self.cyc['dt'].values[ind2[0][1]:self.cycind]
            stress1 = stress[ind2[0][1]:self.cycind]
            ind3 = np.where(dt1 > 0)
            dt2 = dt1[ind3]
            stress2 = stress1[ind3]
            self.PTratio = peaktime / self.ElapsedTime
            self.Ar = calcAriasRatio(stress2,
                                     dt2) # This dt value might not always be right
            print('dt= {} sec'.format(np.max(dt2)))

    def dissen(self,**kwargs):
        '''
        Calculates the dissipated energy from the test
        '''
        verbose = kwargs.get('verbose',True)
        stresses = self.cyc['sstress_int'].values
        strains = self.cyc['sstrain_int'].values
        cycles = self.cyc.Cycles.values

        if self.cycletype == 'backcalc':

            try:
                dissEn = np.cumsum((stresses[1:] + stresses[:-1]) *
                           (strains[1:] - strains[:-1])) * 0.5
                dissEnZS = dissEn[self.ZeroStressInd]
                mylen = int(len(self.ZeroStressInd) / 2)
                mydf = pd.DataFrame(dict(MyIndex = self.ZeroStressInd[:mylen],
                    DW=dissEnZS[:mylen]))

                fit = sm.formula.ols('MyIndex ~ DW', data=mydf).fit()
```

```
            params = fit.params

            #ind = np.where(dissEnZS >= self.DWliqbc)[0]
            #self.maxind = self.ZeroStressInd[ind[0]]
            ind0 = int(params[0] + params[1] * self.DWliqbc)
            ind = np.where(self.ZeroStressInd <= ind0)[0]
            self.maxind = self.ZeroStressInd[ind[-1]]
            if self.maxind == 0 : self.Error = True; return 1
        except:
            self.Error = True
            return 1
            #pdb.set_trace()
        maxind = self.maxind
        self.cycind = maxind
        self.Cycles = cycles[self.cycind]
        self.r_u_fin = self.cyc['Ru'].values[self.cycind]
    else:
        #maxind = self.cycind
        # In this section, I get the index of the beginning of the cycle
        # in which the sample has liquefied.
        cycstartind = np.where((cycles[:-1] != cycles[1:]))[0]
        zerostrainind = np.where(
               np.sign(strains[:-1]) != np.sign(strains[1:]))[0]
        try:
            #maxind = cycstartind[np.where(self.cycind >= cycstartind)[0][-1]]
            # Finds where the strain was last zero for DE calcs
            #maxind = zerostrainind[np.where(self.cycind>= zerostrainind)[0][-1]]
            # Finds where the stress was last zero for DE calcs
            maxind = self.ZeroStressInd[np.where(self.cycind >=
                           self.ZeroStressInd)[0][-1]]
            self.maxind = maxind
        except IndexError:
            self.Error = True
            return 1
    try:
        dissEn = np.cumsum((stresses[1:] + stresses[:-1]) *
                       (strains[1:] - strains[:-1])) * 0.5
    except ValueError:
        pdb.set_trace()
    if np.abs(np.mean(stresses)) > 1:
        ('Adjusting the internal shear stresses based on the externals!!')
        stresses = stresses + np.mean(self.cyc['sstress_ext'].values -
                       stresses)

    try:
        dissEn2 = np.zeros([len(self.cyc.index),],dtype=float)
        dissEn2[1:] = dissEn
        self.DE = dissEn[maxind]
        #dissEn2[maxind:] = self.DE
        self.cyc['DE'] = dissEn2
    except IndexError:
        self.DE = 0.
        self.DE_corr = 0.
        self.DE_b = 0.
        self.DE_m = 0.
        self.PEC = 0.
        self.DE_x = 0.
```

```
            self.DE_y = 0.
            self.DE_1 = 0.
            self.cyc['DE'] = np.zeros(len(self.cyc.index))
            self.Error = True
            return 1

        # Calculate the liquefaction-corrected dissipated energy
        #ind = np.where((stresses[:maxind-1] < 0) &
                   #(stresses[1:maxind] >= 0))
        x = self.cyc['CycleRatio'].values[self.ZeroStressInd]
        y = self.cyc['DE'].values[self.ZeroStressInd]
        alen = int(0.15 * len(x))
        if alen < 4:
            alen = int(0.25 * len(x))
        elif alen < 4:
            alen = int(0.35 * len(x))
        elif alen < 4:
            alen = int(0.55 * len(x))
        #A = np.vstack([x[:alen],np.ones([alen,])]).T
        A = np.vstack([x[:alen],np.zeros([alen,])]).T
        try:
            m, b = np.linalg.lstsq(A, y[:alen])[0]
            b = 0.
        except ValueError:
            self.DE_corr = 0.
            self.DE_b = 0.
            self.DE_m = 0.
            self.PEC = 0.
            self.DE_x = 0.
            self.DE_y = 0.
            self.DE_1 = 0.
            return
        self.DE_corr = m + b
        self.DE_m = m
        self.DE_b = b
        self.DE_x = x
        self.DE_y = y
        self.DE_1 = 1.0 / float(self.cycles) * m + b

        # Calculate the PEC (Green's Dissertation, eq. 4-15)
        Ru = self.cyc['Ru'].values
        ind = np.where(Ru > 0.65)
        try:
            self.PEC = self.cyc['DE'].values[ind[0][0]] / 0.4225
        except IndexError:
            self.PEC = -1
        if verbose:
            print('   Total DE: %.4f, Corrected DE: %.4f, DE/N: %.4f'%(
                   self.DE,self.DE_corr,self.DE/self.cycles))

    def plotter(self, **kwargs):
        '''
        Produces plots of interest.

        Keyword       Definition
        save            True [default] or False
        savetype        png or pdf, etc
```

```
        show            Should the figures be shown? True or False
        '''
        save = kwargs.get('save',True)
        savetype = kwargs.get('savetype','png')
        show = kwargs.get('show',False)

        # Time plot
        useTime = False
        if self.testtype == 'EQ': useTime = True
        if useTime:
            xcol = 'Time'
            xlabel = 'Time (s)'
        else:
            #xcol = 'CycleRatio'
            #xlabel = 'Cycle Ratio, $n/N$'
            xcol = 'CyclesD'
            xlabel = 'Cycles, $N$'
        myalt = True
        if myalt:
            fig1, (ax1, ax2, ax3, ax4) = plt.subplots(nrows=4, ncols=1, sharex=False,
                   sharey=False, num='Results of %s'%self.name, figsize=(8,11))
        else:
            fig1, (ax1, ax2, ax3) = plt.subplots(nrows=3, ncols=1, sharex=False,
                   sharey=False, num='Results of %s'%self.name, figsize=(8,8))
        if self.testtype == 'EQ':
            ax1.set_title(
               r'$D_r$={:.1f} \%, $CSR$={:.2f},'.format(self.Dr,self.CSR)+
               ' $\sigma\'_{v0}$='+'{:.0f} kPa'.format(self.sigveff))
        elif self.strain:
            ax1.set_title(
            r'$D_r$={:.1f} \%'.format(self.Dr)+
            ' $N_{liq}$'+'={:.1f},'.format(self.cycles)+' $\sigma\'_{v0}$='+
            '{:.0f} kPa'.format(self.sigveff))
        else:
            ax1.set_title(
            r'$D_r$={:.1f} \%, $CSR$={:.2f},'.format(self.Dr,self.CSR)+
            ' $N_{liq}$'+'={:.1f},'.format(self.cycles)+' $\sigma\'_{v0}$='+
            '{:.0f} kPa'.format(self.sigveff))

        ax1.plot(self.cyc[xcol], self.cyc.nstress, label='Normal Stress')
        ax1.plot(self.cyc[xcol], self.cyc.sstress_int,
               label='Internal Shear Stress')
        ax1.plot(self.cyc[xcol].values[self.cycind],0.,'or')
        ax1.plot(self.cyc[xcol].values[self.maxind],0.,'og')
        ax1.yaxis.set_major_locator(MaxNLocator(nbins=4))
        ax1.legend(loc='best')
        ax1.set_ylabel('Stress, $\\tau$ (kPa)')
        ax1.set_xlabel(xlabel)
        ax1.grid(True)

        #plt.subplot(312)
        ax2.plot(self.cyc[xcol], self.cyc.nstrain_int*100.)
        ax2.set_ylabel(r'Normal Strain, $\varepsilon_v$ (\%)')
        ax2.set_xlabel(xlabel)
        ax2.yaxis.set_major_locator(MaxNLocator(nbins=4)) # added
        ax2.grid(True)
```

```
        if myalt:
            ax3.plot(self.cyc[xcol], self.cyc.Ru,
               )
            ax3.plot(self.cyc[xcol].values[self.ZeroStressInd],
                   self.cyc['Ru'].values[self.ZeroStressInd], 'ws')
            ax3.plot(self.cyc[xcol].values[self.cycind],
                    self.cyc['Ru'].values[self.cycind],
                       'or')
            ax3.plot(self.cyc[xcol].values[self.maxind],
                   self.cyc['Ru'].values[self.maxind],
                      'og')
            [xmin, xmax, ymin, ymax] = ax3.axis()
            ax3.plot([xmin, xmax], [1., 1.], '--', lw=0.5, color='0.5')
            ax3.yaxis.set_major_locator(MaxNLocator(nbins=4))
            ax3.axis([xmin, xmax, 0, 1.1])
            ax3.set_ylabel(r'$r_u$')
            ax3.set_xlabel(xlabel)
            #ax3.autoscale(True)
            ax3.grid(True)


            ax4.plot(self.cyc.sstrain_ext * 100., self.cyc.sstress_ext,
                label=r'Shear Strain, $\gamma$')
            ax4.plot(self.cyc['sstrain_ext'].values[self.cycind] * 100.,
                    self.cyc['sstress_ext'].values[self.cycind],
                       'or')
            ax4.plot(self.cyc['sstrain_ext'].values[self.maxind] * 100.,
                    self.cyc['sstress_ext'].values[self.maxind],
                       'og')
            ax4.yaxis.set_major_locator(MaxNLocator(nbins=4))
            ax4.set_ylabel(r'Shear Stress, $\tau$ (kPa)')
            ax4.set_xlabel(r'Shear Strain, $\gamma$ (\%)')
            ax4.autoscale(True)
            ax4.grid(True)

        else:
            ax3.plot(self.cyc.sstrain_ext, self.cyc.sstress_ext,
                label=r'Shear Strain, $\gamma$')
            ax3.plot(self.cyc['sstrain_ext'].values[self.cycind],
                    self.cyc['sstress_ext'].values[self.cycind],
                       'or')
            ax3.plot(self.cyc['sstrain_ext'].values[self.maxind],
                    self.cyc['sstress_ext'].values[self.maxind],
                       'og')
            ax3.yaxis.set_major_locator(MaxNLocator(nbins=4))
            ax3.set_ylabel(r'Shear Stress, $\tau$ (kPa)')
            ax3.set_xlabel(r'Shear Strain')
            ax3.autoscale(True)
            ax3.grid(True)

        fig1.subplots_adjust(hspace=0.47, left=0.16)
        #plt.show()
        #pdb.set_trace()

        fig2 = plt.figure('Shear-Normal strains')
        plt.plot(self.cyc.nstress,self.cyc.sstress_int)
        plt.xlabel('Normal Stress (kPa)')
```

```
        plt.ylabel('Shear Stress (kPa)')
        plt.autoscale()
        plt.grid(True)

        # Dissipated Energy
        fig3, (ax13, ax23, ax33) = plt.subplots(nrows=3, ncols=1,
               sharex=False, sharey=False,
               num='Dissipated Energy-%s'%self.name)
        fig3.suptitle(r'$\Delta W$ = %.3f, $\Delta W_{corr}$= %.3f, PEC= %.3f'
                  %(self.DE,self.DE_corr,self.PEC))

        ax13.plot(self.cyc.Time, self.cyc.DE)
        ax13.set_ylabel(r'$\Delta W$')
        ax13.set_xlabel('Time (sec)')
        ax13.grid(True)

        #plt.subplot(312)
        ax23.plot(self.DE_x,self.DE_y,'o')
        ax23.set_xlabel(r'$\frac{N}{N_{liq}}$')
        ax23.set_ylabel(r'$\Delta W$')
        ax23.plot(self.DE_x,self.DE_m*self.DE_x+self.DE_b)
        ax23.plot(self.DE_x, self.DE_x * self.cycles * self.dWone,'sw')
        ax23.grid(True)

        #plt.subplot(313)
        DEsqr = np.sqrt(self.cyc['DE'].values)
        Ru = self.cyc['Ru'].values
        ax33.plot(DEsqr,Ru,'o')
        ax33.set_xlabel(r'$\sqrt{\Delta W}$')
        ax33.set_ylabel(r'$r_u$')
        ax33.plot([0,np.sqrt(self.PEC)],[0,1.],'-')
        ax33.grid(True)


        plt.figure('DE-Ru')
        plt.plot(self.cyc['DE'].values, Ru,'o')
        plt.xlabel(r'$\Delta W$')
        plt.ylabel(r'$r_u$')
        plt.grid(True)
        #plt.show()

        if show:
            plt.show()
        if save:
            import matplotlib
            figures=[manager.canvas.figure for manager in
              matplotlib._pylab_helpers.Gcf.get_all_fig_managers()]
            for i, figure in enumerate(figures):
                figure.savefig(self.path +
                               self.name +'-'+str(i+1)+'.'+savetype)

        plt.close('all')
        DEplot = False
        if DEplot == True:
            self.cyc.plot(x='sstrain_ext',y='sstress_ext',
                     label=r'Shear Strain, $\gamma$')
            plt.ylabel(r'Shear Stress, $\tau$ (kPa)')
```

```
            plt.xlabel(r'Shear Strain, $\gamma$')
            plt.show()

        if self.extrafigs:
            plt.close('all')
            plt.figure('s-s')
            self.cyc.plot(x='sstrain_ext',y='sstress_ext',
                     label=r'Shear Strain, $\gamma$')
            plt.ylabel(r'Shear Stress, $\tau$ kPa')
            plt.title(r'$D_r$=%.1f %%, $CSR$= %.2f, $N$=%.1f, $\sigma_v$\'=%.0f (kPa)'
                   %(self.Dr,self.CSR,self.cycles,self.sigveff))
            plt.savefig('./ExtraFigs/'+self.name +'-'+'tau-gam'+'.png')

            plt.figure('PP-N')
            EquivPorePress = (self.cyc['nstress'].values[0] -
                          self.cyc['nstress'].values)
            plt.plot(self.cyc['CyclesD'].values,
                    EquivPorePress)
            plt.xlabel(r'Cycles')
            plt.ylabel(r'Equivalent Pore Pressure (kPa)')
            plt.title(r'$D_r$=%.1f %%, $CSR$= %.2f, $N$=%.1f, $\sigma_v$\'=%.0f (kPa)'
                   %(self.Dr,self.CSR,self.cycles,self.sigveff))
            plt.savefig('./ExtraFigs/'+self.name +'-'+'PP-N'+'.png')

            plt.figure('Ru-DE')
            stresses = self.cyc['sstress_int'].values
            ind = np.where((stresses[:-1] < 0) &
                   (stresses[1:] >= 0))
            plt.plot(self.cyc['DE'].values[ind], Ru[ind], 'o')
            plt.xlabel(r'Dissipated Energy')
            plt.ylabel(r'Pore Pressure Ratio')
            plt.title(r'$D_r$=%.1f %%, $CSR$= %.2f, $N$=%.1f, $\sigma_v$\'=%.0f (kPa)'
                   %(self.Dr,self.CSR,self.cycles,self.sigveff))
            plt.savefig('./ExtraFigs/'+self.name +'-'+'Ru-DE'+'.png')
            plt.close('all')

        if self.PaperFigs:
            color = '0.35'
            lw=1.2
            alpha = 0.9
            fig1, (ax1, ax2) = plt.subplots(nrows=1, ncols=2,
                   sharex=False, sharey=True,
                   num='PaperFig1-%s'%self.name, figsize=(9,5))
            ax1.plot(self.cyc.sstrain_int * 100., self.cyc.sstress_int, lw=lw,
                   color='0.15', alpha=alpha)
            ax1.set_xlabel(r'Shear Strain, $\gamma$ (\%)')
            ax1.set_ylabel('Shear Stress (kPa)')
            ax1.xaxis.set_major_locator(MaxNLocator(nbins=4, prune='both')) # added
            ax1.grid(True)

            ax2.plot(self.cyc.nstress,self.cyc.sstress_int, lw=lw, color=color,
                   alpha=alpha)
            ax2.set_xlabel('Vertical Stress (kPa)')
            ax2.grid(True)
            fig1.subplots_adjust(bottom=0.12, wspace=0)

            if os.name == 'posix':
```

```
2111                sys.path.append('/media/Storage/Documents/00-VtResearch/Thesis/Markdown/Figures/SeismicCompression')
2112            else:
2113                sys.path.append('/../../Documents/00-VtResearch/Thesis/Markdown/Figures/SeismicCompression')
2114
2115            from SCplots import dimlines
2116
2117            fig2, ax3 = plt.subplots(num='PaperFig2-%s'%self.name,
2118                   figsize=(10, 8.33))
2119            ax3.plot(self.DE_x,self.DE_y,'o', color=color, label='Data')
2120            ax3.plot(1.,self.DE*1.1,'^k',
2121                   label=r'$\Delta W_{\mathrm{Effect.}}$')
2122            ax3.set_xlabel(r'Cycles / Cycles to Liquefaction, $N/N_{liq}$')
2123            ax3.set_ylabel(r'$\Delta W$ $(kPa)$')
2124            ax3.plot(self.DE_x,self.DE_m*self.DE_x+self.DE_b, '--', lw=lw,
2125                   color='0.2', alpha=0.7)
2126            ax3.plot(1., self.DE_m, '*k', ms=6, label=r'$\Delta W_{\mathrm{Total}}$')
2127            [xmin, xmax, ymin, ymax] = ax3.axis()
2128            ax3.plot([1.0,1.0],[0,ymax * 1.2], '-', color='0.5', lw=0.7,
2129                   alpha=0.7)
2130            dimlines(ax3, [1., self.DE_m], [1.,self.DE*1.1],
2131                '$\Delta W_{\mathrm{Effect.}} - \Delta W_{\mathrm{Total}}$', 0.5,
2132                side='left', textscale=0.5)
2133            ax3.axis([0, 1.1, 0, ymax])
2134            ax3.legend(loc='upper left', numpoints=1)
2135            ax3.grid(True)
2136            fig2.subplots_adjust(bottom=0.15)
2137            #plt.show()
2138            #pdb.set_trace()
2139            fig1.savefig('./Sine/PaperFigs/{}-1.pdf'.format(self.name), dpi=100)
2140            fig1.savefig('./Sine/PaperFigs/{}-1.png'.format(self.name), dpi=100)
2141            fig2.savefig('./Sine/PaperFigs/{}-2.pdf'.format(self.name), dpi=100)
2142            fig2.savefig('./Sine/PaperFigs/{}-2.png'.format(self.name), dpi=100)
2143            plt.close('all')
2144
2145    def loadcorrection(self, **kwargs):
2146        '''
2147        Uses the water test files to correct the loads.
2148        '''
2149        loadcorr_type = kwargs.get('type', self.loadcorr_type)
2150        friction = HDF5saver(testtype='water')
2151 #       wtable = friction.read()
2152        if self.waterfile == None:
2153            dates = friction.summtable.read(field='Date_of_Test')
2154            ind = np.argmin(np.abs(dates-self.date_of_test))
2155            self.WaterTestName = friction.summtable.read(ind,field='File')[0]
2156        if loadcorr_type == 'old':
2157            from scipy.interpolate import interp1d
2158            try:
2159                posm = friction.summtable.read_coordinates([ind],field='posm')[0]
2160                posb = friction.summtable.read_coordinates([ind],field='posb')[0]
2161                negm = friction.summtable.read_coordinates([ind],field='negm')[0]
2162                negb = friction.summtable.read_coordinates([ind],field='negb')[0]
2163            except AttributeError:
2164                posm = friction.summtable.readCoordinates([ind],field='posm')[0]
2165                posb = friction.summtable.readCoordinates([ind],field='posb')[0]
2166                negm = friction.summtable.readCoordinates([ind],field='negm')[0]
2167                negb = friction.summtable.readCoordinates([ind],field='negb')[0]
```

```
2168
2169
2170            intshear = self.cyc['Internal Shear LVDT'].values
2171            intshear1 = intshear[np.isfinite(intshear)]
2172            dispx = np.array([np.min(intshear1),
2173                              -0.0001,0.0001,
2174                              np.max(intshear1)])
2175            loady = np.ravel(np.array([negm*dispx[0]+negb,
2176                              negm*dispx[1]+negb,
2177                                posm*dispx[2]+posb,
2178                              posm*dispx[3]+posb]))
2179            loadcorrf = interp1d(dispx,loady,kind='linear',bounds_error=True)
2180        elif loadcorr_type == 'bilinear':
2181            try:
2182                Gmax = friction.summtable.read_coordinates([ind],field='Gmax')[0]
2183                gamr = friction.summtable.read_coordinates([ind],field='gamr')[0]
2184                G2 = friction.summtable.read_coordinates([ind],field='G2')[0]
2185            except AttributeError:
2186                Gmax = friction.summtable.readCoordinates([ind],field='Gmax')[0]
2187                gamr = friction.summtable.readCoordinates([ind],field='gamr')[0]
2188                G2 = friction.summtable.readCoordinates([ind],field='G2')[0]
2189            loadcorrf = bilinear(Gmax, G2, gamr)
2190        elif loadcorr_type == 'hyperbolic':
2191            try:
2192                Gmax = friction.summtable.read_coordinates([ind],field='Gmax')[0]
2193                gamr = friction.summtable.read_coordinates([ind],field='gamr')[0]
2194                a = friction.summtable.read_coordinates([ind],field='hypa')[0]
2195                b = friction.summtable.read_coordinates([ind],field='hypb')[0]
2196            except AttributeError:
2197                Gmax = friction.summtable.readCoordinates([ind],field='Gmax')[0]
2198                gamr = friction.summtable.readCoordinates([ind],field='gamr')[0]
2199                a = friction.summtable.readCoordinates([ind],field='hypa')[0]
2200                b = friction.summtable.readCoordinates([ind],field='hypb')[0]
2201            taumax = gamr * Gmax
2202            loadcorrf = hyperbolic(Gmax, taumax, a=a, b=b)
2203        friction.close()
2204        return loadcorrf
2205
2206    def wu_etal_liq_crit(self,**kwargs):
2207        '''
2208        Uses the liquefaction criteria from Wu et al. 2004.
2209        Initial liquefaction is defined as the occurrence of 6% Single
2210          Amplitude or 6% Double Amplitude shear strain.
2211
2212        Wu, J., Kammerer, A. M., Riemer, M. F., Seed, R. B., and
2213          Pestana, J. M. (2004). "Laboratory study of liquefaction triggering
2214          criteria." 13th World Conference on Earthquake Engineering,
2215          Vancouver, BC.
2216        '''
2217        strain = self.cyc['sstrain_int'].values
2218        stress = self.cyc['sstress_int'].values
2219        ind = np.where((stress[:-1] >= 0) & (stress[1:]< 0))[0]
2220        maxs = np.zeros([len(ind),],dtype=float)
2221        mins = np.zeros([len(ind),],dtype=float)
2222        for i, val in enumerate(ind):
2223            if i == 0:
2224                if ind[i] == 0:
```

```
                    continue
                else:
                    maxs[i] = np.max(strain[0:ind[i]])
                    mins[i] = np.min(strain[0:ind[i]])
            else:
                maxs[i] = np.max(strain[ind[i-1]:ind[i]])
                mins[i] = np.min(strain[ind[i-1]:ind[i]])

            if (maxs[i] or np.abs(mins[i])) >= 0.06:
                indout = ind[i]
                return self.cyc['Cycles'].values[indout], indout
            elif np.abs(maxs[i] - mins[i]) >= 0.06:
                indout = ind[i]
                return self.cyc['Cycles'].values[indout], indout

        print('Did not liquefy according to the Wu et al. method')
        return -1, -1

    def mod_liq_crit(self, **kwargs):
        '''
        Determines the point of liquefaction by the degradation of the
        shear modulus. Still working on the details...
        '''
        from smooth import smooth

#       G = self.cyc['G'].values
        strain = smooth(self.cyc['sstrain_int'].values)
        #[self.cyc['sstrain_int'].values!=0]
#       strain = strain - strain[0] # Should I be zeroing this out?
        stress = smooth(self.cyc['sstress_int'].values)
        ind = np.where((stress[:-1] >= 0) & (stress[1:]< 0))[0]
        maxs = np.zeros([len(ind),],dtype=float)
        mins = np.zeros([len(ind),],dtype=float)
        maxind = np.zeros([len(ind),], dtype=int)
        minind = np.zeros([len(ind),], dtype=int)
        Gpos = np.zeros([len(ind),], dtype=float)
        Gneg = np.zeros([len(ind),], dtype=int)
        for i, val in enumerate(ind):
            if i == 0:
                maxs[i] = np.max(stress[0:ind[i]])
                mins[i] = np.min(stress[0:ind[i]])
                maxind[i] = np.argmax(stress[0:ind[i]])
                minind[i] = np.argmin(stress[0:ind[i]])
            else:
                maxs[i] = np.max(stress[ind[i-1]:ind[i]])
                mins[i] = np.min(stress[ind[i-1]:ind[i]])
                maxind[i] = ind[i-1] + np.argmax(stress[ind[i-1]:ind[i]])
                minind[i] = ind[i-1] + np.argmin(stress[ind[i-1]:ind[i]])
        avgstress = (np.abs(stress[maxind]) + np.abs(stress[minind])) / 2.
        avgstrain = (strain[maxind] + strain[minind]) / 2.

        G = (np.abs(stress[maxind] - avgstress) /
                    np.abs(strain[maxind] - avgstrain))

        pdb.set_trace()
        Gpos = stress[maxind] / strain[maxind]
        Gneg = stress[minind] / strain[minind]
```

```
        plt.figure('G')
        plt.plot(strain, stress)
        plt.plot(strain[maxind], stress[maxind], 'ro')
        plt.plot(strain[minind], stress[minind], 'ro')
        plt.plot(strain[maxind[np.where(np.abs(Gpos)<np.max(Gpos)/100)[0]]],
               stress[maxind[np.where(np.abs(Gpos)<np.max(Gpos)/100)]],'g+')
        plt.figure('Other')
        plt.plot(stress)
        plt.plot(maxind[np.where(np.abs(Gneg)<np.max(np.abs(Gneg)/100))[0]],
            stress[maxind[np.where(np.abs(Gneg)<np.max(np.abs(Gneg)/100))]],
            'ro')
        plt.plot(maxind[np.where(np.abs(Gpos)<np.max(Gpos)/100)[0]],
               stress[maxind[np.where(np.abs(Gpos)<np.max(Gpos)/100)]],'ro')
        plt.show()
        pdb.set_trace()

    def fileout(self, **kwargs):
        '''
        Writes the specified time history to file.
        '''
        motiontype = kwargs.get('motiontype', 'displacement')
        sensor = kwargs.get('sensor', 'external')
        from GCTSInput import FileMaker
        if motiontype == 'displacement':
            if sensor in ['internal', 'int', 'i']:
                FileMaker(self.cyc['Internal Shear LVDT'].values,
                       self.cyc['Time'].values, self.name,
                       sensor=sensor, motiontype=motiontype)
            elif sensor in ['external', 'ext', 'e']:
                FileMaker(self.cyc['Shear Displacement'].values,
                       self.cyc['Time'].values, self.name,
                       sensor=sensor, motiontype=motiontype)
            else:
                print('I cannot understand "{}" sensor'.format(motiontype))
        else:
            print('I cannot understand "{}" motiontype'.format(motiontype))
        pdb.set_trace()

def saverpickle(testobj, **kwargs):
    fid1=open('%s.dp1'%(testobj.name),'w')
    p.dump(testobj, fid1)
    fid1.close()

class WaterTest(object):
    '''
    This class parses the GCTS test file for water only (or friction) tests.
    It returns data to be used to correct the data from regular CV-CSS tests.
    '''

    def __init__(self,infile,**kwargs):
        self.csvimport(infile)
        if kwargs.get('surcharge',False):
            self.calcs_sur(**kwargs)
        else:
            self.calcs1(**kwargs)

    def csvimport(self,infile):
```

```
2339         datafile = csv.reader(open(infile,'rU'))
2340         rowcount = -1
2341         rows = []
2342         self.units = []
2343         for row in datafile:
2344             rowcount += 1
2345 
2346             if 'Starting Date: ' in row:
2347                 testdate = row[1]
2348             if 'Starting Time: ' in row:
2349                 testtime = row[1]
2350 
2351             if len(row) > 17:
2352                 if row[0] == 'Time':
2353                     rows.append(rowcount)
2354                     if len(rows) == 1:
2355                         header = rowcount
2356                         self.headerlabel = row
2357                     if len(rows) == 2:
2358                         nrows = rows[-1] - rows[0] - 5
2359                         break
2360                 if row[0] == 'sec':
2361                     if len(self.units) == 0:
2362                         self.units = row
2363 
2364         df = pd.read_csv(infile,names=self.headerlabel,
2365                          skiprows=header+3)#,
2366 #                         nrows=nrows)
2367         df = df.convert_objects(convert_numeric=True)
2368         select = df.apply(lambda r : any([isinstance(e, basestring)
2369                           for e in r ]),axis=1)
2370         df = df[~select]
2371         df = df.dropna()
2372         self.df = df
2373         self.filename = infile
2374         self.name = os.path.splitext(self.filename)[0]
2375         self.date_of_test = datetimeconvert(testdate,testtime)
2376 
2377     def calcs1(self,**kwargs):
2378         '''
2379         Performs the calculations for the water test.
2380         '''
2381         save = kwargs.get('save',True)
2382         savetype = kwargs.get('savetype','png')
2383         show = kwargs.get('show',False)
2384         movie = kwargs.get('movie', True)
2385 
2386 
2387         print(self.name)
2388         self.df['Disp'] = (self.df['Internal Shear LVDT'] -
2389                            self.df['Internal Shear LVDT'][0])
2390         self.df['Load'] = (self.df['Internal Shear Load'])
2391         Load = self.df['Load'].values
2392         Disp = self.df['Disp'].values
2393         Time = self.df['Time'].values
2394         #Find indices of zero crossing
2395         ind = np.where((Disp[:-1] < 0) & (Disp[1:] >= 0))
```

```
        Vel = np.zeros(Disp.shape,dtype=float)
        Vel[1:] = np.diff(Disp) / np.diff(Time)

        Lmax = np.zeros(ind[0].shape,dtype=float)
        Lmin = np.zeros(ind[0].shape,dtype=float)
        Dmax = np.zeros(ind[0].shape,dtype=float)
        Dmin = np.zeros(ind[0].shape,dtype=float)
        Maxind = np.zeros(ind[0].shape,dtype=int)
        Minind = np.zeros(ind[0].shape,dtype=int)
        Dx = np.max(Disp) - np.min(Disp)
        Dy = np.max(Load) - np.min(Load)
        maxtype = kwargs.get('maxtype', 'sinefit')
        for i in xrange(len(ind[0])):
            if maxtype == 'notsinefit':
                from scipy.optimize import curve_fit
                def mysine(x, A, w, c):
                    return A * np.sin(w*x) + c
                if i == 0:
                    X = Time[0:ind[0][i]]
                    Y = Disp[0:ind[0][i]]
                else:
                    X = Time[ind[0][i-1]:ind[0][i]]
                    Y = Disp[ind[0][i-1]:ind[0][i]]
                if len(X)<2:
                    Dmax[i] = 0.
                    Dmin[i] = 0.
                    Lmax[i] = 0.
                    Lmin[i] = 0.
                else:
                    A = np.max(Y)
                    w = 2. * np.pi / (X[-1] - X[0])
                    c = Time[ind[0][i-1]]
                    popt, pov = curve_fit(mysine, X, Y, p0=[A, w, c])
                    Maxind[i] = ind[0][i-1] + np.argmin(np.abs(Y - popt[0]))
                    Minind[i] = ind[0][i-1] + np.argmin(np.abs(Y + popt[0]))
                    Dmax[i] = Disp[Maxind[i]]
                    Dmin[i] = Disp[Minind[i]]
                    Lmax[i] = Load[Maxind[i]]
                    Lmin[i] = Load[Minind[i]]
            else:
                if i == 0:
                    maxind = np.argmax(Disp[0:ind[0][i]])
                    minind = np.argmin(Disp[0:ind[0][i]])
                    Maxind[i] = maxind
                    Minind[i] = minind
                else:
                    maxind = np.argmax(Disp[ind[0][i-1]:ind[0][i]])
                    minind = np.argmin(Disp[ind[0][i-1]:ind[0][i]])
                    Maxind[i] = maxind + ind[0][i-1]
                    Minind[i] = minind + ind[0][i-1]
                Dmax[i] = Disp[Maxind[i]]
                Dmin[i] = Disp[Minind[i]]
                Lmax[i] = Load[Maxind[i]]
                Lmin[i] = Load[Minind[i]]

        poscoeff = np.polyfit(Dmax,Lmax,1)
        negcoeff = np.polyfit(Dmin,Lmin,1)
```

```
        print('Pos Coeffs: {}'.format(poscoeff))
        print('Neg Coeffs: {}'.format(negcoeff))
        self.poscoeff = poscoeff
        self.negcoeff = negcoeff
        plt.plot(Time,Disp)
        plt.plot(Time[Maxind],Dmax,'o')
        plt.plot(Time[Minind],Dmin,'o')
        plt.figure('{}'.format(self.name))
        plt.plot(Dmax,Lmax,'o')
        plt.plot(Dmin,Lmin,'o')
        xpos = np.array([0,Dmax[-1]])
        ypos = poscoeff[1] + poscoeff[0]*xpos
        xneg = np.array([0,Dmin[-1]])
        yneg = negcoeff[1] + negcoeff[0]*xneg
        plt.plot(xpos,ypos)
        plt.plot(xneg,yneg)
        plt.xlabel('Displacement (mm)')
        plt.ylabel('Load (kN)')
        titletext = (
            'Positive: m = {}, b = {}, Negative: m = {}, b = {}'.format(
            round(poscoeff[0],4),
            round(poscoeff[1],4),
            round(negcoeff[0],4),
            round(negcoeff[1],4)))
        plt.title(titletext)
        if save:
            plt.savefig(self.name + '-1.' + savetype)
        if show:
            plt.show()
        try:
            Gmax = np.polyfit(Disp[:200],Load[:200],1)[0]
        except TypeError:
            pdb.set_trace()
        pdb.set_trace()
        m = np.mean([poscoeff[0], negcoeff[0]])
#       Gmax = 0.04 # May need to change this
        gamr = 0.5 * (Dy - m * Dx) / (Gmax - m)
#       offsety = np.mean([poscoeff[1], negcoeff[1]])
        offsety = (np.max(Load) + np.min(Load)) / 2.
    taumax = (np.max(Load) - np.min(Load)) / 2.
    a = 0.2
#       b = 0.16
        if a == 0:
            b = 0.
        else:
            b = -np.log((Gmax ** 2 + m * (Gmax + 1)) / m * a )
#           b = -np.log((Gmax - 2 * m) / (a * m))
        self.hypa = a
        self.hypb = b
    hyp = hyperbolic(Gmax, taumax,
         a=a, b=b,
         voffset=offsety)
    tauhyp = hyp(Disp)
        bil = bilinear(Gmax, m, gamr, voffset=offsety)
        self.Gmax = Gmax
        self.G2 = m
        self.gamr = gamr
```

```
2510            self.voffset = offsety
2511    #        from smooth import smooth
2512    #        Dispsm = smooth(Disp, window_len=15)
2513            taubil = bil(Disp)
2514            plt.figure('2')
2515            plt.plot(Disp, Load, 'ro')
2516            plt.plot(Disp, taubil, 'b')
2517            plt.plot(Disp, tauhyp, 'g')
2518            plt.xlabel('Displacement (mm)')
2519            plt.ylabel('Load (kN)')
2520            plt.title(
2521            'Gmax = {:.4f}, G2 = {:.4f}, $\gamma_r$ = {:.3f}, V.Off={:.4f}'.format(
2522                      Gmax,
2523                      m,
2524                      gamr,
2525                      offsety))
2526            if save:
2527                plt.savefig(self.name + '-2.' + savetype)
2528            plt.show()
2529            pdb.set_trace()
2530
2531
2532            if movie:
2533                moviemaker([Time,Disp,[Load,taubil, tauhyp]], xlabel='Displacement (mm)',
2534                           ylabel='Load (kN)',
2535                           num_y=3,
2536                             moviename=self.name[:-4])
2537
2538            plt.close('all')
2539
2540
2541        def calcs_sur(self,**kwargs):
2542            '''
2543            Does the calcs for an air and surcharge friction test.
2544            '''
2545            save = kwargs.get('save',True)
2546            savetype = kwargs.get('savetype','png')
2547            show = kwargs.get('show',False)
2548
2549            Time = self.df['Time'].values
2550            dt = np.diff(Time)
2551            ind = np.where(dt!=0)
2552            dt = dt[ind]
2553            Load = self.df['Internal Shear Load'].values[ind]
2554            Disp = self.df['Shear Displacement'].values[ind]
2555            self.PDE = 0.5 * np.sum((Load[1:] + Load[:-1]) *
2556                     (Disp[1:] - Disp[:-1]))
2557            Time = Time[ind]
2558
2559            surcharge = kwargs.get('load')
2560            if surcharge == None:
2561                surcharge = float(raw_input('Give me the surcharge load (in kg)!'))
2562            self.surcharge = surcharge
2563
2564            Acc = np.zeros(Disp.shape,dtype=float)
2565            ind4 = np.where((Disp[:-1]<=0) & (Disp[1:]>0))
2566            freq = 1. / np.average(np.diff(Time[ind4]))
```

```
        w = 2 * np.pi * freq
        Acc = -Disp*0.001 * w ** 2

        inertial_load = Acc * surcharge / 1000 # units of kN
        Load_corr = Load - inertial_load
        self.PDE_corr = 0.5 * np.sum((Load_corr[1:] + Load_corr[:-1]) *
                 (Disp[1:] - Disp[:-1]))
        DissEn = np.zeros(Time.shape,dtype=float)
        DissEn[1:] = 0.5 * ((Load_corr[1:] + Load_corr[:-1]) *
                 (Disp[1:] - Disp[:-1]))
        plt.figure('DE')
        plt.plot(Time,DissEn,label='{} kN'.format(surcharge))
        plt.legend(loc='upper left')

        #Find indices of zero crossing
        ind = np.where((Disp[:-1] < 0) & (Disp[1:] >= 0))

        # Get the max and mins for each set of indices
        Lmax = np.zeros(ind[0].shape,dtype=float)
        Lmin = np.zeros(ind[0].shape,dtype=float)
        Dmax = np.zeros(ind[0].shape,dtype=float)
        Dmin = np.zeros(ind[0].shape,dtype=float)
        Maxind = np.zeros(ind[0].shape,dtype=int)
        Minind = np.zeros(ind[0].shape,dtype=int)
        for i in xrange(len(ind[0])):

            if i == 0:
                if ind[0][i] == 0:
                    Maxind[i] = 0
                    Minind[i] = 0
                    continue
                else:
                    maxind = np.argmax(Disp[0:ind[0][i]])
                    minind = np.argmin(Disp[0:ind[0][i]])
                    Maxind[i] = maxind
                    Minind[i] = minind
            else:
                maxind = np.argmax(Disp[ind[0][i-1]:ind[0][i]])
                minind = np.argmin(Disp[ind[0][i-1]:ind[0][i]])
                Maxind[i] = maxind + ind[0][i-1]
                Minind[i] = minind + ind[0][i-1]
            Dmax[i] = Disp[Maxind[i]]
            Dmin[i] = Disp[Minind[i]]
            Lmax[i] = Load_corr[Maxind[i]]
            Lmin[i] = Load_corr[Minind[i]]


        poscoeff = np.polyfit(Dmax,Lmax,1)
        negcoeff = np.polyfit(Dmin,Lmin,1)
        self.poscoeff = poscoeff
        self.negcoeff = negcoeff
        plt.figure('{}'.format(self.name))
        plt.plot(Dmax,Lmax,'o')
        plt.plot(Dmin,Lmin,'o')
        xpos = np.array([0,Dmax[-1]])
        ypos = poscoeff[1] + poscoeff[0]*xpos
        xneg = np.array([0,Dmin[-1]])
```

```
        yneg = negcoeff[1] + negcoeff[0]*xneg
        plt.plot(xpos,ypos)
        plt.plot(xneg,yneg)
        plt.xlabel('Displacement (mm)')
        plt.ylabel('Load (kN)')
        titletext = (
            'Positive: m = {}, b = {}, Negative: m = {}, b = {}'.format(
            round(poscoeff[0],4),
            round(poscoeff[1],4),
            round(negcoeff[0],4),
            round(negcoeff[1],4)))
        plt.title(titletext)
        if save:
            plt.savefig(self.name + '.' + savetype)
        if show:
            plt.show()
        plt.close('{}'.format(self.name))

def runwater(*args, **kwargs):
    '''
    This is a convenience function that pushes test files through
    the WaterTest and the results into HDF5saver.
    '''
    if os.name == 'posix':
        defaultDir=(u'/media/Storage/Documents/Python'+
                        '/GCTS/ConstantVolume/Friction')
    else:
        defaultDir=''
    infilelist=[]

    if len(args) > 0:
        argsin = list(args)
    else:
        argsin=sys.argv[1:]

    filters = 'CSV (*.csv)' # for qtfiledialog
#    filters = 'CSV|*.csv' # for wxfiledialog
    if argsin==[]:
        filelist=fd.fileopen(message='Choose a gcts water output...',
                        filters=filters,
                        defaultDir=defaultDir)
        for file in filelist:
            infilelist.append(file)
        del filelist
    elif len(argsin)>0:
        if argsin[0] in ['all','--all','-a','a']:
            infilelist=glob.glob('*.csv')
        else:
            for file in argsin:
                infilelist.append(file)
    del argsin
    if kwargs.get('surcharge',False):
        h5table = HDF5saver(testtype='surcharge')
        surcharges = []
        for i,infile in enumerate(infilelist):
            print(infile)
            surcharges.append(
```

```
2681                float(raw_input('Give me the surcharge load (in kg)!: ')))
2682            for i,infile in enumerate(infilelist):
2683                result = WaterTest(infile, load=surcharges[i], **kwargs)
2684                h5table.input(result)
2685
2686        else:
2687            h5table = HDF5saver(testtype='water')#,filename="WaterSummary.h5")
2688            for infile in infilelist:
2689                result = WaterTest(infile, **kwargs)
2690                h5table.input(result)
2691        h5table.close()
2692
2693    def runsoil(*args,**kwargs):
2694        '''
2695        This is a convenience function that pushes test files through
2696        the SoilTest and the results into HDF5saver.
2697        '''
2698        strain = kwargs.get('strain', False)
2699        PaperFigs = kwargs.get('PaperFigs')
2700        if os.name == 'posix':
2701            if strain:
2702                defaultDir=(u'/media/Storage/Documents/Python'+
2703                                '/GCTS/ConstantVolume/StrainControlled')
2704            else:
2705                defaultDir=(u'/media/Storage/Documents/Python'+
2706                                '/GCTS/ConstantVolume/Sine')
2707        else:
2708            defaultDir=''
2709        h5 = kwargs.get('h5file','default')
2710        extrafigs = kwargs.get('extrafigs', False)
2711        #cycletype = kwargs.get('cycletype', 'strainSA')
2712        cycletype = kwargs.get('cycletype', 'ru')
2713        strainfailval = kwargs.get('strainfailval', '0375')
2714        rufailval = kwargs.get('rufailval', '98')
2715        loadcorr = kwargs.get('loadcorr', False)
2716        loadcorr_type = kwargs.get('loadcorr_type', 'bilinear')
2717        infilelist=[]
2718        if len(args) > 0:
2719            argsin = args
2720        else:
2721            argsin=sys.argv[1:]
2722        filters = 'CSV (*.csv)' # for qtfiledialog
2723    #   filters = 'CSV|*.csv' # for wxfiledialog
2724        if argsin==[]:
2725            filelist=fd.fileopen(message='Choose a gcts output...',
2726                                 filters=filters,
2727                                 defaultDir=defaultDir)
2728            for file in filelist:
2729                infilelist.append(file)
2730            del filelist
2731        elif len(argsin)>0:
2732            if argsin[0] in ['all','--all','-a','a']:
2733                if strain:
2734                    infilelist=glob.glob('./StrainControlled/*.csv')
2735                else:
2736                    infilelist=glob.glob('./Sine/*.csv')
2737            else:
```

```
            for file in argsin:
                infilelist.append(file)
    del argsin
    if h5 == 'choose':
        h5file=fd.fileopen(message='Choose a Summary Table...',
                           filters='h5 (*.h5)',
                           defaultDir=defaultDir[:-5])
        h5table = HDF5saver(filename=h5file[0], testtype='sand')
    elif h5 == 'default':
        if strain:
            h5table = HDF5saver(filename='StrainSummaryResults.h5',
                   testtype='sand', strain=True)
        else:
            h5table = HDF5saver(testtype='sand')
    else:
        if strain:
            h5table = HDF5saver(filename=h5,
                   testtype='sand', strain=True)
        else:
            h5table = HDF5saver(filename=h5, testtype='sand')


    for infile in infilelist:
        if strain:
            testobj = SoilTest(infile=infile,
                               loadcorr=loadcorr,
                               loadcorr_type=loadcorr_type,
                               extrafigs=extrafigs,
                               cycletype=cycletype,
                               rufailval=rufailval,
                               strain=strain#, **kwargs
                               )
        else:
            testobj = SoilTest(infile=infile,
                               loadcorr = loadcorr,
                               loadcorr_type=loadcorr_type,
                               cycletype=cycletype,
                               strainfailval=strainfailval,
                               PaperFigs=PaperFigs,
                               rufailval=rufailval
                               #**kwargs
                               )#'strain01')
        if testobj.Error or PaperFigs:
            #pdb.set_trace()
            pass
        else:
            h5table.input(testobj)
        del testobj
    if PaperFigs:
        pass
    else:
        if kwargs.get('h5plot', True):
            h5table.plot()
        h5table.close()

def runEQ(*args,**kwargs):
    '''
```

```
2795      Runs eq test files into SoilTest and saves the results via HDF5saver
2796      '''
2797      #cycletype = kwargs.get('cycletype', 'strainSA')
2798      cycletype = kwargs.get('cycletype', 'ru')
2799      strainfailval = kwargs.get('strainfailval', '02')
2800      rufailval = kwargs.get('rufailval', '98')
2801      loadcorr = kwargs.get('loadcorr', False)
2802      loadcorr_type = kwargs.get('loadcorr_type', 'bilinear')
2803      h5plot = kwargs.get('h5plot', True)
2804      if os.name == 'posix':
2805          defaultDir=(u'/media/Storage/Documents/Python'+
2806                      '/GCTS/ConstantVolume/EQ')
2807      else:
2808          defaultDir=''
2809      h5 = kwargs.get('h5file','default')
2810      infilelist=[]
2811      if len(args) > 0:
2812          argsin = args
2813      else:
2814          argsin=sys.argv[1:]
2815
2816      if argsin==[]:
2817          filelist=fd.fileopen(message='Choose a gcts output of an EQ test...',
2818  #                        filters='CSV|*.csv', # for the wxfiledialog
2819                           filters='.csv files (*.csv)', #for qtfiledialog
2820                           defaultDir=defaultDir)
2821          for file in filelist:
2822               infilelist.append(file)
2823          del filelist
2824      elif len(argsin)>0:
2825          if argsin[0] in ['all','--all','-a','a']:
2826             infilelist=glob.glob('./EQ/*.csv')
2827          else:
2828              for file in argsin:
2829                  infilelist.append(file)
2830      del argsin
2831      if h5 == 'choose':
2832          h5file=fd.fileopen(message='Choose a Summary Table...',
2833                           filters='h5|*.h5',
2834                           defaultDir=defaultDir)
2835          h5table = HDF5saver(filename=h5file[0], testtype='EQ')
2836      elif h5 == 'default':
2837         h5table = HDF5saver(testtype='EQ')
2838      else:
2839         h5table = HDF5saver(filename=h5, testtype='EQ')
2840
2841      for infile in infilelist:
2842
2843          testobj = SoilTest(infile=infile, loadcorr=loadcorr,
2844                             loadcorr_type=loadcorr_type,
2845                             testtype='EQ',
2846                            cycletype=cycletype,
2847                            strainfailval=strainfailval,
2848                             rufailval=rufailval
2849                            #**kwargs
2850                            )
2851          if testobj.Error:
```

```
2852            pass
2853        else:
2854            h5table.input(testobj)
2855
2856        del testobj
2857    if h5plot:
2858        h5table.plot()
2859    h5table.close()
2860
2861 def runsummary(**kwargs):
2862    '''
2863    Opens results files and plots everything
2864    '''
2865    filters = 'HDF5 (*.h5)' # for qtfiledialog
2866 #    filters = 'HDF5 |*.h5' # for wxfiledialog
2867    if os.name == 'posix':
2868        defaultDir=(u'/media/Storage/Documents/Python'+
2869                    '/GCTS/ConstantVolume')
2870    else:
2871        defaultDir=''
2872    if os.path.exists(kwargs.get('h5table', '')):#,'./SummaryResults.h5')):
2873        h5table = HDF5saver(filename=kwargs.get('h5table',
2874                                                './SummaryResults.h5'))
2875    else:
2876        h5file=fd.fileopen(message='Choose a Summary Table...',
2877                           filters=filters,
2878                           defaultDir=defaultDir)
2879        h5table = HDF5saver(filename=h5file[0],**kwargs)
2880    h5table.plot(**kwargs)
2881    h5table.close()
2882
2883 def delete(**kwargs):
2884    '''
2885    Will delete test results from an .h5 file.
2886    '''
2887    filters = 'HDF5 (*.h5)' # for qtfiledialog
2888 #    filters = 'HDF5 |*.h5' # for wxfiledialog
2889    if os.name == 'posix':
2890        defaultDir=(u'/media/Storage/Documents/Python'+
2891                    '/GCTS/ConstantVolume')
2892    else:
2893        defaultDir=''
2894    if os.path.exists(kwargs.get('h5table','')):#'./SummaryResults.h5')):
2895        h5table = HDF5saver(filename=kwargs.get('filename',
2896                                                './SummaryResults.h5'))
2897    else:
2898        h5file=fd.fileopen(message='Choose a Summary Table...',
2899                           filters=filters,
2900                           defaultDir=defaultDir)
2901        h5table = HDF5saver(filename=h5file[0],**kwargs)
2902    h5table.delete(**kwargs)
2903    h5table.close()
2904
2905 def selector():
2906    '''
2907    What would you like to do today?
2908    '''
```

```
    x = None

    while x not in [1,2,3,4,5,6,7, 8, 9, 10, 11]:
        print('Run Options:\n 1. Select a GCTS soil output\n' +
              ' 2. Select a GCTS soil output and choose .h5 file\n' +
              ' 3. Select GCTS EQ motions\n' +
              ' 4. Select a GCTS water output\n' +
              ' 5. Select a GCTS Surcharge Friction Test\n' +
              ' 6. Plot all data\n' +
              ' 7. Import all data and go to a prompt\n'
              ' 8. Choose a data table and go to a prompt\n' +
              ' 9. Select a strain-controlled test\n' +
              ' 10. Delete some test results \n' +
              ' 11. Plot \'PaperFigs\'\n' +
              ' 12. Quit')
        selection = raw_input('Please make a selection: ').split()
        try:
            x = int(selection[0])
        except IndexError:
            x = None
        if x == 1:
            runsoil(h5file='default')
        elif x == 2:
            runsoil(h5file='choose')
        elif x == 3:
            runEQ()
        elif x == 4:
            runwater()
        elif x == 5:
            runwater(surcharge=True)
        elif x == 6:
            runsummary()
        elif x == 7:
            dataprompt()
        elif x == 8:
            dataprompt2()
        elif x == 9:
            runsoil(strain=True, cycletype='ru')
        elif x == 10:
            delete()
        elif x == 11:
            runsoil('./Sine/2014-11-21-3-dss.csv', h5file='default',
                    PaperFigs=True, h5plot=False, cycletype='strainSA',
                    strainfailval='03', loadcorr=False)
        elif x == 12:
            break
        else:
            print('I didn\'t understand you.')
```

J.9 Data Reduction of Seismic Compression Lab Data

The file `gcts_sc.py` takes the output files from CSS seismic compression tests on the GCTS equipment and reduces the data.

```
,
PaperFigures = True
import os
import sys
if os.name == 'posix':
    sys.path.append('/media/Storage/Documents/Python/Modules')
else:
    sys.path.append('D:\Documents\Python\Modules')
try:
    import ipdb as pdb
except:
    import pdb
import re

import glob
import fnmatch
import csv
import numpy as np
if PaperFigures:
    import matplotlib
    matplotlib.rcParams.update({'font.size': 18, 'font.family': 'STIXGeneral',
        'mathtext.fontset': 'stix'})
import matplotlib.pyplot as plt
import pandas as pd
import cPickle as p
#import wxfiledialog as fd
import qtfiledialog as fd
import tables as pt
from time import strftime
import datetime as dt
from scipy.optimize import curve_fit
import CycleCounting as cc
from scipy.interpolate import interp1d
from scipy.special import erf

def logcdf(x, sig):
    mu = 0.
    a, b = 1., 1.
    return a + b * erf((np.log(x) - mu) / sig)

def logistic(x,alpha,beta):
    return 1. / (1 + np.exp(beta * x + alpha))

def rnsine(x, a, b, c):
    return x**(a - b*np.sin(x+c))

def rnlogis(x, a, c):
    return x ** (a - 1. / (1 + np.exp(c * x)))
```

```
47
48  def rn2(x,a,b):
49      return x ** (a + 1. / (b * x))
50
51  def curvPL(x, a, b):
52      return x ** ( a + b*x)
53
54  def curv_power(x, a, b, c):
55      return x ** (a + b * x + c * x ** 2)
56
57  def curv_power2(x, a, b):
58      return x ** (1 + a + b * x)
59
60  class asymptotecons(object):
61      def __init__(self, **kwargs):
62          self.b = kwargs.get('b')
63
64      def asymptote(self,x,a,c):
65          y = a / x ** c + self.b
66          return y
67
68      def power_asymp (x, a, c):
69          y = x ** (a / x ** c + self.b)
70          return y
71
72  def power_asymp (x, a, b, c):
73      y = x ** (a / x ** c + b)
74      return y
75
76  def asymptote(x, a, b, c):
77      return a / x ** c + b
78
79  def asymptote3(x, b, c):
80      return 0.1 / x ** c + b
81
82  def asymptote4(x, b):
83      return 0.1 / x ** b + b
84
85  def asymptote5(x, a, b):
86      return a / x ** b + b
87
88  def asymptote2(x, a, b):
89      return a / (x ** b * (x - 1)) + b
90
91  def ellipse(x, a, b):
92      return np.sqrt(b**2 - b**2 * (x - a)**2 / a**2)
93
94  def exp_cutoff(x,a,b):
95      return x ** a * np.exp(b * x)
96
97  def power(x,a,b):
98      return a * x ** b
99
100 class SumTable(pt.IsDescription):
101     '''
102     Format of output PyTable
103     '''
```

```
104     File = pt.StringCol(itemsize=16,pos=1)
105     e_0 = pt.Float32Col(pos=2)
106     e_1 = pt.Float32Col(pos=3)
107     Dr = pt.Float32Col(pos=4)
108     gam = pt.Float32Col(pos=5)
109     Nlim = pt.Float32Col(pos=6)
110     r0 = pt.Float32Col(pos=7)
111     s0 = pt.Float32Col(pos=8)
112     t0 = pt.Float32Col(pos=9)
113     r1 = pt.Float32Col(pos=10)
114     s1 = pt.Float32Col(pos=11)
115     t1 = pt.Float32Col(pos=12)
116     Gs = pt.Float32Col(pos=13)
117     mass = pt.Float32Col(pos=14)
118     Hcyc1 = pt.Float32Col(pos=15)
119     di = pt.Float32Col(pos=16)
120     emin = pt.Float32Col(pos=17)
121     emax = pt.Float32Col(pos=18)
122     sigv_nom = pt.Float32Col(pos=19)
123     Dr_nom = pt.Float32Col(pos=20)
124     mass_ass = pt.Float32Col(pos=21)
125     Modified = pt.StringCol(itemsize=32,pos=22)
126     Date_of_Test = pt.Int64Col(pos=23)
127     Filepath = pt.StringCol(itemsize=128,pos=24)
128     rpow = pt.Float32Col(pos=25)
129     apow = pt.Float32Col(pos=26)
130     rmean = pt.Float32Col(pos=27)
131     aa = pt.Float32Col(pos=28)
132     ab = pt.Float32Col(pos=29)
133     ac = pt.Float32Col(pos=30)
134     DW_1 = pt.Float32Col(pos=31)
135     C1 = pt.Float32Col(pos=32)
136     C2 = pt.Float32Col(pos=33)
137     rn2_a = pt.Float32Col(pos=34)
138     rn2_b = pt.Float32Col(pos=35)
139     mu = pt.Float32Col(pos=36)
140     sig = pt.Float32Col(pos=37)
141     ncycles = pt.Float32Col(shape=(10,1), pos=38)
142
143
144 class EQSumTable(pt.IsDescription):
145     '''
146     Format of output PyTable
147     '''
148     File = pt.StringCol(itemsize=128,pos=1)
149     Modified = pt.StringCol(itemsize=32,pos=2)
150     sigv = pt.Float32Col(pos=3)
151     Ar = pt.Float32Col(pos=4)
152     MaxSStrain = pt.Float32Col(pos=5)
153     MaxNStrain = pt.Float32Col(pos=6)
154     Duration = pt.Float32Col(pos=7)
155     Dr = pt.Float32Col(pos=8)
156     PPresid = pt.Float32Col(pos=9)
157     PPpererr = pt.Float32Col(pos=10)
158     ZCresid = pt.Float32Col(pos=11)
159     ZCpererr = pt.Float32Col(pos=12)
160     RFresid = pt.Float32Col(pos=13)
```

```
    RFpererr = pt.Float32Col(pos=14)
    bPPresid = pt.Float32Col(pos=15)
    bPPpererr = pt.Float32Col(pos=16)
    bZCresid = pt.Float32Col(pos=17)
    bZCpererr = pt.Float32Col(pos=18)
    bRFresid = pt.Float32Col(pos=19)
    bRFpererr = pt.Float32Col(pos=20)
    lPPresid = pt.Float32Col(pos=21)
    lPPpererr = pt.Float32Col(pos=22)
    lZCresid = pt.Float32Col(pos=23)
    lZCpererr = pt.Float32Col(pos=24)
    lRFresid = pt.Float32Col(pos=25)
    lRFpererr = pt.Float32Col(pos=26)
    MotionType = pt.StringCol(itemsize=32,pos=27)
    DissEn = pt.Float32Col(pos=28)
    Damage = pt.Float32Col(pos=29)
    Date_of_Test = pt.Int64Col(pos=30)
    Filepath = pt.StringCol(itemsize=128,pos=31)
    steptype = pt.StringCol(itemsize=2,pos=32)


def calcRmsAccel(motion, dt, **kwargs):
    '''
    Calculates the rms acceleration.

    Keyword Arguments:
        duration       5to95 [default] Uses the duration from 5-95% of
                          the area under the motion squared
                       total       Uses the entire time of the input motion

    '''
    motion = np.ravel(motion)
    durationtype = kwargs.get('duration','5to95')
    NPTS = len(motion)
    time = np.linspace(0,(NPTS-1) * dt,NPTS)
    asqr = np.cumsum(motion ** 2) * dt

    if durationtype == '5to95':

        int1 = interp1d( asqr, time,
                            kind='linear')
        T = int1(0.95 * asqr[-1]) - int1(0.05 * asqr[-1])
    else:
        T = time[-1]

    return np.sqrt(asqr[-1] / T)


def datetimeconvert(datestr,timestr):
    try:
        date = dt.datetime.strptime(datestr+timestr,
                                    '%m/%d/%y%H:%M:%S')
    except ValueError:
        date = dt.datetime.strptime(datestr+timestr,
                                    '%m/%d/%Y%H:%M:%S')
    return int(date.strftime('%y%m%d%H%M%S'))

```

```
def unit_lookup(header1,header2,Label):
    '''
    Returns the unit of a given variable (Label).
    header1 is an array of labels: Shear Load, Shear Displacement, etc
    header2 is an array of units corresponding to the labels: kN, mm, etc
    Label is label for which a unit is desired.
    '''
    for i,label in enumerate(header1):
        if Label == label:
            return header2[i]
            break

class RichNewInc(object):
    '''
    Calculates the value of volumetric strain using Richart-Newmark Hypo.
    Returns not the increment, but the next value of volumetric strain.
    It is assumed that each shear strain is one half cycle.
    '''
    def __init__(self, Cycles, Amp, **kwargs):
        self.name = kwargs.get('name')
        self.Dr = kwargs.get('Dr')
        self.sigv = kwargs.get('sigv', 100.)
        self.rtype = kwargs.get('rtype', 'asymptotic')
        self.Amp = Amp
        self.threshold = kwargs.get('threshold', 0.0002)
        self.rcalctype = kwargs.get('rcalctype', 'every')
        self.Cyc = Cycles
        self.Ncalc = kwargs.get('Ncalc', None)
        if self.Ncalc == None:
            self.Ncalc = Ncycles()
        self.N = np.exp(self.Ncalc(0.01, np.log(self.Amp), self.sigv/101.325,
                           self.Dr))
        self.N[self.Amp<self.threshold] = np.inf
        self.D = np.zeros(Amp.shape, dtype=float)
        self.i = -1
        self.Ratio = self.Cyc / self.N
        self.R = np.cumsum(self.Ratio)
        #self.calcr(ToState=True)

    def calcr(self, **kwargs):
        '''
        calculates r where D = R^r.
        '''
        Amp = kwargs.get('Amp', self.Amp)
        Dr = kwargs.get('Dr', self.Dr)
        sigv = kwargs.get('sigv', self.sigv)
        R = kwargs.get('R', self.R)
        ToState = kwargs.get('ToState', False)

        if self.rtype == 'quadratic':
            Rlim = np.min((R, 1.2*np.ones(R.shape)), axis=0)
            r0 = (0.3606 - 0.00257 * Dr + 61.1 *
                       Amp - 0.0004098 * sigv)
            if np.isscalar(Amp):
                if r0 < 0.045:
                    r0 = 0.045
                elif r0 > 1.:
```

```
                r0 = 1.
            else:
                r0[r0<0.045] = 0.045 # Limited here!!!
            r = (r0 - r0**2 * Rlim + r0 ** 4 * Rlim ** 2)
            if ToState:
                self.r0 = r0
        elif self.rtype == 'mean':
            r = (1.811 - 0.0003387 * sigv - 0.002351 * Dr +
                0.2253 * np.log(Amp))
            r = np.max((0.2*np.ones(Amp.shape),r),axis=0)
        elif self.rtype == 'asymptotic':
            a = 0.1 #* np.ones(Amp.shape, dtype=float)
            b = (0.2380 - 0.00202 * Dr + 35.234 * Amp -
                        0.0603 * sigv / 101.325 + 7.09347 * Amp
                         * sigv /101.325)
            if b > 1.:
                b = 1.
                #print('b = 1')
            if b < 0 :
                #print('b = 0')
                b = 0.
                #pdb.set_trace()
            c = b
            r = a / R ** c + b
            if ToState:
                self.a, self.b, self.c = a*np.ones(self.N.shape), b, c
        # Limit r
        #r = np.min((np.ones(r.shape, dtype=float),r),axis=0)
        if ToState:
            self.r = r
        else:
            return r

    def __call__(self, **kwargs):
        '''
        This is an iterator-type function.
        Calculates next value of SC for next time step
        '''
        returns = kwargs.get('returns', True)
        self.i += 1
        j = self.i
        if self.Amp[j] < self.threshold:
            self.D[j] = self.D[j-1]
            self.R[j] = self.R[j-1]
        else:
            if j == 0:
                self.R[j] = self.Ratio[j]
                r = self.calcr(R=self.R[j], Amp=self.Amp[j])# self.r[j]
            else:
                self.R[j] = self.R[j-1] + self.Ratio[j]
                r = self.calcr(R=self.R[j], Amp=self.Amp[j])# self.r[j]
                if self.rcalctype == 'every':
                    R = self.R[j]
                    r2 = self.calcr(R=R, Amp=self.Amp[j])
                    count = 0
                    R2 = self.D[j-1] ** ( 1/ r2) + self.Ratio[j]
                    #while np.abs(R2 - R)/R > 0.001:
```

```
                    #count += 1
                    #R = R2
                    #r2 = self.calcr(R=R, Amp=self.Amp[j])
                    #R2 = self.D[j-1] ** ( 1/ r2) + self.Ratio[j]
                while (np.abs(r - r2) / r) > 0.01:
                    count += 1
                    R = self.D[j-1]**(1/(r2)) + self.Ratio[j]
                    r = r2
                    r2 = self.calcr(R=R, Amp=self.Amp[j])
                    if count > 100:
                        if self.Amp[j] < 5e-5:
                            break
                        else:
                            pdb.set_trace()
                            #print('Took avg r')
                            #r = (r + r2) / 2.
                r = r2
            self.R[j] = self.D[j-1]**(1/r) + self.Ratio[j]
            self.D[j] = (self.D[j-1] ** (1/r) + self.Ratio[j]) ** r
    if returns:
        return self.D[self.i]

def run(self):
    '''
    This just runs thru all the calls
    '''
    for val in self.Amp:
        self.__call__(returns=False)

def totalVolStrain(self):
    '''
    This returns the final value of volumetric strain only.
    '''
    self.run()
    if self.D[-1] == np.inf:
        pdb.set_trace()
    return self.D[-1] * 0.01

def calcNeq(self, **kwargs):
    threshold = kwargs.get('threshold', self.threshold)
    gameff = kwargs.get('gameff',np.max(np.abs(self.Amp)) * 0.65)
    self.gamNeq = gameff
    if gameff < threshold:
        return np.inf
    else:
        N = np.exp(self.Ncalc(0.01, np.log(gameff), self.sigv/101.325,
                    self.Dr))
    finalD = self.D[-1]
    if finalD == 0:
        self.run()
        finalD = self.D[-1]

    # Iterative section: Guess an R
    R = finalD ** (1/0.3)
    r = self.calcr(R=R, Amp=gameff)
    R2 = finalD ** (1 / r)
    count = 0
```

```
        while np.abs(R-R2)/R2 > 0.01:
            count += 1
            R = R2
            r = self.calcr(R=R, Amp=gameff)
            R2 = finalD ** (1 / r)
            if count > 100:
                if np.abs(R-R2)/R2 < 0.1:
                    break
                if np.abs(R-R2) < 0.1:
                    break
                return -2000
        self.Neq = R * N
        return self.Neq



class LineAcc(object):
    '''
    Calculates the value of volumetric strain assuming a linear accumulation
    of damage (Palmgren-Miner).
    Returns not the increment, but the next value of damage.
    To obtain the shear strain, multiply by the value of ev.
    '''
    def __init__(self, Cycles, Amp, **kwargs):
        self.name = kwargs.get('name')
        #self.Ntype = kwargs.get('Ntype',0.01)
        self.Dr = kwargs.get('Dr')
        self.sigv = kwargs.get('sigv', 100.)
        self.threshold = kwargs.get('threshold', 0.0002)
        self.Amp = np.max([Amp, np.ones(Amp.shape)*0.000], axis=0)
        self.Cyc = Cycles
        self.Ncalc = kwargs.get('Ncalc')
        self.ev = kwargs.get('ev',0.01)
        self.N = np.exp(self.Ncalc(self.ev, np.log(self.Amp), self.sigv/101.325,
                        self.Dr))
        self.N[self.Amp<self.threshold] = np.inf
        self._R = Cycles / self.N
        self.D = np.cumsum(self._R)
        self.i = -1

    def __call__(self, **kwargs):
        '''
        This is an iterator function.
        '''
        returns = kwargs.get('returns', True)
        self.i += 1
        if returns:
            return self.D[self.i]

    def calcNeq(self, **kwargs):
        '''
        Calculates Equiv Number of Cycles
        '''
        evtotal = kwargs.get('evtotal', self.D[-1])
        gameff = kwargs.get('gameff', 0.65 * np.max(np.abs(self.Amp)))
        evN = kwargs.get('ev', self.ev)
        if gameff < self.threshold:
```

```
            self.Neq = np.inf
            return np.inf
        else:
            N = np.exp(self.Ncalc(evN, np.log(gameff), self.sigv/101.325,
                      self.Dr))


        self.Neq = evtotal * N / evN
        return self.Neq

def ByrneNeq(EvTarget, gam, Dr, **kwargs):
    '''
    Calculates the number of equivalent cycles using Byrne's model.
    '''
    threshold = kwargs.get('threshold', 0.0002)
    original = kwargs.get('original', True)
    sigv = kwargs.get('sigv', 100.)
    cyclelimit = kwargs.get('cyclelimit', 3000)
    Ev = 0.
    cyclecount = 0.
    while Ev < EvTarget:
        cyclecount += 0.5
        Ev = ByrneIncrem(Ev,
                    gam,
                    Dr = Dr,
                    cycles=0.5,
                    sigv=sigv,
                    threshold=threshold,
                    original=original)
        if cyclecount > cyclelimit:
            break

    return cyclecount

def ByrneIncAll(gam, Dr, **kwargs):
    '''
    Does the same as ByrneIncrem, but all at once.
    '''
    cyc = kwargs.get('cycles', 0.5 * np.ones(gam.shape))
    threshold = kwargs.get('threshold', 0.0002)
    original = kwargs.get('original', True)
    sigv = kwargs.get('sigv', 100.)
    Ev = np.zeros(gam.shape)
    for i,amp in enumerate(gam):
        Ev[i] = ByrneIncrem(Ev[i-1],
                    amp,
                    Dr = Dr,
                    cycles=cyc[i],
                    sigv=sigv,
                    threshold=threshold,
                    original=original)

    return Ev

def ByrneIncrem(Evprev, gam, **kwargs):
    '''
    Calculates the value of volumetric strain using Byrne's method (1991).
```

```
503     Returns not the increment, but the next value of volumetric strain.
504     It is assumed that each shear strain is one half cycle.
505     Make sure your Ev previous and gam are the same units.
506     '''
507     Dr = kwargs.get('Dr')
508     sigv = kwargs.get('sigv', 100.)
509     cyc = kwargs.get('cycles', 0.5)
510     threshold = kwargs.get('threshold', 0.0002)
511 #    if cyc != 0.5:
512 #        pdb.set_trace()
513     if kwargs.get('original', True):
514         c1 = 7600 * Dr ** -2.5
515         c2 = 0.4 / c1
516     else:
517         c1 = 0.456 - 67.17 * gam + 0.005046 * Dr + 0.001413 * sigv
518         c2 = -0.6306 + 101.521 * gam + 0.0207865 * Dr + 0.7257 * c1
519     if gam < threshold:
520         Evinc = 0
521     else:
522         #gam = gam - threshold # This is what Byrne does; see eq. 17
523         # The above gives better results and is what Byrne did,
524         # but it is inconsistent with the way I have calibrated the model.
525         passes = int(cyc / 0.5)
526         i = 0
527         Evp = Evprev
528         while i < passes:
529             i += 1
530             Evinc = 0.5 * gam * c1 * np.exp(-c2 * Evp / gam)
531             Evp = Evinc + Evp
532         Evinc = Evp - Evprev
533 
534     return Evinc + Evprev
535 
536 class HDF5saver(object):
537     '''
538     This object will save the data to a pytable HDF5 file.
539     '''
540     def __init__(self,**kwargs):
541         testtype = kwargs.get('testtype','sine')
542 
543         if testtype == 'sine':
544             filename = kwargs.get('filename','SummaryResults.h5')
545             # Check to see if the output table exist, if not, create it.
546             self.outfile = pt.openFile(filename,mode='a',
547                                        title='Cyclic Compression Lab Testing Results')
548             # Check if the internal summary table exists
549             if self.outfile.__contains__('/Summary'):
550                 self.summtable = self.outfile.root.Summary
551             else:
552                 self.summtable = self.outfile.createTable('/','Summary',
553                                                           SumTable)
554         else:
555             filename = kwargs.get('filename','EQSummary.h5')
556             # Check to see if the output table exist, if not, create it.
557             self.outfile = pt.openFile(filename,mode='w', # notice how I am deleting the
                    previous table.
558                                        title='EQ Motion Testing Results')
```

```
559             # Check if the internal summary table exists
560             if self.outfile.__contains__('/EQSummary'):
561                 self.summtable = self.outfile.root.EQSummary
562             else:
563                 self.summtable = self.outfile.createTable('/',
564                                                           'EQSummary',
565                                                           EQSumTable)
566         self.testtype = testtype
567 
568     def input(self, ResultObj, **kwargs):
569         '''
570         This is the mechanism whereby data is input into the table.
571         '''
572         # Check to see if an entry already exists
573         name = ResultObj.infile
574         infile = name
575         name = str(os.path.basename(name))
576         if self.testtype == 'sine':
577             self.delete(name)
578             entry = self.summtable.row
579             entry['File'] = name
580             entry['e_0'] = ResultObj.e_0
581             entry['e_1'] = ResultObj.e_1
582             entry['Dr'] = ResultObj.Dr
583             entry['gam'] = ResultObj.gam
584             entry['Nlim'] = ResultObj.Nlim
585             if ResultObj.Nlim < 4:
586                 entry['r0'] = np.nan
587                 entry['s0'] = np.nan
588                 entry['t0'] = np.nan
589                 entry['r1'] = np.nan
590                 entry['s1'] = np.nan
591                 entry['t1'] = np.nan
592                 entry['rpow'] = np.nan
593                 entry['apow'] = np.nan
594                 entry['rmean'] = np.nan
595                 entry['aa'] = np.nan
596                 entry['ab'] = np.nan
597                 entry['ac'] = np.nan
598                 if ResultObj.lognorm:
599                     entry['mu'] = np.nan
600                     entry['sig'] = np.nan
601                 entry['rn2_a'] = np.nan
602                 entry['rn2_b'] = np.nan
603             else:
604                 entry['r0'] = ResultObj.r0
605                 entry['s0'] = ResultObj.s0
606                 entry['t0'] = ResultObj.t0
607                 entry['r1'] = ResultObj.r1
608                 entry['s1'] = ResultObj.s1
609                 entry['t1'] = ResultObj.t1
610                 entry['rpow'] = ResultObj.rpow
611                 entry['apow'] = ResultObj.apow
612                 entry['rmean'] = ResultObj.rmean
613                 entry['aa'] = ResultObj.aa
614                 entry['ab'] = ResultObj.ab
615                 entry['ac'] = ResultObj.ac
```

```
            if ResultObj.lognorm:
                entry['mu'] = ResultObj.mu
                entry['sig'] = ResultObj.sig
            entry['rn2_a'] = ResultObj.rn2_a
            entry['rn2_b'] = ResultObj.rn2_b
        entry['Gs'] = ResultObj.Gs
        entry['mass'] = ResultObj.mass
        entry['Hcyc1'] = ResultObj.Hcyc1
        entry['di'] = ResultObj.di
        entry['emin'] = ResultObj.emin
        entry['emax'] = ResultObj.emax
        entry['sigv_nom'] = ResultObj.sigv_nom
        entry['Dr_nom'] = ResultObj.Dr_nom
        entry['mass_ass'] = ResultObj.mass_ass
        entry['Modified'] = strftime("%Y-%m-%d %H:%M:%S")
        entry['Date_of_Test'] = ResultObj.date_of_test
        entry['Filepath'] = infile
        entry['DW_1'] = np.mean(ResultObj.cyc['DE'].values[-50:] /
                        ResultObj.cyc['Cycles'].values[-50:])
        if ResultObj.byrne:
            entry['C1'] = ResultObj.C1
            entry['C2'] = ResultObj.C2
        else:
            entry['C1'] = -1
            entry['C2'] = -1
        entry['ncycles'] = ResultObj.ncycles.reshape(10,1)
    else:
        entry = self.summtable.row
        entry['File'] = name
        entry['Modified'] = strftime("%Y-%m-%d %H:%M:%S")
        entry['sigv'] = ResultObj.sigv
        entry['Ar'] = ResultObj.Ar
        entry['MaxSStrain'] = ResultObj.MaxSStrain
        entry['MaxNStrain'] = ResultObj.MaxNStrain
        entry['Duration'] = ResultObj.Duration
        entry['Dr'] = ResultObj.Dr
        entry['PPresid'] = ResultObj.resid[0]
        entry['PPpererr'] = ResultObj.perError[0]
        entry['ZCresid'] = ResultObj.resid[1]
        entry['ZCpererr'] = ResultObj.perError[1]
        entry['RFresid'] = ResultObj.resid[2]
        entry['RFpererr'] = ResultObj.perError[2]
        if ResultObj.byrne:
            entry['bPPresid'] = ResultObj.bresid[0]
            entry['bPPpererr'] = ResultObj.bperError[0]
            entry['bZCresid'] = ResultObj.bresid[1]
            entry['bZCpererr'] = ResultObj.bperError[1]
            entry['bRFresid'] = ResultObj.bresid[2]
            entry['bRFpererr'] = ResultObj.bperError[2]
        if ResultObj.linear:
            entry['lPPresid'] = ResultObj.lresid[0]
            entry['lPPpererr'] = ResultObj.lperError[0]
            entry['lZCresid'] = ResultObj.lresid[1]
            entry['lZCpererr'] = ResultObj.lperError[1]
            entry['lRFresid'] = ResultObj.lresid[2]
            entry['lRFpererr'] = ResultObj.lperError[2]
        entry['MotionType'] = ResultObj.MotType
```

```
673             entry['DissEn'] = ResultObj.cyc['DE'].values[-1]
674             entry['Damage'] = ResultObj.cyc['Dam'].values[-1]
675             entry['Date_of_Test'] = ResultObj.date_of_test
676             entry['Filepath'] = infile
677             entry['steptype'] = ResultObj.steptype
678         entry.append()
679         self.summtable.flush()
680
681
682     def read(self):
683         '''
684         Reads in the result of the Summary Table.
685         '''
686         results = self.summtable.read()
687         return results
688
689     def delete(self,*arg, **kwargs):
690         '''
691         Deletes an entry from an H5 Table.
692         '''
693         filters = 'csv (*.csv)' # for qtfiledialog
694 #       filters = 'csv |*.csv' # for wxfiledialog
695         returns = kwargs.get('returns', False)
696         infilelist = []
697         if len(arg) == 0:
698             filelist=fd.fileopen(
699                    message='Select the files you want to remove from the table:',
700                    filters=filters,
701                    defaultDir=(u'/media/Storage/Documents/Python'+
702                             '/GCTS/ConstantVolume'))
703             print('Files to remove from {}:\n\n'.format(self.summtable))
704
705             for file1 in filelist:
706                 infilelist.append(file1)
707                 print(file1 + '\n')
708             del filelist
709             print('\n')
710             selection = raw_input('Are you sure you want to remove these ' +
711             'files from "{}"? [y]/n: '.format(self.filename))
712
713             if selection.lower() == 'n':
714                 return
715             elif selection.lower() == 'y':
716                 pass
717             elif selection == '':
718                 pass
719             else:
720                 return
721         else:
722             for val in arg:
723                 infilelist.append(val)
724
725         rowdata = []
726         for name in infilelist:
727             name = str(os.path.basename(name)).split('.')[0]
728             try:
729                 row1 = self.summtable.getWhereList('File == name')
```

```
730             except NameError:
731                 row1 = self.summtable.get_where_list('File == name')
732             except AttributeError:
733                 pdb.set_trace()
734             if len(row1) == 0:
735                 pass
736             else:
737                 for i,val in enumerate(row1):
738                     if kwargs.get('returns') == True:
739                         rowdata.append(self.summtable.read(val,val+1))
740                     try:
741                         self.summtable.remove_row(val)
742                     except NameError:
743                         self.summtable.removeRows(val-i)
744                     except IndexError, NotImplementedError:
745                         pdb.set_trace()
746         if returns == True:
747             return rowdata
748 
749     def plot(self, **kwargs):
750         interval = kwargs.get('interval',24.)
751         from mpl_toolkits.mplot3d import Axes3D
752         gam = self.summtable.read(field='gam')
753         Dr = self.summtable.read(field='Dr')
754         Nlim = self.summtable.read(field='Nlim')
755         sigv_nom = self.summtable.read(field='sigv_nom')
756         C1 = self.summtable.read(field='C1')
757         C2 = self.summtable.read(field='C2')
758         C2C1 = C2 * C1
759         rn2_a = self.summtable.read(field='rn2_a')
760         rn2_b = self.summtable.read(field='rn2_b')
761         Date = self.summtable.read(field='Modified')
762         fig = plt.figure(1)
763 #        ax = fig.add_subplot(111, projection='3d')
764         ax = fig.gca(projection='3d')
765         ax.scatter(gam,Dr,zs=C1, marker='o', s=75, c=sigv_nom, cmap='spectral',
766               alpha=1, edgecolors='none',)
767 #        ax.scatter(CSR[ind2],Dr[ind2],zs=np.log10(N[ind2]), facecolor='r')
768         ax.set_zlabel(r'$C_1$')
769         ax.set_ylabel('Dr (%)')
770         ax.set_xlabel('$\gamma$')
771         x = np.linspace(0.002,0.01, num=10)
772         y = np.linspace(28,120,num=10)
773         X, Y = np.meshgrid(x, y)
774         c1 = 0.456 - 67.17 * X + 0.005046 * Y + 0.001413 * 100
775         ax.plot_surface(X,Y,c1, rstride=1, cstride=1, cmap='coolwarm',
776         linewidth=0, antialiased=False)
777 
778         fig = plt.figure(2)
779         ax = fig.add_subplot(111, projection='3d')
780         ax.scatter(gam,Dr,zs=C2, marker='o', s=75, c=sigv_nom, cmap='spectral',
781               alpha=1, edgecolors='none',)
782 #        ax.scatter(CSR[ind2],Dr[ind2],zs=np.log10(N[ind2]), facecolor='r')
783         ax.set_zlabel(r'$C_2$')
784         ax.set_ylabel('Dr (%)')
785         ax.set_xlabel('$\gamma$')
786         c2 = -0.6306 + 101.521 * X + 0.0207865 * Y + 0.7257 * c1
```

```
787         ax.plot_surface(X,Y,c2, rstride=1, cstride=1, cmap='coolwarm',
788         linewidth=0, antialiased=False)
789 
790         fig = plt.figure(3)
791         ax = fig.add_subplot(111, projection='3d')
792         ax.scatter(gam,Dr,zs=rn2_a, marker='o', s=75, c=sigv_nom, cmap='spectral',
793               alpha=1, edgecolors='none',)
794 #        ax.scatter(CSR[ind2],Dr[ind2],zs=np.log10(N[ind2]), facecolor='r')
795         ax.set_zlabel(r'$rn2 a$')
796         ax.set_ylabel('Dr (%)')
797         ax.set_xlabel('$\gamma$')
798 
799         fig = plt.figure(4)
800         ax = fig.add_subplot(111, projection='3d')
801         ax.scatter(gam,Dr,zs=rn2_b, marker='o', s=75, c=sigv_nom, cmap='spectral',
802               alpha=1, edgecolors='none',)
803 #        ax.scatter(CSR[ind2],Dr[ind2],zs=np.log10(N[ind2]), facecolor='r')
804         ax.set_zlabel(r'$rn2 b$')
805         ax.set_ylabel('Dr (%)')
806         ax.set_xlabel('$\gamma$')
807 
808         fig = plt.figure(5)
809         ax = fig.add_subplot(111, projection='3d')
810         ax.scatter(gam,Dr,zs=(rn2_a+1./(rn2_b*0.0001)), marker='o', s=75, c=sigv_nom,
              cmap='spectral',
811               alpha=1, edgecolors='none',)
812 #        ax.scatter(CSR[ind2],Dr[ind2],zs=np.log10(N[ind2]), facecolor='r')
813         ax.set_zlabel(r'rn2-low R')
814         ax.set_ylabel('Dr (%)')
815         ax.set_xlabel('$\gamma$')
816         plt.show()
817         pdb.set_trace()
818 
819     def close(self):
820         self.outfile.close()
821 
822 
823 class EQTest(object):
824     '''
825     This object controls all the calcs and i/o for each test.
826     Keyword          Definition
827     file             name/path of the GCTS data file for the test.
828     '''
829 
830     def __init__(self,**kwargs):
831         infile = kwargs.get('infile')
832         try:
833             if os.name == 'posix':
834                 self.sigv_nom = int(re.search('motions/(.+?)kPa',infile).group(1))
835                 self.Dr_nom = int(re.search('kPa/Dr(.+?)/',infile).group(1))
836                 self.MotType = re.search('Dr{}/(.+?)/'.format(self.Dr_nom),
837                                          infile).group(1)
838             else:
839                 self.sigv_nom = int(re.search('motions\\\(.+?)kPa',
840                                               infile).group(1))
841                 self.Dr_nom = int(re.search('kPa\\\\Dr(.+?)\\\\',
842                                             infile).group(1))
```

```
                self.MotType = re.search('Dr{}\\\\(.+?)\\\\'.format(
                                self.Dr_nom), infile).group(1)



        except AttributeError:
            print('AttributeError!!!!')

        self.limEvstrain = kwargs.get('limEvstrain',0.01)
        self.byrne = kwargs.get('byrne',True)
        self.lognorm = kwargs.get('lognorm', True)
        self.linear = kwargs.get('linear', True)
        self.emin = kwargs.get('emin',0.68)
        self.emax = kwargs.get('emax',0.9)

        self.verbose = kwargs.get('verbose',True)


        if 'infile' == None:
            infile = raw_input('Nothing to do. Specify an input file:')
            self.csvimport(infile)
        else:
            self.csvimport(infile)
        self.infile = infile
        self.Drcorr()
        self.calcs()
        self.DamagePred()
#        self.dissen(verbose=True)
#        self.plotter()

    def csvimport(self, infile):
        '''
        This function pulls in the data from the GCTS output file.
        The GCTS file should not have the data separated into stages.
        '''
        self.data=[]
        self.label=[]
        self.name=[]
        self.Gs=[]
        self.mass=[]
        self.cyc_header=[]
        self.pist_area=[]
        testtime = None
        footer = []
        datasection=0
        dimsec=0
        datafile = csv.reader(open(infile,'rU'))
        rowcount = -1
        header = None
        Volume = None
        for row in datafile:
            rowcount += 1
            if datasection == 0:
                if self.Gs:
                    pass
                elif 'Specific Gravity: ' in row:
                    self.Gs=float(row[1])
```

```
#                 self.Gs=2.66 #Because sometimes I forget to change the value in the
    GCTS software
                if self.name:
                    pass
                elif 'Specimen: ' in row:
                    self.name=row[1]

                if self.mass:
                    pass
                elif 'Dry Mass of Specimen: ' in row:
                    self.mass=float(row[1]); self.mass_unit=row[2]
                    if self.mass==0:
                        self.mass=float(raw_input('Input the dry mass (g) for specimen %s:'
                        %self.name))
                elif 'Moist Mass of Specimen: ' in row:
                    self.mass=float(row[1]); self.mass_unit=row[2]
                    if self.mass==0:
                        self.mass=float(raw_input('Input the dry mass (g) for specimen %s:'
                        %self.name))
                if 'Type:' and 'Universal' in row:
                    dimsec=1
                if 'Starting Date: ' in row:
                    testdate = row[1]
                if 'Starting Time: ' in row:
                    testtime = row[1]
            if dimsec==1:
                if 'Height: ' in row:
                    self.hi=float(row[3])#*25.4 ;
                    if row[4] == '(mm)':
                        pass
                    else:
                        self.hi=self.hi*25.4
                    self.hi_unit= 'mm' #Convert to mm
                if 'Diameter: ' in row:
                    self.di=float(row[3])#*25.4;
                    if row[4] == '(mm)':
                        pass
                    else:
                        self.di=self.di*25.4
                    self.di_unit= 'mm' #Convert to mm
                    self.area = (self.di / 2) ** 2 * np.pi # in mm^2
                if 'Volume: ' in row:
                    Volume = row[3]
                    dimsec=0
            if len(row) >= 17:
                if header == None:
                    if Volume != None:
                        datasection = 1
                    if row[0] == 'Time':
                        header = rowcount
                        self.headerlabel = row

                    if row[0] == 'sec':
                        self.units = row
            if datasection == 1:
                if row[0][-14:] == 'Static Loading':
                    footer = rowcount
```

```
                    break
        if footer == []:
            df = pd.read_csv(infile,names=self.headerlabel,skiprows=header+3)
        else:
            df = pd.read_csv(infile,names=self.headerlabel,skiprows=header+3,
                             nrows = footer - header - 4)
        df = df.convert_objects(convert_numeric=True)
        select = df.apply(lambda r : any([isinstance(e, basestring)
                          for e in r ]),axis=1)
        df = df[~select]
        df = df.dropna()

        # Fix the time values
        time = df['Time'].values
        ind = np.where(time[1:] < time[:-1] )
        if ind != [np.array([])]:
            for i,val in enumerate(ind[0]):
                if val == ind[0][-1]:
                    time[val+1:] += time[val]
                else:
                    time[val+1:ind[0][i+1]+1] += time[val]
            df['Time'] = time
    # Create a new dataframe that contains only the cyclic portion
        columns = ['Time', 'Cycles','Shear Load','Shear Displacement',
                   'Normal Load','Normal Displacement',
                   'Internal Shear Load','Internal Shear LVDT',
                   'Internal Normal LVDT']
        try:
            x = df['Shear Load'] * 2.
        except KeyError:
            df['Shear Load'] = df['Internal Shear Load']
        try:
            self.cyc = df[columns].reset_index(drop=True)
        except KeyError:
            raise KeyError
        self.date_of_test = datetimeconvert(testdate,testtime)


    def calcs(self, **kwargs):
        '''
        EQTest
        This does the heavy lifting.
        '''
        # Calculate the volumes
        self.Vcyc_T1 = self.Hcyc1 * self.area # in mm^2
        if self.Gs == 2.65:
            pass
        else:
            Gs = self.Gs
            self.Gs = 2.65
            print('Changed Gs from {} to 2.65'.format(Gs))
        try:
            self.Vcyc_s = self.mass / (self.Gs * 0.001) # in mm^3 where density of water is
                0.001 g/mm^3
        except TypeError:
            self.Vcyc_s = self.mass_ass / (self.Gs * 0.001) # in mm^3 where density of
                water is 0.001 g/mm^3
```

```
            self.mass = self.mass_ass
            print('Assuming a mass of {}g'.format(self.mass_ass))
        nLVDT = (self.cyc['Internal Normal LVDT'].values -
                 self.cyc['Internal Normal LVDT'].values[0])
        self.cyc['Volume'] = self.Vcyc_T1 - nLVDT * self.area
        self.cyc['Vol_voids'] = self.cyc['Volume'] - self.Vcyc_s
        # Calc e
        self.cyc['e'] = self.cyc['Vol_voids'] / self.Vcyc_s
        self.e_0 = self.cyc['e'].values[0]
        self.e_1 = 0.99 * self.cyc['Vol_voids'].values[0] / self.Vcyc_s - 0.01
        self.Dr = (self.emax-self.e_0)/(self.emax-self.emin)*100
        #Calc changes in volume, damage, cycle ratio
        self.cyc['DeltaV'] = (self.Vcyc_T1 - self.cyc['Volume']) / self.Vcyc_T1
        self.cyc['Dam'] = self.cyc['DeltaV'] / self.limEvstrain
        self.MaxNStrain = self.cyc['DeltaV'].values[-1]
        self.dt = np.mean(np.diff(self.cyc['Time'].values))
        self.SStrain = self.cyc['Internal Shear LVDT'].values / self.Hcyc1
        self.Ar = self.calcAr(self.SStrain,self.dt)
        self.sigv = (np.mean(self.cyc['Normal Load'].values / self.area) *
                     10**6) # to convert from mm^2 to m^2 to get kPa
        self.MaxSStrain = np.max(self.SStrain)
        ind5 = np.where(np.abs(self.SStrain) > 0.0005) # set this as needed for duration
        self.Duration = (self.cyc['Time'].values[ind5][-1] -
                         self.cyc['Time'].values[ind5][0])
        stresses = self.cyc['Shear Load'].values / (self.area * 10 ** -6)
        strains = self.cyc['Internal Shear LVDT'].values / self.Hcyc1
        DE1 = np.zeros(stresses.shape, dtype=float)
        DE1[1:] = np.cumsum((stresses[1:] + stresses[:-1]) *
                  (strains[1:] - strains[:-1])) * 0.5
        self.cyc['DE'] = DE1
        if self.MotType == 'Step loadin':
            amp1 = np.max(np.abs(strains[:-len(strains)/5]))
            if amp1 > 0.004:
                self.steptype = 'bf'
            else:
                self.steptype = 'bs'
        else:
            self.steptype = 'na'
            self.SStrain -= np.mean(self.SStrain[100:300])
#


    def DamagePred(self, **kwargs):
        '''
        Here is where I predict damage for the test.
        '''
        #threshold = kwargs.get('threshold', 0.0002)
        threshold = kwargs.get('threshold', 0.000)
        Load = cc.Load(self.SStrain, dt=self.dt)
        Amps = [Load.PP_Amp, Load.ZC_Amp, Load.RF_Amp]
        Ncalc = Ncycles()
        Cycs = [Load.PP_Cyc, Load.ZC_Cyc, Load.RF_Cyc]
        Times = [Load.PP_time, Load.ZC_time, Load.RF_time]
        colors = ['rs','gs','ms']
        colors2 = ['r^','g^','m^']
        colors3 = ['r+','g+','m+']
        colors4 = ['rx','gx','mx']
```

```
        labels = ['P2P','0X','RF']
        Damage = []
        if self.byrne:
            bDamage = []
        if self.lognorm:
            lnDamage = []
        if self.linear:
            lDamage = []
        plt.plot(self.cyc['Time'].values,self.cyc['DeltaV'].values,'-b',label='Data')
        for i, amp in enumerate(Amps):
            cyc = Cycs[i]
            time = Times[i]
            RNFunc = RichNewInc(cyc, amp, Dr=self.Dr, sigv=self.sigv_nom,
                   threshold=threshold, rtype='asymptotic', Ncalc=Ncalc,
                   rcalctype='every')
            D = np.zeros(cyc.shape, dtype=float)
            if self.lognorm:
                LogNormFunc = LogNormInc(cyc, amp, Dr=self.Dr,
                        sigv=self.sigv_nom, threshold=threshold)
                Dln = np.zeros(cyc.shape, dtype=float)
            if self.byrne:
                Dbyr = np.zeros(cyc.shape, dtype=float)
            if self.linear:
                Linear = LineAcc(cyc, amp, Dr=self.Dr, sigv=self.sigv_nom,
                       threshold=threshold, Ncalc=Ncalc,
                       ev=self.limEvstrain)
                Dlin = Linear.D
            for j in xrange(len(amp)):
                D[j] = RNFunc()
                if self.byrne:
                    Dbyr[j] = ByrneIncrem(Dbyr[j-1], amp[j],
                                                cycles=cyc[i],
                                                Dr=self.Dr,
                                                sigv=self.sigv_nom,
                                                threshold=threshold)
                if self.lognorm:
                    Dln[j] = LogNormFunc()
            plt.plot(time,D*0.01,colors[i],label='{}'.format(labels[i]+'-R-N'))
            Damage.append(D)
            if self.byrne:
                bDamage.append(Dbyr)
                plt.plot(time,Dbyr,colors2[i],
                           label='{}'.format(labels[i]+'-Byrne'))
            if self.lognorm:
                lnDamage.append(Dln)
                plt.plot(time,Dln*0.01,colors4[i],
                           label='{}'.format(labels[i]+'-ln'))
            if self.linear:
                lDamage.append(Dlin)
                plt.plot(time,Dlin*0.01,colors3[i],
                           label='{}'.format(labels[i]+'-P-M'))
        plt.legend(bbox_to_anchor=(1.05, -0.05), loc='lower right', borderaxespad=0.)
        plt.xlabel('Time (sec)')
        plt.ylabel(r'$\varepsilon _v$')
        [xmin, xmax,ymin,ymax] = plt.axis()
        plt.axis([xmin, xmax,ymin,np.min([1.7*self.cyc['DeltaV'].values[-1],ymax])])
        plt.savefig(self.infile[:-4] +'-1'+'.'+'png')
```

```
1125
1126        plt.figure('2')
1127        plt.plot(self.cyc['Time'].values, self.SStrain, 'k-', lw=1.5)
1128        plt.xlabel('Time (sec)')
1129        plt.ylabel('Shear Strain')
1130
1131        plt.gcf().subplots_adjust(wspace=0, bottom=0.2, left=0.2)
1132        plt.savefig(self.infile[:-4] +'-2'+'.'+'png')
1133        plt.savefig(self.infile[:-4] +'-2'+'.'+'pdf')
1134
1135        if PaperFigures:
1136            lw = 2
1137            plt.figure('Dam-PF')
1138            plt.plot(self.cyc['Time'].values[::10],
1139                   self.cyc.DeltaV.values[::10],
1140                   'o', color='gray',
1141                      ms=4, label='Data', alpha=0.9)
1142            plt.plot(Load.ZC_time, Damage[1] * self.limEvstrain, '-k', lw=lw,
1143                   label='R-N')
1144            plt.plot(Load.ZC_time, bDamage[1] , '--k', lw=lw,
1145                   dashes=(10,5),
1146                   label='Byrne')
1147            plt.plot(Load.ZC_time, lDamage[1] * self.limEvstrain, '-.k', lw=lw,
1148                   dashes=(10,5,2,5),
1149                   label='P-M')
1150            plt.legend(loc='lower right', numpoints=1) # try scatterpoints=1 if ax.legend
1151            plt.xlabel('Time (sec)')
1152            plt.ylabel(r'Volumetric Strain, $\varepsilon _v$')
1153            [xmin, xmax,ymin,ymax] = plt.axis()
1154            plt.axis([0, 1.2 * Load.ZC_time[-1],0,
1155                np.min([1.7*self.cyc['DeltaV'].values[-1],ymax])])
1156            plt.tight_layout()
1157
1158            plt.savefig(self.infile[:-4] +'-PF.png')
1159            plt.savefig(self.infile[:-4] +'-PF.pdf')
1160
1161        plt.close('all')
1162        # residual = observed - predicted
1163        self.resid = [(self.MaxNStrain - Damage[i][-1] * self.limEvstrain)
1164                   for i in [0,1,2]]
1165        self.perError = [self.resid[i] / self.MaxNStrain * 100.
1166               for i in [0,1,2]]
1167        if self.byrne:
1168            self.bresid = [self.MaxNStrain - bDamage[i][-1] for i in [0,1,2]]
1169            self.bperError = [self.bresid[i] / self.MaxNStrain * 100.
1170                   for i in [0,1,2]]
1171        if self.linear:
1172            self.lresid = [(self.MaxNStrain - lDamage[i][-1] * self.limEvstrain)
1173                   for i in [0,1,2]]
1174            self.lperError = [self.lresid[i] / self.MaxNStrain *100.
1175                   for i in [0,1,2]]
1176
1177
1178    def Drcorr(self):
1179        '''
1180        Corrects the relative density and void ratio since Qingsheng didn't
1181        keep track of volumetric strains while the confining load was being
```

```
        applied.
        '''
        df1 = pd.read_csv('/media/Storage/Documents/Python/SeismicComp/DrCorr.csv')
        df2 = df1[(df1['sigv_nom']==self.sigv_nom) & (df1['Dr_nom']==self.Dr_nom)]
        self.mass_ass = df2['mass'].values[0]
        self.Vstr_avg = df2['Vstr_avg'].values[0]
        self.Hcyc1 = self.hi - df2['deltaH'].values[0] # Assuming it was 24 mm high to
             begin.
        self.e_new = df2['e_new'].values[0]
        self.Dr_new = df2['Dr_new'].values[0]
        self.standev = df2['standev'].values[0]
        self.standevomean = df2['standevomean'].values[0]

    def calcAr(self, motion, dt, **kwargs):
        '''
        Calculates the following:
        Ia / (max(abs(motion))**2 * time)

        '''
        if 1 in motion.shape or len(motion.shape) == 1:
            NumMotions = 1
        else:
            NumMotions = np.min(motion.shape)
        Ar = np.zeros([NumMotions,], dtype=float)
        if NumMotions == 1:
            Ar = (calcRmsAccel(motion, dt, **kwargs) /
                       np.max(np.abs(motion))) # Assume data is in rows
        else:
            for i in xrange(NumMotions):
                Ar[i] = (calcRmsAccel(motion[i,:], dt, **kwargs) /
                       np.max(np.abs(motion[i,:]))) # Assume data is in rows

        return Ar

class SineTest(object):
    '''
    This object controls all the calcs and i/o for each test.
    Keyword          Definition
    file             name/path of the GCTS data file for the test.
    '''

    def __init__(self,**kwargs):
        infile = kwargs.get('infile')
        try:
            self.sigv_nom = int(re.search('soids/(.+?)kPa',infile).group(1))
            self.Dr_nom = int(re.search('kPa/Dr(.+?)/',infile).group(1))
        except AttributeError:
            print('AttributeError!!!!')

        self.byrne = kwargs.get('byrne',True)
        self.emin = kwargs.get('emin',0.68)
        self.emax = kwargs.get('emax',0.9)
        self.ncycles = np.ones((10,)) * -1
        self.lognorm = kwargs.get('lognorm', True)
        self.linear = kwargs.get('linear', True)
        self.verbose = kwargs.get('verbose',True)
        self.limEvstrain = kwargs.get('limEvstrain',0.01)
```

```
1238         self.Error = False
1239 
1240         if 'infile' == None:
1241             infile = raw_input('Nothing to do. Specify an input file:')
1242             self.csvimport(infile)
1243         else:
1244             self.csvimport(infile)
1245         self.infile = infile
1246         self.Drcorr()
1247         val = self.calcs()
1248     #self.scratch()
1249 #        self.dissen(verbose=True)
1250         if val == 0:
1251             return
1252         if self.byrne:
1253             self.ByrneCalcs()
1254     if self.lognorm:
1255         self.LogNormCalcs()
1256         self.plotter()
1257         self.calcNcycles()
1258 
1259     def csvimport(self, infile):
1260         '''
1261         This function pulls in the data from the GCTS output file.
1262         The GCTS file should not have the data separated into stages.
1263 
1264         '''
1265         self.data=[]
1266         self.label=[]
1267         self.name=[]
1268         self.Gs=[]
1269         self.mass=[]
1270         self.cyc_header=[]
1271         self.pist_area=[]
1272         footer = []
1273         datasection=0
1274         dimsec=0
1275         datafile = csv.reader(open(infile,'rU'))
1276         rowcount = -1
1277         for row in datafile:
1278             rowcount += 1
1279             if datasection==0:
1280                 if self.Gs:
1281                     pass
1282                 elif 'Specific Gravity: ' in row:
1283                     self.Gs=float(row[1])
1284 #                    self.Gs=2.66 #Because sometimes I forget to change the value in the
     GCTS software
1285                 if self.name:
1286                     pass
1287                 elif 'Specimen: ' in row:
1288                     self.name=row[1]
1289 
1290                 if self.mass:
1291                     pass
1292                 elif 'Dry Mass of Specimen: ' in row:
1293                     self.mass=float(row[1]); self.mass_unit=row[2]
```

```
                if self.mass==0:
                    self.mass=float(raw_input('Input the dry mass (g) for specimen %s:'
                    %self.name))
            elif 'Moist Mass of Specimen: ' in row:
                self.mass=float(row[1]); self.mass_unit=row[2]
                if self.mass==0:
                    self.mass=float(raw_input('Input the dry mass (g) for specimen %s:'
                    %self.name))

            if 'Type:' and 'Universal' in row:
                dimsec=1
            if 'Starting Date: ' in row:
                testdate = row[1]
            if 'Starting Time: ' in row:
                testtime = row[1]

        if dimsec==1:
            if 'Height: ' in row:
                self.hi=float(row[3])#*25.4 ;
                if row[4] == '(mm)':
                    pass
                else:
                    self.hi=self.hi*25.4
                self.hi_unit= 'mm' #Convert to mm
            if 'Diameter: ' in row:
                self.di=float(row[3])#*25.4;
                if row[4] == '(mm)':
                    pass
                else:
                    self.hi=self.hi*25.4
                self.di_unit= 'mm' #Convert to mm
                self.area = (self.di / 2) ** 2 * np.pi # in mm^2
                dimsec=0

        if len(row) >= 17:
            datasection = 1
            if row[0] == 'Time':
                header = rowcount
                self.headerlabel = row

            if row[0] == 'sec':
                self.units = row
        if datasection == 1:
            if row[0][:7] == 'Stage 2':
                footer = rowcount
                break
    if footer == []:
        df = pd.read_csv(infile,names=self.headerlabel,skiprows=header+3)
    else:
        df = pd.read_csv(infile,names=self.headerlabel,skiprows=header+3,
                         nrows = footer - header - 4)
    # Create a new dataframe that contains only the cyclic portion
    columns = ['Time', 'Cycles','Shear Load','Shear Displacement',
               'Normal Load','Normal Displacement',
               'Internal Shear Load','Internal Shear LVDT',
               'Internal Normal LVDT']
    try:
```

```
            x = df['Shear Load'] * 2.
        except KeyError:
            df['Shear Load'] = df['Internal Shear Load']
        try:
            self.cyc = df[columns].reset_index(drop=True)
        except KeyError:
            raise KeyError
        self.date_of_test = datetimeconvert(testdate,testtime)

    def calcs(self, **kwargs):
        '''
        SineTest
        This does the heavy lifting.
        '''
        # Calculate the volumes
        self.Vcyc_T1 = self.Hcyc1 * self.area # in mm^2
        if self.Gs == 2.65:
            pass
        else:
            Gs = self.Gs
            self.Gs = 2.65
            print('Changed Gs from {} to 2.65'.format(Gs))
        try:
            self.Vcyc_s = self.mass / (self.Gs * 0.001) # in mm^3 where density of water is
                0.001 g/mm^3
        except TypeError:
            self.Vcyc_s = self.mass_ass / (self.Gs * 0.001) # in mm^3 where density of
                water is 0.001 g/mm^3
            self.mass = self.mass_ass
            print('Assuming a mass of {}g'.format(self.mass_ass))
        self.cyc['Volume'] = self.Vcyc_T1 - (
                            self.cyc['Internal Normal LVDT'] * self.area)
        self.cyc['Vol_voids'] = self.cyc['Volume'] - self.Vcyc_s
        # Calc e
        self.cyc['e'] = self.cyc['Vol_voids'] / self.Vcyc_s
        self.e_0 = self.cyc['e'].values[0]
        self.e_1 = 0.99 * self.cyc['Vol_voids'].values[0] / self.Vcyc_s - 0.01
        self.Dr = (self.emax-self.e_0)/(self.emax-self.emin)*100
        self.cyc['Dr'] = ((self.emax-self.cyc['e'].values)/
                        (self.emax-self.emin)*100)

        #Calc changes in volume, damage, cycle ratio
        stresses = self.cyc['Internal Shear Load'].values / (self.area * 10 ** -6)
        self.cyc['SStress'] = stresses
        strains = self.cyc['Internal Shear LVDT'].values / self.Hcyc1
        self.cyc['SStrain'] = strains
        DE1 = np.zeros(stresses.shape, dtype=float)
        DE1[1:] = np.cumsum((stresses[1:] + stresses[:-1]) *
                 (strains[1:] - strains[:-1])) * 0.5
        self.cyc['DE'] = DE1
        self.cyc['DeltaV'] = (self.Vcyc_T1 - self.cyc['Volume']) / self.Vcyc_T1

        self.cyc['Dam'] = self.cyc['DeltaV'] / self.limEvstrain
        df = self.cyc[self.cyc['Dam']>1]
        try:
            self.Nlim = df['Cycles'].values[0]
            if self.Nlim < 4:
```

```
            print('Too few cycles to run some analyses')
            self.calcStrain()
            self.Error = True
            return 0
        except IndexError:
            print('Did not reach {}% Vol strain'.format(self.limEvstrain*100))
            self.Nlim = -1
            return 0
        self.cyc['ratio'] = self.cyc['Cycles'] / float(self.Nlim)
        self.cyc['r'] = (np.log(self.cyc['Dam'].values) /
                         np.log(self.cyc['ratio'].values))
        strain = self.cyc['Internal Shear LVDT'].values
        ind = np.where((((strain[:-1] < 0) & (strain[1:] >= 0)) |
               ((strain[:-1] > 0) & (strain[1:] <= 0))) &
               np.abs(strain[:-1]<0.03))
        self.cyc2 = self.cyc.ix[ind[0]]
        r = self.cyc2['r'].values
        D = self.cyc2['Dam'].values
        R = self.cyc2['ratio'].values
        stress = self.cyc2['Shear Load'].values
        self.cyc4 = self.cyc2[self.cyc2['ratio']<0.5]
        r2 = self.cyc4['r'].values
        D2 = self.cyc4['Dam'].values
        R2 = self.cyc4['ratio'].values
        ind5 = np.where(D < 0.8)
        ind5 = np.where(D < 0.8)
        self.rmean = np.mean(r[ind5])
        b = r[0]
        # Get the regression coefficients for various functional forms
        popt7, pov7= curve_fit(rn2, R, D)
        self.rn2_a, self.rn2_b = popt7[0],popt7[1]
        try:
            plt.plot(R,r,'ok', ms=3, alpha=0.5)
            popt2,pov2 = curve_fit(asymptote4,R2,r2,# p0=[r2[len(r2)/2], 0.4],
                   **dict(maxfev=10000))
            self.aa = 0.1
            self.ab = popt2[0]
            self.ac = popt2[0]
            plt.title('Asymptotic $r$')
            plt.plot(R,asymptote4(R,*popt2), '-k', lw=2,
                label=r'$r = {{ {:.3f} / R^{{ {:.3f} }} + {:.3f} }}$'.format(
                    self.aa, self.ac, self.ab))
        except:
            pdb.set_trace()
            self.aa = -1
            self.ab = -1
            self.ac = -1
        plt.axis([0,1.2,0,0.6])
        plt.xlabel('$R = n/N$', fontsize=15)
        plt.ylabel('$r$', fontsize=15)
        plt.legend(loc='best')
        plt.savefig(self.infile[:-4] +'-6.png')
        plt.close('all')

        cyc3 = self.cyc2[self.cyc2['Dam']<=1.25]
        popt2,pcov2 = curve_fit(curv_power, cyc3['ratio'].values,
                                cyc3['Dam'].values, p0=[0.3, 0.0001,0])
```

```
        popt3,pcov3 = curve_fit(power, cyc3['ratio'].values * 100,
                                cyc3['Dam'].values * 100)#, p0=[1.3, 0.000001,0])
        popt,pcov = curve_fit(curv_power, self.cyc2['ratio'].values,
                                self.cyc2['Dam'].values, p0=[0.3, 0.0001,0])
        self.r0 = popt2[0]
        self.s0 = popt2[1]
        self.t0 = popt2[2]
        self.r1 = popt[0]
        self.s1 = popt[1]
        self.t1 = popt[2]
        self.rpow = popt3[1]
        self.apow = popt3[0]

        # Calculate the strain
        self.calcStrain()
        if self.verbose:
            print('Test: %s, Dr=%.2f, Strain: %.2f per.,'
                %(self.name,self.Dr, self.gam*100) +
                ' Cycles to %.3f Vol strain: %.1f'
                %(self.limEvstrain, self.Nlim))
            print('   r: {:1.3f}, s: {:1.5f}, t:{:1.5f}'.format(
                        self.r0,self.s0,self.t0))
        return 1

    def calcStrain(self, **kwargs):
        '''
        Calculates the average shear strain over the majority of the cyclic portion.
        '''
        cycrange = kwargs.get('cycrange',75)
        # Get the zero crossings
        strain = self.cyc['Internal Shear LVDT'].values / self.Hcyc1
        ind = np.where((strain[:-1] < 0) & (strain[1:] >= 0))[0]
        cycles = int(len(ind)*cycrange/100) # Currently runs throughout the entire file.
             May want to limit this to 1% strain
        # Calc the max and min in each cycle
        maxs = np.zeros([cycles,],dtype=float)
        mins = np.zeros([cycles,],dtype=float)
        for i in xrange(cycles+1):
            if i == 0:
                continue
            maxs[i-1] = np.max(strain[ind[i-1]:ind[i]])
            mins[i-1] = np.min(strain[ind[i-1]:ind[i]])
        self.gam = np.mean(maxs - mins) / 2

    def DiscreteCycles(self, **kwargs):
        '''
        Modifies the cycle count so that partial cycles are given, and saved
        as 'CyclesD' in the dataframe.
        '''
        returns = kwargs.get('returns', False)
        cycles = self.cyc['Cycles'].values
        time = self.cyc['Time'].values
        ind = np.where((cycles[:-1] != cycles[1:]))
        cyclesD = np.zeros(cycles.shape, dtype=float)

        for i,val in enumerate(ind[0]):
            if i == 0:
```

```
1519                in1 = 0
1520                in2 = val + 1
1521                prevcyc = 0.
1522            else:
1523                in1 = ind[0][i-1] + 1
1524                in2 = val + 1
1525                prevcyc = cycles[in1] - 1.
1526            time2 = time[in1:in2] - time[in1]
1527            cyclesD[in1:in2] = prevcyc + time2 / time2[-1]
1528        in1 = ind[0][i-1] + 1
1529        in2 = len(time)
1530        prevcyc = cycles[in1] - 1.
1531        time2 = time[in1:in2] - time[in1]
1532        cyclesD[in1:in2] = prevcyc + time2 / time2[-1]
1533        self.cyc['CyclesD'] = cyclesD
1534        if returns:
1535            return cyclesD
1536
1537
1538     def scratch(self, **kwargs):
1539         '''
1540         A place to try new things without making a mess elsewhere.
1541         '''
1542 #        pdb.set_trace()
1543         cycles = self.cyc2['Cycles'].values
1544         cycles2 = self.DiscreteCycles(returns=True)
1545
1546         pdb.set_trace()
1547         D = self.cyc2['Dam'].values
1548         R = self.cyc2['ratio'].values
1549         plt.figure('scratch')
1550         plt.plot(R,D)
1551     popt, pov = curve_fit(curvPL, R, D)
1552         print(popt)
1553         plt.plot(R, curvPL(R,*popt), label='CurvPL fit')
1554     popt2, pov2 = curve_fit(logcdf, R,D)
1555     print(popt2)
1556     plt.plot(R, logcdf(R, *popt2), label='logcdf')
1557         plt.legend()
1558         plt.show()
1559         pdb.set_trace()
1560         ind15 = np.where(cycles==16)[0][0]
1561         D15 = D[ind15]
1562
1563         Dn = D / D15
1564
1565         plt.plot(cycles[:ind15],Dn[:ind15],'-o')
1566         print(cycles[:ind15])
1567         print(Dn[:ind15])
1568         pdb.set_trace()
1569         plt.show()
1570
1571
1572     def LogNormCalcs(self, **kwargs):
1573     '''
1574     Calculates the parameters mu and sig for the log-normal-cdf-fit
1575     of the D-R curve.
```

```
    '''
        R1 = self.cyc2['ratio'].values
    R = R1[R1<1.6]
        D = self.cyc2['Dam'].values[R1<1.6]
    popt, pov = curve_fit(logcdf, R, D)
    plt.figure('LogNormCDF Method')
    plt.plot(R, D, 'o')
    plt.plot(R, logcdf(R, *popt), '-')
    plt.xlabel(r'$R = n /N$')
    plt.ylabel(r'$D$')
    plt.title(r'Log Norm Fit: $\mu =$ {:.3f}, $\sigma =$ {:.3f}'.format(
          0, popt[0]))
    self.sig = popt[0]
        self.mu = 0.

     def ByrneCalcs(self, **kwargs):
        '''
        Calculates parameters C1 and C2 for the Byrne model.
        (Byrne, Peter. 1991. "A cyclic shear-volume coupling and pore pressure
        model for sand." Proceedings: Second International Conference on Recent
        Advances in Geotechnical Earthquake Engineering and Soil Dynamics,
        March 11-15, 1991, St. Louis, Missouri. Paper 1.24)
        '''
        def byrneexp(x,c1,c2):
            '''
            returns \Delta \epsilon_v / \gamma
            '''
            return c1 * np.exp(-c2 * x)

        sstrain = self.cyc['SStrain'].values
        ind = np.where((sstrain[:-1] < 0) & (sstrain[1:] >= 0))[0]
        DV = self.cyc['DeltaV'].values
        vstrain = np.zeros(ind.shape, dtype=float)
        for i,index in enumerate(ind):
            try:
                if (i == 0):
                    if index == 0:
                        vstrain[i] = 0
                    else:
                        vstrain[i] = np.max(DV[:index])
                else:
                    vstrain[i] = np.max(DV[ind[i-1]:index])
            except ValueError:
                 pdb.set_trace()
        dvstrain = np.zeros(vstrain.shape, dtype=float)
        dvstrain[1:] = np.diff(vstrain)
        evogam = vstrain / self.gam
        devogam = dvstrain / self.gam
        # Now, to the parameters:
        C1 = dvstrain[1] / self.gam
        popt, pcov = curve_fit(byrneexp, evogam[1:], devogam[1:], p0=[C1, 0.4 / C1])

        plt.figure('Byrne\'s Method')
        plt.plot(evogam, devogam, 'ko', ms=6, label='Data')
        plt.plot(evogam,byrneexp(evogam, *popt), 'k',lw=2, label='Best Fit:\n '+
               'C1={:.2f},\n C2={:.2f}'.format(*popt))
        plt.legend(loc='best')
```

```
1633         plt.xlabel(r'$\epsilon_v / \gamma}$')
1634         plt.ylabel(r'$\Delta \epsilon_v / \gamma}$')
1635         self.C1, self.C2 = popt[0],popt[1]
1636 
1637 #    def dissen(self,**kwargs):
1638 #        '''
1639 #        Calculates the dissipated energy from the test
1640 #        '''
1641 #        verbose = kwargs.get('verbose',True)
1642 #        stresses = self.cyc['sstress_int'].values # may want to change to internal
1643 #        strains = self.cyc['sstrain_int'].values
1644 #        maxind = self.cycind
1645 #        dissEn = np.cumsum((stresses[1:maxind] + stresses[:maxind-1]) *
1646 #                (strains[1:maxind] - strains[:maxind-1])) * 0.5
1647 #        dissEn2 = np.zeros([len(self.cyc.index),],dtype=float)
1648 #        dissEn2[1:maxind] = dissEn
1649 #        self.DE = dissEn[-1]
1650 #        dissEn2[maxind:] = self.DE
1651 #        self.cyc['DE'] = dissEn2
1652 #
1653 #        # Calculate the liquefaction-corrected dissipated energy
1654 #
1655 #        ind = np.where((stresses[:maxind-1] < 0) &
1656 #                   (stresses[1:maxind] >= 0))
1657 #        x = self.cyc['Cycles'].values[ind] / float(self.cycles)
1658 #        y = self.cyc['DE'].values[ind]
1659 #        A = np.vstack([x[:-len(x)/2], np.ones(len(x)/2)]).T
1660 #        m, b = np.linalg.lstsq(A, y[:-len(x)/2])[0]
1661 #        self.DE_corr = m + b
1662 #        self.DE_m = m
1663 #        self.DE_b = b
1664 #        self.DE_x = x
1665 #        self.DE_y = y
1666 #
1667 #        # Calculate the PEC (Green's Dissertation, eq. 4-15)
1668 #        Ru = self.cyc['Ru'].values
1669 #        ind = np.where(Ru > 0.65)
1670 #
1671 #        self.PEC = self.cyc['DE'].values[ind[0][0]] / 0.4225
1672 #
1673 #        if verbose:
1674 #            print('Total DE: %.4f, Corrected DE: %.4f'%(self.DE,self.DE_corr))
1675 
1676     def plotter(self, **kwargs):
1677         '''
1678         Produces plots of interest.
1679 
1680         Keyword      Definition
1681         save           True [default] or False
1682         savetype       png or pdf, etc
1683         show           Should the figures be shown? True or False
1684         '''
1685         save = kwargs.get('save',True)
1686         savetype = kwargs.get('savetype','png')
1687         show = kwargs.get('show',False)
1688         fontsize = kwargs.get('fontsize',15)
1689         lw = 5
```

```
        if kwargs.get('latex', True):
            fs = kwargs.get('fs',18)
            import matplotlib
            matplotlib.rcParams.update({'font.size': fs,
                'font.family': 'STIXGeneral', 'mathtext.fontset': 'stix'})
            fontsize=18
        else:
            import matplotlib
            matplotlib.rcParams.update({'font.size': 12,
                'font.family': 'sans', 'text.usetex': False})

        # Time plot
        plt.figure('Results of %s'%self.name)
        plt.subplot(121) # This stopped working in the gcts-cv-css code
        if not PaperFigures:
            plt.title(
                   '$D_r$=%.1f%%, $\gamma_{cyc}$=%.2f%%, $N_{%.1f}$=%.1f'%(
                      self.Dr,self.gam*100,self.limEvstrain*100,self.Nlim))

        plt.plot(self.cyc['SStrain'].values,
                 self.cyc['Shear Load'].values / (self.area * 10 ** -6),
                 '-k', lw=.8, alpha=0.7)
        plt.ylabel('Shear Stress (kPa)', fontsize=fontsize)
        plt.xlabel('Shear Strain', fontsize=fontsize)

        plt.setp(plt.subplot(121).get_xticklabels(), rotation=90)


        plt.subplot(122)
        try:
            plt.plot(self.cyc['SStrain'].values,
                     self.cyc['DeltaV'].values,
                     '-k', lw=.8, alpha=0.7)
        except:
            pdb.set_trace()
            self.cyc.plot(x='Shear Strain',
                     y='DeltV')

        plt.xlabel(r'Shear Strain',fontsize=fontsize)
        plt.ylabel(r'Volumetric Strain', fontsize=fontsize)
        if PaperFigures:
            [xmin, xmax, ymin, ymax] = plt.axis()
            plt.axis([xmin, xmax, ymax, ymin])
        plt.setp(plt.subplot(122).get_xticklabels(), rotation=90)
        plt.tight_layout()

        plt.figure('2')
        if self.Nlim < 4:
            plt.plot(self.cyc['ratio'].values,self.cyc['Dam'].values,'o',
                    label='Data')

        else:
            lw = 1.6
            plt.plot(self.cyc2['ratio'].values[::3],
                   self.cyc2['Dam'].values[::3],'ok',
                    label='Data', ms=7, alpha=0.4)
            if self.limEvstrain < 0.01:
```

```
                pass
            else:
                plt.plot(self.cyc2['ratio'].values,
                        curv_power(self.cyc2['ratio'].values,self.r0, self.s0,
                                   self.t0), '-k', lw=lw,
                        label=r'$R^{{ {:.2f} + {:.3f}R + {:.3f}R^2 }}$'.format(self.r0,
                               self.s0, self.t0))
            plt.plot(self.cyc2['ratio'].values,
                   self.cyc2['ratio'].values ** self.rmean, 'k-.',lw=lw,
                   label=r'$R^{{ {:.3f} }}$'.format(
                       self.rmean))

            if self.aa != -1:
               plt.plot(self.cyc2['ratio'].values,
                   self.cyc2['ratio'] ** asymptote(self.cyc2['ratio'].values,self.aa,
                        self.ab,
                              self.ac), 'k:', lw=lw,
                   label=r'$R^{{ {:.3f} / R^{{ {:.3f} }} + {:.3f} }}$'.format(self.aa,
                   self.ac, self.ab))
        if self.lognorm:
          plt.plot(self.cyc2['ratio'].values,
             logcdf(self.cyc2['ratio'].values, self.sig),
             'k--', lw=lw,
                        label=r'$1 + erf((\log(R)-{:.1f})/{:.1f})$'.format(
              self.mu, self.sig))
          plt.ylabel(r'Damage = $\frac{\varepsilon _v}{%.1f \%%}$'%(self.limEvstrain*100),
               fontsize=fontsize)
          plt.xlabel(r'$R=\frac{n}{N}$', fontsize=fontsize)
          plt.axis([0,1.1,0,1.1])
          plt.legend(loc='lower right', numpoints=1)
          plt.tight_layout()
          if show:
              plt.show()
          if save:
              import matplotlib
              figures=[manager.canvas.figure for manager in
                 matplotlib._pylab_helpers.Gcf.get_all_fig_managers()]
              for i, figure in enumerate(figures):
                  figure.savefig(self.infile[:-4] +'-'+str(i+1)+'.'+savetype)
                  if PaperFigures:
                      figure.savefig(self.infile[:-4] +'-'+str(i+1)+'.'+'pdf')
          plt.close('all')

      def Drcorr(self):
          '''
          COrrects the relative density and void ratio since Qingsheng didn't
          keep track of volumetric strains while the confining load was being
          applied.
          '''
          df1 = pd.read_csv('DrCorr.csv')
          df2 = df1[(df1['sigv_nom']==self.sigv_nom) & (df1['Dr_nom']==self.Dr_nom)]
          self.mass_ass = df2['mass'].values[0]
          self.Vstr_avg = df2['Vstr_avg'].values[0]
          self.Hcyc1 = self.hi - df2['deltaH'].values[0] # Assuming it was 24 mm high to
               begin.
          self.e_new = df2['e_new'].values[0]
          self.Dr_new = df2['Dr_new'].values[0]
```

```
1802         self.standev = df2['standev'].values[0]
1803         self.standevomean = df2['standevomean'].values[0]
1804 
1805     def calcNcycles(self):
1806         '''
1807         Calculates the number of cycles to various volumetric strains.
1808         From 0.001 to 0.01.
1809         '''
1810         strains = 0.001 * np.arange(1,11)
1811         VolStrain = self.cyc2['DeltaV'].values
1812         Cycles = self.cyc2['Cycles'].values
1813         for i, ev in enumerate(strains):
1814             try:
1815                 self.ncycles[i] = Cycles[np.where(VolStrain >= ev)[0][0]]
1816             except IndexError:
1817                 self.ncycles[i] = -1
1818 
1819 def runsine(*args,**kwargs):
1820     '''
1821     Takes the output files from the GCTS and runs them through SineTest
1822     '''
1823     h5 = kwargs.get('h5file','default')
1824     infilelist=[]
1825     if len(args) > 0:
1826         argsin = args
1827     else:
1828         argsin=sys.argv[1:]
1829 #    argsin.append('/media/Storage/Documents/Python/SeismicComp/Sinusoids/50kPa/Dr30/1.csv')
1830     if kwargs.get('runall', False):
1831         rootdir = '/media/Storage/Documents/Python/SeismicComp/Sinusoids/'
1832         for root, dirnames, filenames in os.walk(rootdir):
1833             for filename in fnmatch.filter(filenames, '*.csv'):
1834                 argsin.append(os.path.join(root, filename))
1835 
1836     numtests = len(argsin)
1837     if argsin==[]:
1838         filters = 'CSV (*.csv)' # for qtfiledialog
1839         filelist=fd.fileopen(message='Choose a gcts output...',
1840                              filters=filters,
1841                              defaultDir=(u'/media/Storage/Documents/Python/'+
1842                              'SeismicComp/Sinusoids/'))
1843         for file in filelist:
1844             if file == '/media/Storage/Documents/Python/SeismicComp/DrCorr.csv':
1845                 continue
1846             infilelist.append(file)
1847         del filelist
1848     elif len(argsin)>0:
1849         if argsin[0] in ['all','--all','-a','a']:
1850             infilelist=glob.glob('*.csv')
1851         else:
1852             for file in argsin:
1853                 if file == '/media/Storage/Documents/Python/SeismicComp/DrCorr.csv':
1854                     continue
1855                 infilelist.append(file)
1856     del argsin
1857     numtests = len(infilelist)
1858     if h5 == 'choose':
```

```
        h5file=fd.fileopen(message='Choose a Summary Table...',
                           filters='h5|*.h5',
                           defaultDir=(u'/media/Storage/Documents/Python'+
                           '/GCTS/ConstantVolume'))
        h5table = HDF5saver(filename=h5file[0])
    else:
       h5table = HDF5saver()
    for infile in infilelist:
        print(infile)
        try:
            testobj = SineTest(infile=infile)
        except KeyError:
            continue
        if testobj.Nlim < 0:
            continue
        if testobj.Error:
            continue
        h5table.input(testobj)

        del testobj
    print(numtests)
    h5table.close()

def runEQ(*args,**kwargs):
    h5 = kwargs.get('h5file','default')
    infilelist=[]
    if len(args) > 0:
        argsin = args
    else:
        argsin=sys.argv[1:]
#    argsin.append('/media/Storage/Documents/Python/SeismicComp/EQmotions/100kPa/Dr30/Step
     loadin/97.csv')
    if os.name == 'posix':
        rootdir = '/media/Storage/Documents/Python/SeismicComp/EQmotions/'
    else:
        rootdir = 'D:\\Documents\\Python\\SeismicComp\\EQmotions'
    for root, dirnames, filenames in os.walk(rootdir):
        for filename in fnmatch.filter(filenames, '*.csv'):
            argsin.append(os.path.join(root, filename))

    if argsin==[]:
        filelist = []
        for file in filelist:
            if file == '/media/Storage/Documents/Python/SeismicComp/DrCorr.csv':
                continue
            infilelist.append(file)
        del filelist
    elif len(argsin)>0:
        if argsin[0] in ['all','--all','-a','a']:
            infilelist=glob.glob('*.csv')
        else:
            for file in argsin:
                if file == '/media/Storage/Documents/Python/SeismicComp/DrCorr.csv':
                    continue
                infilelist.append(file)
    del argsin
    if h5 == 'choose':
```

```
        h5file=fd.fileopen(message='Choose a Summary Table...',
                           filters='h5|*.h5',
                           defaultDir=(u'/media/Storage/Documents/Python'+
                           '/GCTS/ConstantVolume'))
        h5table = HDF5saver(filename=h5file[0])
    else:
       h5table = HDF5saver(testtype='EQ')
    for infile in infilelist:
        print(infile)
        try:
            testobj = EQTest(infile=infile)
        except KeyError:
            continue
        h5table.input(testobj)

        del testobj
    h5table.close()

def PMcalibrator(**kwargs):
    '''
    This function fit curves for N to a range of volumetric strains.
    '''
    import statsmodels.api as sm
    import pandas as pd
    formula = kwargs.get('formula', 'np.log(N) ~ np.log(gam) + sigv_norm + Dr')
    noterms = formula.count('+') + 2
    infile = kwargs.get('infile', 'SummaryResults.h5')
    outfile = kwargs.get('outfile', 'PMCalResults.h5')
    strains = 0.001 * np.arange(1,11)
    class PMtable(pt.IsDescription):
        VolStrain = pt.Float32Col(pos=1)
        formula = pt.StringCol(itemsize=128, pos=2)
        Coeffs = pt.Float32Col(shape=(noterms,), pos=3)
        R2adj = pt.Float32Col(pos=4)

    f = pt.open_file(infile, 'r')
    table = f.root.Summary
    cyc = table.read(field='ncycles')
    ncycles = cyc.reshape(-1,10)
    data0 = np.vstack((
                   table.read(field='Nlim'),
                   table.read(field='gam'),
                   table.read(field='sigv_nom')/101.325,
                   table.read(field='Dr'))).T
    data = pd.DataFrame(data0, columns=['Nlim','gam','sigv_norm','Dr'])
    # Clean up:
    f.close()
    del table, data0

    f = pt.open_file(outfile, 'w', title='PM Calibration Parameters')
    outtable = f.create_table('/', 'PMresults', PMtable)

    for i, strain in enumerate(strains):
        print('\n\n\n Strain={}\n\n'.format(strain))
        data['N'] = ncycles[:,i]
        model = sm.formula.ols(formula=formula, data=data)
        results = model.fit()
```

```
        print(results.summary())

        row = outtable.row
        row['VolStrain'] = strain
        row['formula'] = formula
        row['Coeffs'] = results.params
        row['R2adj'] = results.rsquared_adj
        row.append()
        outtable.flush()

    f.close()

class Ncycles(object):
    '''
    Calculates the number of cycles to any given volumetric strain between
    0.001 and 0.01.
    '''
    def __init__(self,**kwargs):
        calfile = kwargs.get('calfile',
                '/media/Storage/Documents/Python/SeismicComp/PMCalResults.h5')
        try:
            self.f = pt.open_file(calfile, 'r')
        except AttributeError:
            self.f = pt.openFile(calfile, 'r')
        self.table = self.f.root.PMresults
        self.strains = self.table.read(field='VolStrain')
        self.formula = self.table.read(field='formula')
        Coeffs = self.table.read(field='Coeffs')
        self.Coeffs = Coeffs.reshape(-1,len(Coeffs[0]))
        self.table.close()
        self.f.close()

    def formula(self):
        print(self.formula)

    def __call__(self,Target_ev, *args, **kwargs):
        if len(args) == 0:
            args = self.oldargs
        else:
            self.oldargs = args

        #  interpolate between periods if necesary
        if np.round(Target_ev,4) not in self.strains:
            lowind = np.where(Target_ev > self.strains)[0][-1]
            T_low = self.strains[lowind]
            T_hi = self.strains[lowind + 1]
            N_low = self.__call__(T_low, *args)
            N_hi = self.__call__(T_hi, *args)
            x = np.array([T_low, T_hi])
            Y_N = np.array([N_low, N_hi])
            try:
                f_N = interp1d(x, Y_N, axis=0)
                try:
                    del N
                except:
                    pass
                N = f_N(Target_ev)
```

```
                return N
            except ValueError:
                pdb.set_trace()
        else:
            i = np.abs(self.strains - Target_ev).argsort()[0] # Identify the strain index
            mults = [1.]
            mults.extend(args)
            if np.all([np.isscalar(val) for val in mults]):
                return np.sum(self.Coeffs[i] * np.array(mults))
            else:
                num = np.max([1 if np.isscalar(val)
                                else len(val) for val in mults])
                return np.sum(self.Coeffs[i][:, np.newaxis] *
                              np.array([val * np.ones((num,), dtype=float)
                                        if np.isscalar(val) else val
                                        for val in mults]), axis=0)

def selector():
    '''
    This is where you might want to start
    '''
    x = None

    while x not in [1,2,3,4,5,6,7]:
        print('Run Options:\n 1. Select a GCTS soil output\n' +
              ' 2. Select a GCTS soil output and choose .h5 file\n' +
              ' 3. Select a GCTS water output\n' +
              ' 4. Plot all data\n' +
              ' 5. Import all data and go to a prompt\n'
              ' 6. Delete some test results (not implemented)\n' +
              ' 7. Plot selected test results (not implemented)\n' +
              ' 8. Quit')
        selection = raw_input('Please make a selection: ').split()

        try:
            x = int(selection[0])
        except IndexError:
            x = None


        if x == 1:
            runsoil(h5file='default')
        elif x == 2:
            runsoil(h5file='choose')
        elif x == 3:
            runwater()

        elif x == 4:
            runsummary()

        elif x == 5:
            dataprompt()

        elif x == 7:
            break
        else:
            print('I didn\'t understand you.')
```

```
2086
2087
2088 class LogNormInc(object):
2089     '''
2090     Calculates the value of volumetric strain using the LogNormal fun.form.
2091     Returns not the increment, but the next value of volumetric strain.
2092     It is assumed that each shear strain is one half cycle.
2093     '''
2094     def __init__(self, Cycles, Amp, **kwargs):
2095         from scipy.special import erf, erfinv
2096         self.erf = erf
2097         self.erfinv = erfinv
2098         self.name = kwargs.get('name')
2099         self.Dr = kwargs.get('Dr')
2100         self.sigv = kwargs.get('sigv', 100.)
2101         self.mu = kwargs.get('mu', 0.0)
2102    self.threshold = kwargs.get('treshold', 0.0002)
2103         self.Amp = Amp
2104         self.Cyc = Cycles
2105         self.N = np.exp(-2.97272 * np.log(self.Amp) + 0.002292 * self.sigv
2106                + 0.02435 * self.Dr - 13.874)
2107         self.N[self.Amp<self.threshold] = np.inf
2108         self._R = Cycles / self.N
2109         self.sig = np.exp(-0.578 * np.log(self.Amp) + 0.0012 * self.sigv +
2110                     0.0064 * self.Dr - 2.3598)
2111         self.D = np.zeros(Amp.shape, dtype=float)
2112         self.R = np.cumsum(self._R)
2113         self.i = -1
2114
2115     def __call__(self, **kwargs):
2116         '''
2117         This is an iterator function.
2118         '''
2119         erf = self.erf
2120         erfinv = self.erfinv
2121         returns = kwargs.get('returns', True)
2122         self.i += 1
2123         if self.Amp[self.i] < self.threshold:
2124             self.D[self.i] = self.D[self.i-1]
2125         else:
2126             if self.i == 0:
2127                 self.D[self.i] = 1 + erf((np.log(self.R[self.i]) - self.mu) /
2128                                          self.sig[self.i])
2129             else:
2130                 self.D[self.i] = (self.D[self.i-1] +
2131                           erf((np.log(self.R[self.i]) - self.mu) /
2132                                       self.sig[self.i]) -
2133                           erf((np.log(self.R[self.i-1]) - self.mu) /
2134                                       self.sig[self.i]))
2135         if returns:
2136             return self.D[self.i]
2137
2138     def run(self):
2139         '''
2140         This just runs thru all the calls
2141         '''
2142         for val in self.Amp:
```

```
            self.__call__(returns=False)

def PMtest():
    Ncalc = Ncycles()
    print(Ncalc.formula[0])
    Dr = 45.
    sigv_norm = 1.
    gam = 0.004

    MyNcycles = np.zeros(Ncalc.strains.shape)

    for i, ev in enumerate(Ncalc.strains):
        MyNcycles[i] = np.exp(Ncalc(ev, np.log(gam), sigv_norm, Dr))
    print(MyNcycles)
    for stra in [0.0015, 0.0095]:
        print('Strain: {}, N={}'.format(stra, np.exp(Ncalc(stra))))
    pdb.set_trace()

def PMresulttable(**kwargs):
    '''
    Prints out a table summarizing the results of the PMcalibrator.
    '''
    import pandoctable as pan
    calfile = kwargs.get('calfile', 'PMCalResults.h5')
    f = pt.open_file(calfile, 'r')
    table = f.root.PMresults
    strains = table.read(field='VolStrain')
    formula = table.read(field='formula')
    Coeffs0 = table.read(field='Coeffs')
    Coeffs = Coeffs0.reshape(-1,len(Coeffs0[0]))
    Rsqadj = table.read(field='R2adj')
    table.close()
    f.close()

    heads = ['$\\varepsilon_v$ (%)', '$\phi_1$',
           '$\phi_2$', '$\phi_3$', '$\phi_4$', '$R^2_{adj.}$']
    align = [('^', '^'), ('^', '^'), ('^', '^'), ('^', '^'),
           ('^', '^'), ('^', '^')]
    fields = range(6)

    data = []
    for i,stra in enumerate(strains):
        data.append((np.round(stra*100.,1),
               round(Coeffs[i,0],2),
               round(Coeffs[i,1],3),
               round(Coeffs[i,2],3),
               round(Coeffs[i,3],4),
               round(Rsqadj[i],2)))

    pan.table(sys.stdout, data, fields, heads, align)
```

J.10 Implementation of ASTM 'Cycle Counting' Methods

The file `CycleCounting.py` returns cycle and amplitude pairs from an irregular motion.

,

```
1
2  import pdb
3  import numpy as np
4
5  class Load(object):
6      '''
7      Accepts a 1D array of values.
8      Returned Amplitudes are single amplitudes.
9      '''
10
11     def __init__(self, load, **kwargs):
12
13         self.mean = kwargs.get('mean', 0.) # <<======================== Do I want to
               auto-set as zero?
14         if self.mean == 'mean':
15             self.mean = np.mean(load)
16
17         self.dt = kwargs.get('dt', 1.)
18         self.time = np.linspace(0,(len(load)-1) * self.dt, len(load))
19         self.load = load
20         self.PRthrshld = kwargs.get('PRthrshld', 0.)
21
22         if kwargs.get('AriasRatio', False):
23             self.Ar = self.calcAriasRatio(load, self.dt)
24
25         runall = kwargs.get('runall', True)
26         if runall == True:
27             self.calcZeroCrossing(returns=False)
28             self.calcPeak2Peak(returns=False)
29             self.calcRainflow(returns=False)
30             self.calcPR(thrshld=self.PRthrshld)
31
32     def countCycles(self, *args, **kwargs):
33         if (len(args) == 0) or ('all' in args):
34             args = ['ZC', 'PP', 'RF']
35         kwargs['returns'] = False
36         if 'ZC' in args:
37             self.calcZeroCrossing(**kwargs)
38         if 'PP' in args:
39             self.calcPeak2Peak(**kwargs)
40         if 'RF' in args:
41             self.calcRainflow(**kwargs)
42
43
44
45
46     def calcAriasRatio(self, motion, dt):
47        '''
```

```
        Calculates the following:
        Ia / (max(abs(motion))**2 * time)

        '''

        if len(motion.shape) == 2:
            return (np.trapz(motion ** 2,dx=dt,axis=1) /
                   np.max(np.abs(motion),axis=1) ** 2 /
                   (dt * max(motion.shape)))

        else:
            return (np.trapz(motion ** 2,dx=dt) / np.max(np.abs(motion)) ** 2 /
                   (dt * len(motion)))


    def calcZeroCrossing(self, **kwargs):
        '''
        Calculates the number of cycles using the zero crossings to separate
        cycles. This is the same as the mean crossing peak counting in
        section 5.2.2 of ASTM 1049 - 85 (2011).
        '''
        # This AmpType is here for ease of performing my seismic compression
        # calculations. To correspond with the cycle count of my sinusoidal
        # motions, I needed to divide the cycles by two. But this is not
        # the ASTM spec advises.
        AmpType = kwargs.get('AmpType', 'SC')
        if AmpType == 'SC':
            divisor = 2.
        else:
            divisor = 1.
        returns = kwargs.get('returns', True)
        zero = self.mean
        load = self.load
        time = self.time
        ind = np.where(((load[:-1] >= zero) & (load[1:] < zero)) |
               ((load[:-1] < zero) & (load[1:] >= zero)) )
        Amp = np.zeros(len(ind[0]), dtype=float)
        Cycles = np.ones(len(ind[0]), dtype=float) / divisor
        time2 = np.zeros(len(ind[0]))
        for i,ind2 in enumerate(ind[0]):
            if i == 0:
                if ind2 == 0:
                    Amp[i] = 0.
                    time2[i] = time[ind2]
                    continue
                else:
                    Amp[i] = np.max(np.abs(load[:ind[0][i]]))
                    time2[i] = time[np.argmax(np.abs(load[:ind[0][i]]))]

            else:
                Amp[i] = np.max(np.abs(load[ind[0][i-1]:ind[0][i]]))
                time2[i] = time[ind[0][i-1] +
                       np.argmax(np.abs(load[ind[0][i-1]:ind[0][i]]))]

        self.ZC_Amp = Amp
        self.ZC_Cyc = Cycles
        self.ZC_time = time2
```

```
105
106         if returns == True:
107             return Amp, Cycles
108
109
110
111     def calcPeak2Peak(self, **kwargs):
112         '''
113         Calculates the cumber of cycles using peak to peak.
114         This is the Simple-Range Counting Method from Section 5.3 of ASTM
115         1049 - 85 (2011) where both positive and negative ranges are counted.
116         Optional Keyword arguments:
117         returns =  True [default] or False
118         thrshld = 0 [default]. Threshold load under which no cycles are
119         counted.
120         '''
121         returns = kwargs.get('returns', True)
122         thrshld = kwargs.get('thrshld', 0.)
123         # This AmpType is here for ease of performing my seismic compression
124         # calculations. To correspond with the $\gamma$ of my sinusoidal
125         # motions, I needed to divide the Amplitudes by two. But this is not
126         # the ASTM spec advises.
127         AmpType = kwargs.get('AmpType', 'SC')
128         if AmpType == 'SC':
129             divisor = 2.
130         else:
131             divisor = 1.
132         zero = 0.
133         load = self.load
134         time = self.time
135         dLoad = np.diff(load)
136         ind = np.where(np.sign(dLoad[:-1]) != np.sign(dLoad[1:]))
137         ind2 = ind[0] + 1
138         peaks = load[(ind2)]
139         Amp = np.abs(peaks[1:] - peaks[:-1]) / divisor
140         Cycles = np.ones(Amp.shape, dtype=float) * 0.5
141         time2 = time[ind2[:-1]]
142         ind4 = np.where(Amp != 0)
143         Amp = Amp[ind4]
144         Cycles = Cycles[ind4]
145         time2 = time2[ind4]
146         ind3 = np.where(np.abs(Amp) >= thrshld)
147         self.PP_Amp = Amp[ind3[0]]
148         self.PP_Cyc = Cycles[ind3[0]]
149         self.PP_time = time2[ind3[0]]
150         if returns == True:
151             return self.PP_Amp, self.PP_Cyc
152
153
154     def calcRainflow(self, **kwargs):
155         '''
156         The Rainflow counting method as described in Section 5.4.4 of ASTM
157         1049 - 85 (2011).
158         '''
159         returns = kwargs.get('returns', True)
160         # This AmpType is here for ease of performing my seismic compression
161         # calculations. To correspond with the $\gamma$ of my sinusoidal
```

```
        # motions, I needed to divide the Amplitudes by two. But this is not
        # the ASTM spec advises.
        AmpType = kwargs.get('AmpType', 'SC')
        if AmpType == 'SC':
            divisor = 2.
        else:
            divisor = 1.
        zero = self.mean
        load = self.load
        time = self.time
        dLoad = np.diff(load)
        ind = np.where(((dLoad[:-1] >= zero) & (dLoad[1:] < zero)) |
               ((dLoad[:-1] < zero) & (dLoad[1:] >= zero)) )
        ind2 = ind[0] + 1
        peaks = load[(ind2)]
        t = time[(ind2)]
        Amp = []
        Cyc = []
        del ind


        S = 0
        ind = [0,1,2]
        indorder = []

        while ind[-1] < len(peaks):
            # Step 2 and 1
            if len(ind) < 3:
                ind.append(ind[-1]+1)
            try:
                Y = np.abs(peaks[ind[-3]] - peaks[ind[-2]])
                X = np.abs(peaks[ind[-2]] - peaks[ind[-1]])
            except IndexError:
                break

            if X < Y: # Step 3a
                ind.append(ind[-1] + 1)
                continue
            else:   # Step 3b
                if S == ind[-3]: # Step 5
                    Amp.append(Y)
                    Cyc.append(0.5)
                    indorder.append(ind[-2])
                    S = ind[-2]
                    ind.remove(ind[-3])
                    continue
                else: # Step 4
                    Amp.append(Y)
                    Cyc.append(1.)
                    ind.remove(ind[-3])
                    ind.remove(ind[-2])
                    indorder.append(ind[-2])
                    continue

        #Step 6
        for val in ind:
            if val == ind[0]:
```

```
                val0 = val
                continue
            elif val == ind[-1]:
                break
            else:
                Amp.append(np.abs(peaks[val] - peaks[val0]))
                Cyc.append(0.5)
                val0 = val
                indorder.append(val)
        time2 = t[indorder]
        time2, Amp, Cyc = zip(*sorted(zip(time2,Amp,Cyc)))

        self.RF_Amp = np.array(Amp) / divisor
        self.RF_Cyc = np.array(Cyc)
        self.RF_time = np.array(time2)
        if returns == True:
            return self.RF_Amp, self.RF_Cyc


    def plotter(self, figname=None, **kwargs):
        '''
        A plotting function for convenience of testing.
        '''

        import matplotlib.pyplot as plt
        if figname == None:
            plt.figure()
        else:
            plt.figure(figname)
        plt.plot(self.time, self.load, 'b', label='Input Loading')
        plt.plot(self.PP_time, self.PP_Amp, 'rx', label='PP')
        plt.plot(self.RF_time, self.RF_Amp, 'c^', label='RF')
        plt.plot(self.ZC_time, self.ZC_Amp, 'g+', label='ZC')
        plt.legend(loc='best')
        plt.xlabel('Time')
        plt.ylabel('Amp')
        plt.show()


    def calcPR(self, **kwargs):
        '''
        Calculates the ratio of the peak of the motion to the
        average peak size.

        Ignores any peaks below the threshold
        '''
        thrshld = kwargs.get('thrshld', 0.)
        returns = kwargs.get('returns', False)
        self.PR = {}

        def mycalculator(Amps, Cycles, thrshld):

            ind = np.where(np.abs(Amps) >= thrshld)[0]
            if len(ind) == 0.:
                return 0
            else:
                return (np.max(np.abs(Amps)) /
```

```
                    np.average(Amps[ind], weights=Cycles[ind]))

        myAmps = ['ZC_Amp', 'PP_Amp', 'RF_Amp']
        myCycs = ['ZC_Cyc', 'PP_Cyc', 'RF_Cyc']
        for cyc,amp in zip(myCycs, myAmps):
            try:
                thisamp = self.__dict__[amp]
                thiscyc = self.__dict__[cyc]
            except ValueError:
                continue
            self.PR[amp[:2]] = mycalculator(thisamp, thiscyc, thrshld)
        if returns == True:
            return self.PR


if __name__ == '__main__':
    import EquivLin as el
#
    motion, NFFT, FA, f, w, NPTS, dt = el.getInMotion('NGA_no_41_ORR021.AT2')
#    motion = np.array([1,-2,1,-3,5,-1,3,-4,4,2])
#    time = np.linspace(0, 15, 200)
#    motion = np.sin(time * 5)
    dt = 0.5
    MyLoad = Load(motion, dt=dt)
    ZCAmp, ZCCycles = MyLoad.calcZeroCrossing()
    PPAmp, PPCycles = MyLoad.calcPeak2Peak()

    RFAmp, RFCyc = MyLoad.calcRainflow()

    MyLoad.plotter()
    pdb.set_trace()
```