SOLUTION OF CONSTRAINED CLUSTERING PROBLEMS THROUGH HOMOTOPY TRACKING

by

David R. Easterling

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

Computer Science and Application

Layne T. Watson, Chair
Yang Cao
Naren Ramakrishnan
Kirk W. Cameron
Jeff Borggaard
William Thacker

September 24, 2014
Blacksburg, Virginia

Keywords: Multiobjective optimization, stochastic optimization, deterministic optimization, biomechanics, quadratic optimization, DIRECT, QNSTOP, KNITRO, SPAN, simulated annealing, homotopy, constrained clustering

SOLUTION OF CONSTRAINED CLUSTERING PROBLEMS THROUGH HOMOTOPY TRACKING

by

David R. Easterling

(ABSTRACT)

Modern machine learning methods are dependent on active optimization research to improve the set of methods available for the efficient and effective extraction of information from large datasets. This, in turn, requires an intense and rigorous study of optimization methods and their possible applications to crucial machine learning applications in order to advance the potential benefits of the field. This thesis provides a study of several modern optimization techniques and supplies a mathematical inquiry into the effectiveness of homotopy methods to attack a fundamental machine learning problem, effective clustering under constraints.

The first part of this thesis provides an empirical survey of several popular optimization algorithms, along with one approach that is cutting-edge. These algorithms are tested against deeply challenging real-world problems with vast numbers of local minima, and compares and contrasts the benefits of each when confronted with problems of different natures.

The second part of this thesis proposes a new homotopy map for use with constrained clustering problems. This thesis explores the connections between the map and the problem, providing several theorems to justify the use of the map and making use of modern homotopy tracking software to compare an optimization that employs the map with several modern approaches to solving the same problem.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Watson, for his patience, encouragement, advice, support, and awe-inspiring rigor in thought and research. His willingness to sacrifice his own time and energy are an unparalleled example of true devotion both to his profession and his students. I would also like to thank the members of my advisory committee, Drs. Borggaard, Cameron, Cao, Ramakrishnan, and Thacker for giving up their time, energy, and advice so that I may advance as a student and as a researcher.

I would like to thank my family for their unwavering support, especially financial support during lean times. Love is good. Cash doesn't hurt either.

Finally, I would also like to thank my labmates for providing a relaxed and engaging research experience. Shubhangi, Nick, and Rhonda, you all reminded me that it's important to stay human, even (or perhaps especially) when you're up to your eyeballs in machinery. I hope you've all made a great life for yourselves since I've known you.

This material is based on research sponsored by Department of Energy grant DE-FG02-06ER25720 and by Air Force Office of Scientific Research grant FA9550-09-1-0153.

TABLE OF CONTENTS

1. Introduction 1

2. Parallel and Stochastic Minimization 3
   - 2.1 Introduction 3
   - 2.2 Biomechanics 5
   - 2.3 Nonconvex Quadratic Minimization 8
   - 2.4 Wave Annihilation Problem 10
   - 2.5 DIRECT 11
   - 2.6 Simulated Annealing 14
   - 2.7 SPSA 15
   - 2.8 KNITRO 17
   - 2.9 QNSTOP 20
   - 2.10 Experimental Results and Discussion 24

3. Probability-One Homotopy Methods for Constrained Clustering 29
   - 3.1 Introduction 29
   - 3.2 Mathematical Background 31
     - 3.2.1 Penalty function and constraints 32
     - 3.2.2 Kreisselmeier-Steinhauser function 33
     - 3.2.3 Positively oriented nonlinear complementarity functions 34
     - 3.2.4 Homotopy theory 35
   - 3.3 Clustering Application 38
     - 3.3.1 First homotopy map 39
     - 3.3.2 K-Means approximation 42
     - 3.3.3 Second homotopy map 44
     - 3.3.4 Convergence proof 45
   - 3.4 Experimental Results 47
   - 3.5 ϵ- and δ-constraints 50
   - 3.6 Discussion 52

4. Conclusion and Future Work 57

References 58

5. Appendix A. Biomechanics problem details 62
6. Appendix B. Fortran code for Homotopy map $\tilde{\rho}_a$ 67

LIST OF TABLES

Table 1. Problem summary. 24
Table 2. Average function evaluations per experiment for each problem. 25
Table 3. Results for the biomechanics problem. 25
Table 4. Results for the nonconvex quadratic minimization problem dual. 26
Table 5. Results for the wave annihilation problem. 26
Table 6. Dataset summary. 49
Table 7. Adjusted Rand index, "easy" constraints. 51
Table 8. Adjusted Rand index, "hard" constraints. 52
Table 9. Davies-Bouldin index, ϵ- and δ- constraints 53

LIST OF FIGURES

Figure 1. Schematic drawing of the three segment sagittal planel model representing the human body 7
Figure 2. $f(x)$ along the line through two points of the biomechanics problem (top), and a zoomed view (bottom). 9
Figure 3. An example of a box scatter plot. 12
Figure 4. A typical QNSTOP progression. 23
Figure 5. The inverse image $\rho_a^{-1}(0)$ for ρ_a transversal to zero. 37
Figure 6. The iris dataset with "easy" constraints. 53
Figure 7. The iris dataset with "hard" constraints. 54
Figure 8. The liver dataset with "easy" constraints. 54
Figure 9. The liver dataset with "hard" constraints. 54
Figure A1. Schematic drawing of the three segment sagittal plane model representing the human body. 62

Chapter 1.

INTRODUCTION

As long as people seek to find correlations in data, machine learning will persist; as long as machine learning persists, advances in optimization will be needed to guide the way. All machine learning is fundamentally optimization. The trick is in knowing how to structure the problem to be optimized, and knowing which optimization tool to use for the job.

There is a fundamental disconnect that is often associated with optimizing real-world problems. If a physical problem does not lend itself to an easily-optimized model, the temptation is to either settle for extremely local optimization or select a model that may allow for ease of optimization at the cost of extensive modelling error. Rarely is it considered feasible for good minima to be discovered in models based on hard real-world problems, especially those containing stochastic noise or many local minima unless some aspect of the problem can be exploited to reduce these factors.

An easy example to illustrate this is the problem of clustering under constraints. This problem contains a host of pitfalls for the unwary optimizer. First, clustering, being a partitioning problem of (usually) discrete points, would appear to lend itself to an integer programming approach. However, such an approach is doomed to fail for constrained clustering problems, which are NP-hard. At best, heuristics can be used to attempt to find the best local minimum point attainable with limited resources; at worst, the unwary optimizer may as well be throwing darts.

Of course, attempting to solve this problem without simplifying it in some way is laughable. On the other hand, finding a good model that provides potential answers without giving up too much in the way of modeling error is a real challenge. It helps that, perhaps more than in some disciplines, the machine learning researcher is not always looking for the best answer; information can still be gleaned from clusterings that are merely adequate. Still, how many constraints can the researcher relax before losing too much of the vital information they contain, while still achieving a valid clustering? The existence of a constraint set does not excuse the researcher from attempting to satisfy the clustering hypothesis, after all. Finding some way to allow the researcher to intelligently trade off

between satisfaction of the constraint set and satisfaction of the clustering hypothesis is the driving motivation behind this research. The aim of this thesis is to provide a set of useful optimization tools to the machine learning researcher and to provide a solid mathematical background to justify their use, both empirically and theoretically.

This thesis is organized as follows. Chapter 2 presents a selection of powerful modern optimization tools and shows their general applicability to a selection of three modern problems that challenge cutting-edge optimizers. The various advantages and disadvantages of each are outlined in the context of these real-world problems. Chapter 3 takes a closer look at one particularly powerful optimization method, the construction and tracking of probability-one homotopy maps, and applies it to a fundamental problem in machine learning, clustering under constraints. Chapter 4 concludes. Supplementary material can be found in the Appendix or requested from the author.

Chapter 2.

PARALLEL AND STOCHASTIC MINIMIZATION

2.1 Introduction

The minimization of difficult functions is an important topic in numerical analysis. An optimization method that solves one problem efficiently and effectively can fail to yield useful results for another problem, even if the two problems under consideration share broad similarities. Three problems are considered here: a biomechanical control problem with an unknown global minimum, a nonconvex nonsmooth quadratic minimization problem with a global minimum known by construction, and a smooth problem in reflection reduction with a global minimum that can be directly calculated. While the problems are of similar dimensionality and all have a large number of local minima, the character of each problem is distinct.

The biomechanical control problem, while deterministic, contains enough modeling noise to cause deterministic optimization algorithms difficulty. Essentially, the objective function is ill-conditioned with respect to motion regime changes (e.g., the switch from extension to flexion models). The biomechanical control problem is a constrained optimization problem of 57 dimensions. The nonconvex nonsmooth quadratic minimization problem is a reformulation of an integer programming problem, and the function considered here is carefully constructed to contain a large number of local minima and a single global optimum point. This minimization problem is an unconstrained problem with a scalable number of dimensions, chosen here to be 57. The wave annihilation problem is a smooth minimization problem with an even number of variables, chosen as 56 to keep the size comparable to that of the other two problems. Together, these three problems provide a useful context in which to compare the performance of the optimization algorithms on moderately large and qualitatively different problems.

Five optimization algorithms are considered for each problem: the simultaneous perturbation stochastic approximation algorithm, two parallel implementations of a simulated

annealing scheme, a parallel implementation of the DIRECT algorithm, the direct interior point method found in the commercial KNITRO optimization package, and a new quasi-Newton stochastic algorithm, QNSTOP.

The simultaneous perturbation stochastic approximation algorithm (SPSA) is an unconstrained stochastic optimization algorithm notable for the small number of objective function evaluations per iteration (Spall, 1998a). This allows it to scale to higher dimensions better than the finite difference methods from which it is derived (Spall, 1987). Like the finite difference methods, it suffers from a tendency to become trapped at local minima. SPSA is usually employed as a local optimization algorithm, but it may function as a global optimization algorithm under certain broad conditions (Maryak & Chin, 2008). The parallel implementation employed here is a naive one, with minimal interprocessor communication. This parallel SPSA is applied to these three problems to test its suitability for problems with a high dimensionality and a large number of local minima. For more information on the SPSA algorithm, see (Spall, 1998b) and (Spall, 1992).

Simulated annealing is an unconstrained stochastic global optimization algorithm commonly used for difficult biomechanics problems. The parallel implementation employed here, simulated parallel annealing within a neighborhood (SPAN), is designed to minimize interprocessor communication while maximizing the use of multiple processors to compute objective function values at the desired points (Higginson et al., 2004). For more on simulated annealing, see (Goffe et al., 1994), (Ingber, 1993), and (Kirkpatrick et al., 1983). (Ram et al., 1996) discusses other parallel simulated annealing algorithms.

The DIRECT algorithm is a highly parallelizable box-constrained deterministic global optimization algorithm (Jones et al., 1993). The parallel implementation employed here, pVTdirect, is designed to preserve the deterministic nature of the algorithm while exploiting its natural parallelism (He et al., 2009b), (He et al., 2000). For more information on DIRECT, see (Jones et al., 2001), (He et al., 2009a), and (He et al., 2002).

KNITRO contains a collection of algorithms for local nonlinear optimization developed by Ziena Optimization, LLC. While all the optimization algorithms in the KNITRO package are designed for twice continuously differentiable problems, KNITRO nevertheless contains code for approximation of derivatives and can be used on nonsmooth problems as well, though for such problems the performance may degrade. As the only truly gradient-driven optimization technique considered here, KNITRO provides a contrast to the other algorithms

employed here to show the usefulness of such a technique with the test problems in this study. For more information on KNITRO, see (Byrd et al., 2006), (Byrd et al., 1999), and (Byrd et al., 2000).

QNSTOP, an algorithm under development at Indiana University (Castle, 2012), is a local search strategy for stochastic optimization that synthesizes ideas from numerical optimization (secant updates, trust regions) and response surface methodology (ridge analysis). Here, stochastic optimization describes problems in which function evaluation is uncertain, i.e., instead of computing $y = \mu(x)$, y is drawn from a probability distribution $P(x)$. For example, the case of additive normal errors with equal variance σ^2 is $y \sim \text{Normal}(\mu(x), \sigma^2)$. Some modifications are therefore necessary to apply this algorithm to the deterministic problems being considered.

This chapter is organized as follows. Sections 2.2, 2.3, and 2.4 describe the three test problems. The parallel DIRECT is described in Section 2.5, simulated annealing in Section 2.6, Spall's SPSA algorithm in Section 2.7, KNITRO in Section 2.8, and QNSTOP in Section 2.9. Section 2.10 contains a discussion of experimental results and concludes.

2.2 Biomechanics

The first problem under consideration is a biomechanical control problem. Optimization is commonly used within biomechanics to determine the control inputs to a model of the human musculoskeletal system that cause the model to move in a desired manner. Optimization is necessary because the complicated dynamics make direct solution for these inputs difficult. The objective function in these optimization problems is commonly a function of model movement patterns, and therefore optimizing these objective functions effectively solves for the control inputs that elicit a desired movement of the model.

The model of the human musculoskeletal system employed here can be visualized as a side view of three body segments (Figure 1) including the shanks (left and right lower legs combined), thighs (left and right combined), and the combined head, arms, and trunk. The foot segments were neglected because the feet are implicitly assumed to stay flat on the supporting surface. As such, the joint representing the ankles simply connects the distal end of the shanks to the supporting surface. Segment length, mass, center of mass location, and mass moments of inertia were determined from subject height (1.6 m), mass (60 kg), and the formula presented in (Pavol et al., 2002). Segments were connected by frictionless

pin joints. The model was actuated by torque generators at the joints representing the ankles, knees, and hips. The "joint torques" produced by these torque generators represent the resultant torque applied to the musculoskeletal model from all skeletal muscles crossing each joint. The inputs to these torque generators were piecewise linear functions formed by joint activation "nodes" spaced 100 ms apart for the entire 1.8 seconds of the model simulation, and whose values vary between -1 and 1. Within a single joint activation profile, the activation level between two consecutive nodes may differ by no more than 1.25. (These constraints are enforced via a bijective conformal mapping of a cube (Easterling et al., 2010).) The input to the torque generators between consecutive nodes was determined by linear interpolations between adjacent nodes. Each joint had 19 activation nodes, and all three joints together had a total of 57 activation nodes. The desired movement of the model was to maintain upright balance without stepping after a short, abrupt backward displacement of the supporting surface. The optimization solved for the 57 activation nodes that minimized horizontal movement of the model and penalized unrealistic movements.

Joint torques produced by the torque generators were the sum of passive and active joint torques $T = T_p + T_a$. Conceptually, passive joint torques result from passive tissues such as elastic ligaments and anatomical constraints that limit a joint's range of motion, and active joint torques result from activation of skeletal muscles. Active joint torques were a function of the input to the torque generator, joint angle (Hoy et al., 1990), joint angular velocity (Selbie & Caldwell, 1996), and activation dynamics (Easterling et al., 2010) that limit how quickly these inputs change between adjacent nodes.

The four optimization algorithms under consideration are used to attempt to determine the values for the joint activation nodes that minimize the objective function

$$\begin{aligned} f = {} & w_1 \int_{t_0}^{t_f} (X_C(t) - X_A)dt + w_2 \int_{t_0}^{t_f} e(\theta(t))dt \\ & + w_3 \int_{t_0}^{t_f} e(\dot{\theta}(t))dt + w_4 \int_{t_0}^{t_f} \left(\sum_{i=1}^{3} \dot{q}_i(t)^2 \right)^{1/2} dt \\ & + w_5 \int_{t_0}^{t_f} \dot{X}_C(t)dt + w_6 \int_{t_0}^{t_f} \ddot{X}_C(t)dt \\ & + w_7 \sum_{i=1}^{3} \int_{t_0}^{t_f} (\tau_i(t)^2)dt \end{aligned}$$

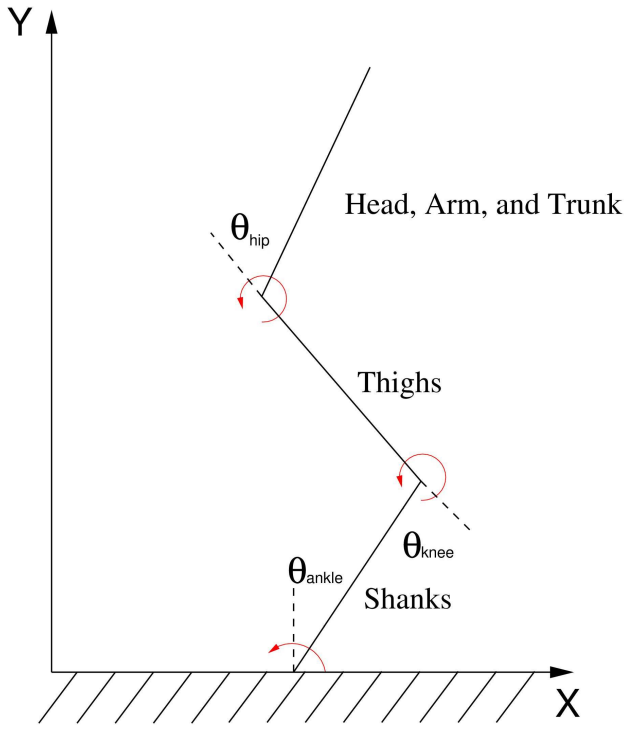

Figure 1. Schematic drawing of the three segment sagittal plane model representing the human body.

adapted from (Yang et al., 2007), (Yang et al., 2008), where $e(s(t)) = \sum_{i=1}^{3} \phi(s_i(t))$ and

$$\phi(s_i(t)) = \begin{cases} s_i(t)^- - s_i(t), & s_i(t) < s_i(t)^-, \\ 0, & s_i(t)^- \leq s_i(t) \leq s_i(t)^+, \\ s_i(t) - s_i(t)^+, & s_i(t) > s_i(t)^+, \end{cases}$$

with $s_i(t)^-$ and $s_i(t)^+$ representing the lower and upper physical bounds of the joint angles, respectively.

The first term in the objective function minimizes the maximum horizontal displacement of the center of mass $X_C(t) - X_A$, where $X_C(t)$ is the center of mass of the body on the displacement platform (m) and X_A is the position of the ankle on the displacement platform (m). These values are taken from experimental data. The second and third terms restrict joint angle $\theta(t)$ (rad) and angular velocity $\dot{\theta}(t)$ (rad/s) to remain within previously published physiologic limits. The fourth, fifth, and sixth terms minimize segment angular velocity $\dot{q}_i(t)$ (rad/s), center of mass velocity $\dot{X}_C(t)$ in (m/s), and center of mass acceleration $\ddot{X}_C(t)$ in (m/s^2), respectively, over the entire simulation. The seventh term minimizes the integral of the square of the joint torques $\tau_i(t)$, the sum of active and passive torques. The weights for f are $w_1 = 1000$, $w_2 = 500$, $w_3 = 500$, $w_4 = 50$, $w_5 = 100$, $w_6 = 25$, and $w_7 = 0.025$.

The initial joint angle configuration is derived from experimental data, and the initial joint angular velocities are set to zero.

In summary, each optimization algorithm attempts to minimize the stated $f(x)$, subject to the constraints that $-1 \leq x \leq 1$ and that within a single activation profile, the activation level between two consecutive discrete nodes may differ by no more than 1.25.

The best known solution to this problem is $f(x^*) = 1222.05$. This solution was found in the course of this investigation by a parallel run of pVTdirect, centered at a point found by the QNSTOP algorithm with a single experiment centered on the origin that was granted a function evaluation budget of 10^5. This measure $f(x)$ is unitless and only relevant when compared to other function evaluations to determine the relative fitness of a minimizing point.

The highly variable nature of the biomechanics objective function $f(x)$ is shown in Figure 2, which shows the function evaluated along the line between two widely separated points and a zoomed view of a "smooth" part of the plot, demonstrating the presence of local noise.

2.3 Nonconvex Quadratic Minimization

The second problem of interest is the nonconvex box-constrained quadratic minimization problem:

$$(\mathcal{P}_b) : \min \left\{ P(x) = \frac{1}{2} x^{\mathrm{T}} A x - f^{\mathrm{T}} x : x \in X_b \right\},$$

where

$$X_b = \{ x \in \mathbb{R}^n | -1 \leq x_i \leq 1, \forall i = 1, \ldots, n \}.$$

Replacing the feasible set X_b by its vertices

$$\delta X_b = \{ x \in \mathbb{R}^n | x \in \{-1, 1\}^n \}$$

gives the integer programming problem

$$(\mathcal{P}_{\mathrm{ip}}) : \min \left\{ P(x) = \frac{1}{2} x^{\mathrm{T}} A x - x^{\mathrm{T}} f : x \in \delta X_b \right\}.$$

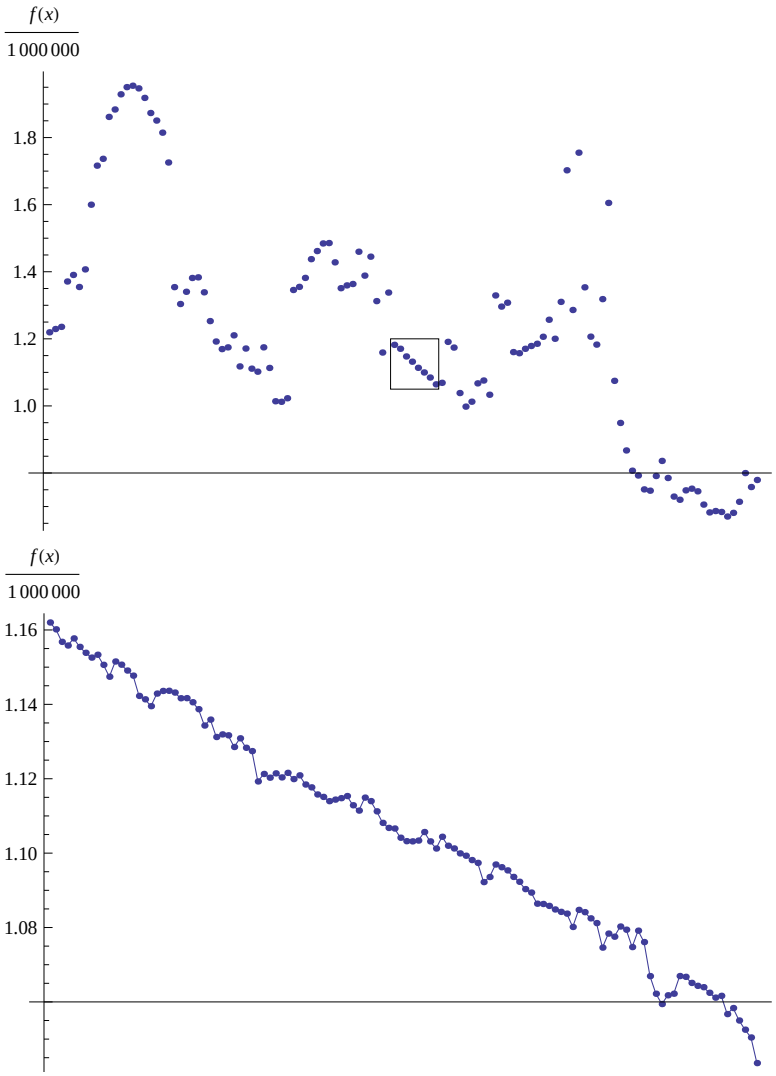

Figure 2. $f(x)$ along the line through two points of the biomechanics problem (top), and a zoomed view (bottom).

Using the canonical duality theory of Gao (Gao, 2000a), (Gao, 2000b), the integer programming problem $(\mathcal{P}_{\rm ip})$ may be reformulated as

$$(\mathcal{P}^d_{\rm ip}) : \min \left\{ Q(\sigma) = \frac{1}{2}\sigma^{\rm T}\sigma - \sum_{i=1}^{n} \left| f_i + (B^{\rm T}\sigma)_i \right| : \sigma \in \mathbb{R}^m \right\},$$

where $\sigma = (\sigma_1, \ldots, \sigma_m)$, $f = (f_1, \ldots, f_n)$, and the $m \times n$ real matrix B is related to A. The reformulation is a nonconvex nonsmooth unconstrained minimization problem. Here $m = 57$, $n = 190$,

$$\hat{B} = \begin{bmatrix} 1 & -1 & 0 & -1 & 2 & 0 & 1 & -2 & 1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 0 & -2 & 2 & 0 & 1 \\ 2 & 2 & -1 & -1 & 2 & -2 & 0 & 0 & -1 & 1 \end{bmatrix},$$

$$\begin{aligned} \hat{f} = 10^{-2}[&1.491803633709836,\ 3.0717213019723066, \\ &5.246230264266409,\ -6.718373452055033, \\ &3.969549763760797,\ 7.502845410079123, \\ &5.622108089244097,\ -1.9585631018739558, \\ &-2.729844702016424,\ 8.26721052052138], \end{aligned}$$

$$B = I_{19\times 19} \otimes \hat{B},\ \text{and}\ f = e_{19} \otimes \hat{f},$$

where $e_{19} = (1, \ldots, 1) \in \mathbb{R}^{19}$. This problem has exactly 2^{19} known local minimum points and the unique global minimum $Q(\sigma^{(1)})$ is located at

$$\sigma^{(1)} = e_{19} \otimes (\, 6 \quad -4 \quad 12\,).$$

All the local minima are within 0.5% of the global minimum $Q(\sigma^{(1)}) = -1866.01$.

2.4 Wave Annihilation Problem

The wave annihilation problem studied here was first presented in (Hager et al., 2000). That study developed a method for producing a coating of total thickness T distributed evenly in n layers of varying properties between two media to eliminate the reflection of waves over a band of frequencies $[\Omega_0, \Omega_1]$ in one of those media. Reflections are eliminated entirely at n specific frequencies and reduced significantly for other frequencies within this band; as n approaches infinity, reflections within this band are eliminated entirely. This process is treated as an acoustic application in (Hager et al., 2000), but as is pointed out, it can easily be adapted to electromagnetism or any other phenomena governed by variants of the linear wave equation.

Given n, a crucial component to this process is to determine the n specific coatings such that the reflection $r = 0$ at frequency

$$\omega_k = \Omega_0 + \left(\frac{k-1}{n-1}\right)(\Omega_1 - \Omega_0)$$

for $k = 1, 2, \ldots, n$, where the complex-valued

$$r(n, \gamma, \kappa, \omega_k) = \frac{(\Gamma_-,\ \gamma_1) \prod_{j=1}^{n} A_j \begin{pmatrix} -1 \\ 1 \end{pmatrix}}{(\Gamma_-,\ \gamma_1) \prod_{j=1}^{n} A_j \begin{pmatrix} 1 \\ 1 \end{pmatrix}},$$

$$A_j = \begin{pmatrix} \gamma_j e_j^+ & \gamma_{j+1} e_j^- \\ \gamma_j e_j^- & \gamma_{j+1} e_j^+ \end{pmatrix}, \qquad \gamma_{n+1} = \Gamma_+, \qquad e_j^{\pm} = \exp\left(\frac{2\gamma_j \Delta x \omega_k i}{\kappa_j}\right) \pm 1,$$

$i = \sqrt{-1}$, Γ_+ and Γ_- are the impedances of the half-spaces surrounding the coating, $\Delta x = T/n$, and γ_j and κ_j are the impedance and stiffness of layer j, respectively. Note that unlike the other problems studied here, r is differentiable and the nonlinear system of equations can be solved using a variation of Newton's method (Hager et al., 2000). An objective function $f = \|r\|^2$ may be formed by observing that the inner product of the complex vector $\big(r(\omega_1), \ldots, r(\omega_n)\big)$ with itself yields a scalar real value with a known minimum of zero where the original vector is zero. By choosing $n = 28$, a problem of 56 real variables is constructed that can be studied using the algorithms presented here, with both the impedance and the stiffness of the n layers being used as arguments to r, while the frequencies ω_k are determined directly from n. For our purposes, Γ_+ is chosen to be 1 and Γ_- is chosen to be 28.14776 (the ratio between the two is the same as the ratio between the reflectivity of water and steel), $T = 1$ m, and $\Omega_0 = 0.09091$ Hz while $\Omega_1 = 10\Omega_0$.

2.5 DIRECT

The DIRECT (DIviding RECTangles) algorithm by D.R. Jones (Jones et al., 1993) is a deterministic global optimization algorithm. The DIRECT algorithm does not require the computation of the gradient of the objective function, or even objective function values (ranking information is sufficient). It performs Lipschitzian optimization, but does not require knowledge of the Lipschitz constant.

The DIRECT algorithm works as follows (He et al., 2009b). The algorithm begins with an initial box normalized to the unit hypercube. The objective function F (assumed to satisfy a Lipschitz condition) is evaluated at the center of this box. The current minimum value is initialized to this value. An evaluation counter m and an iteration counter t are

initialized to $m = 1$ and $t = 0$. The following process is repeated until m or t reaches some prespecified limit.

Selection. Identify the set S of "potentially optimal" boxes. Here "potentially optimal" means that (1) for some Lipschitz constant K, the box potentially contains a point with smaller objective function value than any other box, and (2) $F(c) - K \cdot L/2 \leq f_{min} - \epsilon|f_{min}|$, where F is the objective function, c is the center point of the box, K is the same Lipschitz constant, L is the box diameter, f_{min} is the current minimum value for the objective function, and ϵ is a small, nonnegative, fixed constant.

Sampling. Select one of the potentially optimal boxes B from S. For box B, identify the set I of dimensions with maximum side length L and let $\delta = \frac{1}{3}L$. Sample the function at all points of the form $c \pm \delta e_i$ for each $i \in I$, where c is the center of the rectangle and e_i is the ith standard basis vector. Update m.

Division. Divide the box containing the point c into thirds along the dimensions in I, beginning with the dimension with the least value of $\min\{F(c + \delta e_i), F(c - \delta e_i)\}$, and ending with the dimension with the greatest such value. Update the minimum value.

Iteration. Remove the box B from the set of potentially optimal boxes S. If $S = \emptyset$, then increment t and go to **Selection**. Otherwise, go to **Sampling**.

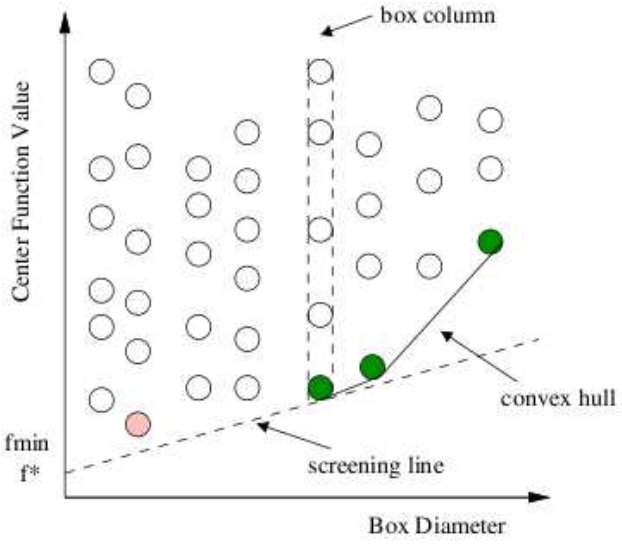

Figure 3. An example of a box scatter plot.

The method of choosing the sub-box according to both objective function value and box size gives DIRECT its local and global aspects. The DIRECT algorithm performs a convex hull computation to determine potentially optimal boxes without using the Lipschitz constant directly (see Figure 3 for an illustration). From Figure 3, it is clear that the convex hull consists of boxes with objective function values that are minimal amongst all boxes of the same size (notice that the set of boxes of the same size form a "box column"). Since every box is ultimately examined, DIRECT will not get stuck at a local optimum, but will instead perform a global search of the feasible set. Further details can be found in (Jones et al., 1993).

A parallel implementation of the DIRECT algorithm, pVTdirect, developed at Virginia Tech (He et al., 2009b), is used here. The pVTdirect implementation contains some modifications to the original DIRECT algorithm in order to meet the needs of various applications and improve the performance on large scale parallel systems. First, an optional domain decomposition step is added to create multiple subdomains, each with a starting point for a DIRECT search. Empirical results have shown that this approach significantly improves load balancing among a large number of processors, and likely shortens the optimization process for problems with asymmetric or irregular structures. The **Selection** step has two new features. The first is an "aggressive" switch that originally appeared in Watson et al. (Watson & Baker, 2001). The switch generates more function evaluation tasks which may help balance the workload under the parallel environment. The second is an adjustable ϵ, which is the minimum improvement required to update the current minimum objective function value. In general, smaller ϵ values make the search more local and generate more function evaluation tasks. On the other hand, larger ϵ values bias the search toward broader exploration, exhibiting slower convergence. The value of ϵ is taken as zero by default, but can be specified by the user depending on problem characteristics and optimization goals.

In the serial version of DIRECT, **Sampling** samples one box at a time to conserve memory. In pVTdirect, more tasks are performed in parallel, so new points are sampled around all boxes in S along their longest dimensions during **Sampling**. This obviates the need for the step **Iteration** and simplifies the loop. Since box center function values may be identical or nearly so, the parallel **Sampling** may yield a different box sequence in each box column (i.e., ordered column of equal-sized boxes) as the parallel scheme varies. Thus boxes will be subdivided in a different order, causing pVTdirect to become nondeterministic. To

maintain its deterministic property, pVTdirect performs lexicographical order comparison between box center coordinates, thereby keeping the boxes in the same sampling sequence on the same platform. However, the computed values can depend on the particular machine or compiler one uses, so the results for the same problem may vary slightly from system to system.

Finally, pVTdirect allows more stopping conditions for the termination of the algorithm. First, the minimum diameter variable `MIN_DIA` causes an exit when the diameter of the best box reaches the value `MIN_DIA`. Second, the objective function convergence tolerance variable `OBJ_CONV` causes an exit when the relative change in the optimum objective function value has reached the given value. These new stopping conditions address a complaint by Jones et al. (Jones et al., 1993) that the original stopping criterion for DIRECT was somewhat artificial and unconvincing for many real-world optimization problems.

2.6 Simulated Annealing

Simulated annealing is a stochastic algorithm, generally well suited to hard problems in biomechanics (Higginson et al., 2004). It begins with an initial feasible guess X_0 and the current minimum is set to $F(X_0)$. The algorithm then pseudorandomly generates points in a neighborhood of X_0 until it generates a feasible point X_1. If $F(X_1) \leq F(X_0)$, then the current point is set to X_1. If $F(X_1) - F(X_0) = d > 0$, then the current point is set to X_1 with probability $e^{-d/T}$, where T is the *temperature*. With probability $1 - e^{-d/T}$, X_1 is rejected and the algorithm continues to generate points around X_0 until a new feasible point is generated. This process is repeated until the distance between successive iterates is less than some small, fixed value.

The temperature begins at some high value T and is continually lowered throughout the search according to some temperature schedule. Since $e^{-d/T} \approx 1$ for $T \gg d$, a relatively large number of random moves will be made at the beginning of the search, when T is large. Thus the beginning of the search is primarily global in nature. As T decreases throughout the search, $e^{-d/T}$ gets closer to zero, and therefore the search becomes increasingly greedy, eventually performing similarly to a gradient descent method.

Simulated Parallel Annealing within a Neighborhood (SPAN) (Higginson et al., 2004), one of the implementations used here, was developed for parallel computation. A straightforward serial simulated annealing algorithm consists of three nested loops: a *temperature*

reduction loop that causes the temperature to gradually decline as the algorithm proceeds, an inner *search radius* loop that causes the neighborhood to gradually shrink while maintaining the same temperature, and an innermost *control* loop that handles the perturbation of the variables to find the minimum point. The SPAN implementation parallelizes the independent search radius loop, so that all processors being utilized have all the information required to do a function evaluation (the current X and the radius of the search). The processors then communicate their results in a global communication (gather) before the search radius is adjusted so that roughly half of all the generated points in the previous neighborhood are acceptable in the new neighborhood. This communication overhead scales linearly with the number of processors involved, causing a notable degradation of performance on a large number of processors, especially with fast function evaluations (Higginson et al., 2004). For comparison's sake, a more traditional naive parallel implementation of simulated annealing (with random X_0) was also constructed, using the same underlying algorithm (Goffe et al., 1994) with only one gather operation at the end of the optimization to collect all results. Both of the implementations used an initial temperature of $T = 5000$ and a cooling schedule of $T_{\text{next}} = 0.85(T_{\text{current}})$ for all three problems.

2.7 SPSA

Spall's simultaneous perturbation for stochastic approximation (SPSA) algorithm is, like simulated annealing, a stochastic global optimization algorithm (Spall, 1987). SPSA is similar to the standard finite difference stochastic approximation (FDSA) algorithm (Kiefer & Wolfowitz, 1952) with one primary difference. The general form of both SPSA and the FDSA algorithm is $X_{k+1} = X_k - a_k \cdot g(X_k)$, where X_k is the variable vector, a_k is the kth element of the gain sequence a, and $g(X_k)$ is meant to approximate the gradient of the objective function at X_k. Whereas the FDSA algorithm perturbs the components of the vector X in the objective function $F(X)$ one at a time, SPSA performs a simultaneous perturbation of each component of X. This might appear to reduce the ability of the algorithm to search the problem space effectively when compared to component-by-component perturbation, but Spall (Spall, 1998a) claims that "one properly chosen simultaneous random change in all the variables in a problem provides as much information for optimization as a full set of one-at-a-time changes of each variable".

As stated above, the main difference between SPSA and the FDSA algorithm is the method of perturbation of the components of X. The FDSA algorithm explores in each dimension around the point X_k, seeking the steepest descent (negative gradient) direction. Formally, the ith component of the (two-sided) gradient approximation for FDSA is computed as

$$g_i(X_k) = \frac{\hat{F}(X_k + c_k e_i) - \hat{F}(X_k - c_k e_i)}{2c_k},$$

where $\hat{F}(X) = F(X) + \text{noise}$, c_k is the kth element of a sequence c that converges monotonically to zero slowly as $k \to \infty$, and e_i is the ith standard basis vector. For an n-dimensional X, $2n$ objective function evaluations per iteration are required.

SPSA generates a vector v using a Monte Carlo method (Spall, 1998b), evaluates the objective function at two points $X_k + v$ and $X_k - v$ at each iteration to approximate the gradient at X_k, and then adjusts X_k based on the resulting estimation of the gradient. Consequently, SPSA uses two objective function evaluations at each iteration. Formally,

$$g_i(X_k) = \frac{\hat{F}(X_k + c_k \Delta_k) - \hat{F}(X_k - c_k \Delta_k)}{2c_k \Delta_{ki}},$$

where $\Delta_k = (\Delta_{k1}, \Delta_{k2}, \ldots, \Delta_{kn})$ is the user-specified random perturbation vector and $v = c_k \Delta_k$. The distribution of Δ_k must satisfy certain conditions in order to guarantee convergence; in particular, each component of Δ_k must be nonzero (Spall, 1998a).

In the implementation of SPSA used for the biomechanics problem, an adaptation called *blocking* is employed that requires an extra function evaluation at each iteration. The idea is to "block" updates to X_k if the update will produce a new objective function value that is significantly worse than the current objective function value. This requires that the objective function be evaluated at X_k, as well as at $X_k + v$ and $X_k - v$. The extra function evaluation at each iteration increases the total number of evaluations by 33%, but can result in faster convergence of the algorithm. However, this technique can also reduce the likelihood that the algorithm will achieve a global minimum (Spall, 1998b). Projection is employed to prevent the algorithm from moving outside the feasible set.

When attempting to solve the nonconvex quadratic minimization problem dual and the wave annihilation problem, blocking is not used and instead an adaptation called *injected noise* is employed that simulates a random element in the objective function, with the intent of inducing the algorithm to abandon local minima. While the resulting implementation

may take longer to converge to a minimum, the likelihood of global convergence is increased (Maryak & Chin, 2008). This adaptation is not necessary in the biomechanics problem because the noisiness inherent in the objective function fills the same role.

2.8 KNITRO

The KNITRO optimization package contains three optimization algorithms, but only one of them is utilized here, the direct interior point method (Byrd et al., 1999). Since the problems here are unconstrained except for upper and lower bound constraints, the sequential quadratic programming (SQP) method used by the KNITRO solver should be very efficient. However, the interior point method, like all the methods in the KNITRO package, assumes that the objective function is twice continuously differentiable. This is not the case for either the biomechanics problem or the nonconvex quadratic minimization problem dual, so a central difference method, included in the package, is invoked to supply second derivative input values; as a result, the efficacy of the direct interior point method suffers.

It is important to note that the direct interior point method employed by KNITRO, while very efficient at solving general constrained nonlinear optimization problems, loses some efficiency compared with other optimization algorithms for problems with only bound constraints (Byrd et al., 1999). Nevertheless, as a widely used commercial gradient-dependent optimization package, KNITRO represents a class of local optimization algorithms that apply to these difficult nonlinear problems.

The direct interior point algorithm seeks to find Karush-Kuhn-Tucker (KKT) points using sequential quadratic programming and trust region methods. As with all nonlinear optimization algorithms, the goal is to solve problems of the form

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{subject to} \quad & c_E(x) = 0, \\ & c_I(x) \geq 0, \end{aligned}$$

where here $f : \mathbb{R}^n \to \mathbb{R}$. The interior point direct algorithm first defines the barrier subproblem

$$\begin{aligned} \min_{x,s} \quad & f(x) - \mu \sum_{i=1}^{m} \ln s_i \\ \text{subject to} \quad & c_E(x) = 0, \\ & c_I(x) - s = 0, \end{aligned}$$

where $c_E : \mathbb{R}^n \to \mathbb{R}^l$, $c_I : \mathbb{R}^n \to \mathbb{R}^m$, the barrier parameter $\mu > 0$ and the vector of slack variables $s \in \mathbb{R}^m$ is positive.

Following (Byrd et al., 2006), the KKT conditions for the above system can be written as

$$\begin{aligned} \nabla f(x) - {A_E}^{\mathrm{T}}(x)y - {A_I}^{\mathrm{T}}(x)z &= 0 \\ -\mu e + Sz &= 0 \\ c_E(x) &= 0 \\ c_I(x) - s &= 0, \end{aligned}$$

where $e = (1, \ldots, 1)^{\mathrm{T}}$, $S = \mathrm{diag}(s_1, \ldots, s_m)$, A_E and A_I are the Jacobian matrices corresponding to the equality and inequality constraint vectors respectively, and y and z represent vectors of Lagrange multipliers. The line search algorithm, used here, applies Newton's method to the above system, backtracking if necessary to ensure that $s, z > 0$, and ensuring that the merit function $\psi(x, s) = f(x) - \mu \sum_{i=1}^{m} \ln s_i$ is sufficiently reduced.

Applying Newton's method in the variables x, s, y, z gives

$$\begin{bmatrix} \nabla^2_{xx} L & 0 & -{A_E}^{\mathrm{T}}(x) & -{A_I}^{\mathrm{T}}(x) \\ 0 & Z & 0 & S \\ A_E(x) & 0 & 0 & 0 \\ A_I(x) & -I & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ d_y \\ d_z \end{bmatrix} = - \begin{bmatrix} \nabla f(x) - {A_E}^{\mathrm{T}}(x)y - {A_I}^{\mathrm{T}}(x)z \\ Sz - \mu e \\ c_E(x) \\ c_I(x) - s \end{bmatrix}$$

where $L(x, s, y, z) = f(x) - \mu \sum_{i=1}^{m} \ln s_i - y^{\mathrm{T}} c_E(x) - z^{\mathrm{T}}(c_I(x) - s)$ is the Lagrangian of the above problem and $Z = \mathrm{diag}(z_1, \ldots, z_m)$.

If the inertia of the above matrix is $(n+m, l+m, 0)$, then the step d determined above is a descent direction for the merit function ψ. If so, the scalars

$$\begin{aligned} \alpha_s^{\max} &= \max\{\alpha \in (0, 1] : s + \alpha d_s \geq (1 - \tau)s\}, \\ \alpha_z^{\max} &= \max\{\alpha \in (0, 1] : z + \alpha d_z \geq (1 - \tau)z\}, \end{aligned}$$

are computed with $\tau = 0.995$. If $\min\{\alpha_s^{\max}, \alpha_z^{\max}\}$ is not too small, a backtracking line search is used to compute the step lengths $\alpha_s \in (0, \alpha_s^{\max}]$, $\alpha_z \in (0, \alpha_z^{\max}]$ that provide a sufficient decrease in ψ. The next iteration, with a reduced barrier variable, is then computed with

$$x^+ = x + \alpha_s d_x, \quad s^+ = s + \alpha_s d_s,$$
$$y^+ = y + \alpha_z d_y, \quad z^+ = z + \alpha_z d_z.$$

However, if the inertia of the matrix is not of the desired form or the step lengths α_s or α_z are too small, a trust region method is employed to compute the current step d. This has the benefit of guaranteeing a successful step even in the presence of negative curvature or singularity.

The trust region method employed, which is also the standard step of the Interior/CG KNITRO algorithm, takes the following form. First, the normal step $v = (v_x, v_s)$ is computed by solving the subproblem

$$\min_v \quad \|A_E v_x + c_E\|_2^2 + \|A_I v_x - v_s + c_I - s\|_2^2 \tag{2.1}$$
$$\text{subject to} \quad \|(v_x, S^{-1} v_s)\|_2 \leq 0.8\Delta. \tag{2.2}$$

This subproblem is solved using a dogleg approach that minimizes (2.1) along a piecewise linear path composed of a steepest descent step in the norm used in (2.2) and a Newton step with respect to the same norm. The scaling $S^{-1}v_s$ in the norm tends to limit the extent to which the bounds on the slack variable are violated.

Once the normal step $v = (v_x, v_s)$ is computed, the tangential subproblem is then defined as

$$\min_{d_x, d_s} \quad \nabla f^t d_x - \mu e^t S^{-1} d_s + \frac{1}{2}\left(d_x^t \nabla_{xx}^2 L d_x + d_s^t S^{-1} Z d_s\right) \tag{2.3}$$
$$\text{subject to} \quad A_E d_x = A_E v_x \tag{2.4}$$
$$A_I d_x - d_s = A_I v_x - v_s \tag{2.5}$$
$$\|(d_x, S^{-1} d_s)\|_2 \leq \Delta. \tag{2.6}$$

To find the approximate tangential solution d, first the scaling $\tilde{d_s} \leftarrow S^{-1} d_s$ is applied to convert (2.6) into a sphere, and then a standard projected conjugate gradient method is

applied to this transformed quadratic program, iterating in the linear manifold defined by (2.4)–(2.5). Finally, the step d is truncated if necessary to preserve $s > 0$.

2.9 QNSTOP

QNSTOP is a class of quasi-Newton methods for stochastic optimization. The implementation considered features several variations specific to global, deterministic optimization. In iteration k, QNSTOP methods compute the gradient vector $\hat{g}_k$ and Hessian matrix $\hat{H}_k$ of a quadratic model

$$\widehat{m}_k(X - X_k) = \hat{f}_k + \hat{g}_k^{\mathrm{T}} (X - X_k) + \frac{1}{2} (X - X_k)^{\mathrm{T}} \hat{H}_k (X - X_k), \tag{2.7}$$

of the objective function f centered at X_k, where $\hat{f}_k$ is generally not $f(X_k)$.

In the unconstrained context, QNSTOP methods progress by

$$X_{k+1} = X_k - \left[\hat{H}_k + \mu_k W_k\right]^{-1} \hat{g}_k, \tag{2.8}$$

where μ_k is the Lagrange multiplier of a trust region subproblem and W_k is a scaling matrix. In these cases, where the feasible set Θ is a convex subset of $\mathbb{R}^p$, consider an algorithm of the form

$$X_{k+1} = \left(X_k - \left[\hat{H}_k + \mu_k W_k\right]^{-1} \hat{g}_k\right)_{\Theta},$$

where $(\cdot)_{\Theta}$ denotes projection onto Θ.

2.9.1 Estimating the Gradient. Following a response surface methodology approach, QNSTOP designs regression experiments in a region of interest containing the current iterate. QNSTOP uses an ellipsoidal design region centered at the current iterate $X_k \in \mathbb{R}^p$. Let

$$\mathrm{W}_\gamma = \left\{W \in \mathbb{R}^{p \times p} : W = W^{\mathrm{T}},\ \det(W) = 1,\ \gamma^{-1} I_p \preceq W \preceq \gamma I_p\right\}$$

for some $\gamma \geq 1$ where I_p is the $p \times p$ identity matrix. Here γ is fixed at 20. The elements of the set W_γ are valid scaling matrices that control the shape of the ellipsoidal design regions with eccentricity constrained by γ. Let the ellipsoidal design regions

$$\mathrm{E}_k(\tau_k) = \left\{X \in \mathbb{R}^p : (X - X_k)^{\mathrm{T}} W_k (X - X_k) \leq \tau_k^2\right\}$$

where $W_k \in \mathrm{W}_\gamma$. For this implementation $\tau_k = \tau > 0$ is fixed at $\tau = 1$ for each iteration.

In each iteration, QNSTOP methods choose a set of N_k design sites $\{X_{k1}, \ldots, X_{kN_k}\} \subset \mathrm{E}_k(\tau_k) \cap \Theta$. In this implementation $N = N_k$ is fixed for each $k = 1, 2, \ldots$ and $X_{k1}, \ldots, X_{kN} \in \mathrm{E}_k(\tau_k) \cap \Theta$ are uniformly sampled in each iteration.

Let $Y_k = (y_{k1},\ldots,y_{kN})^{\mathrm{T}}$ denote the N-vector of responses where $y_{ki} = F(X_{ki}) + \text{noise}$. The response surface is modeled by the linear model $y_{ki} = \hat{f}_k + X_{ki}^{\mathrm{T}}\hat{g}_k + \epsilon_{ki}$ where ϵ_{ki} accounts for lack of fit. Let $\bar{X}_k = N^{-1}\sum_{i=1}^{N} X_{ki}$. The least squares estimate of the gradient $\hat{g}_k$, ignoring the estimate for $\hat{f}_k$, is obtained by observing the responses and solving

$$\left(D_k^{\mathrm{T}} D_k\right)\hat{g}_k = D_k^{\mathrm{T}} Y_k \tag{2.9}$$

where

$$D_k = \begin{bmatrix} \left(X_{k1} - \bar{X}_k\right)^{\mathrm{T}} \\ \vdots \\ \left(X_{kN} - \bar{X}_k\right)^{\mathrm{T}} \end{bmatrix}.$$

2.9.2 Updating the Model Hessian Matrix. In the stochastic context, QNSTOP methods constrain the Hessian matrix update to satisfy

$$-\eta I_p \preceq \hat{H}_k - \hat{H}_{k-1} \preceq \eta I_p \tag{2.10}$$

for some $\eta \geq 0$. Conceptually, this prevents the quadratic model from changing drastically from one iteration to the next. In (Castle, 2012), a variation of the SR1 (symmetric, rank one) update that satisfies this constraint is proposed. However, this constraint is simply relaxed in the deterministic case and the BFGS update is used, i.e., with the Hessian matrix updated according to

$$\hat{H}_k = \hat{H}_{k-1} + \frac{\hat{H}_{k-1} s_k s_k^{\mathrm{T}} \hat{H}_{k-1}}{s_k^{\mathrm{T}} \hat{H}_{k-1} s_k} + \frac{\nu_k \nu_k^{\mathrm{T}}}{\nu_k^{\mathrm{T}} s_k}$$

where

$$s_k = X_k - X_{k-1},$$

$$\nu_k = \hat{g}_k - \hat{g}_{k-1}.$$

2.9.3 Step Length Control. QNSTOP methods utilize an ellipsoidal trust region concentric with the design region for controlling step length. Typically, in the stochastic case, the volume of the ellipsoid is adjusted from iteration to iteration. Here, the volume of the ellipsoid (controlled by some $\rho > 0$) is fixed with $\rho = 1$, and the following optimization problem is solved:

$$\min_{X \in \mathrm{E}_k(\rho)} \quad \hat{g}_k^{\mathrm{T}}\left(X - X_k\right) + \frac{1}{2}\left(X - X_k\right)^{\mathrm{T}} \hat{H}_k \left(X - X_k\right) \tag{2.11}$$

The solution to (2.11) is on the arc

$$X(\mu) = X_k - \left[\hat{H}_k + \mu W_k\right]^{-1} \hat{g}_k. \tag{2.12}$$

It remains to estimate μ_k such that $X(\mu_k)$ solves (2.11). Using (Dennis, Jr. & Schanbel, 1996) [Lemma 6.4.1], and a little manipulation, it can be established that there is a unique $\mu_k \geq 0$ such that $\|X(\mu_k) - X_k\|_{W_k} = \rho$, unless $\|X(0) - X_k\|_{W_k} \leq \rho$ in which case $\mu_k = 0$. Estimating μ_k is difficult, but well understood. Chapter 7 in (Conn et al., 2000) is a comprehensive treatment. In particular, Algorithm 7.3.6 in (Conn et al., 2000) is robust and easily implemented.

2.9.4 Updating the Experimental Design Region. The QNSTOP approach to constructing the ellipsoidal design regions is considered here. (Stablein et al., 1983) considers confidence regions for the constrained minimizer of a quadratic model fit by regression. An early suggestion for the QNSTOP approach was a convenient ellipsoidal approximation of the confidence set for the minimizer of a quadratic subject to a trust region constraint.

However, if a linear model is fit by least squares and the model Hessian matrix is updated by a secant update then a different approach is warranted. This implementation uses an approximation derived in (Castle, 2012). First, the approximation for the covariance matrix of $\nabla \widehat{m}_k(X_{k+1} - X_k)$,

$$V_k = 4\sigma^2 (D_k^{\mathrm{T}} D_k)^{-1}, \tag{2.13}$$

is computed, where σ^2 is the ordinary least squares estimate of the variance. Then

$$\mathrm{E}_{k+1}(\chi_{p,1-\alpha}) = \left\{ X \in \mathbb{R}^p : (X - X_{k+1})^{\mathrm{T}} W_{k+1} (X - X_{k+1}) \leq \chi^2_{p,1-\alpha} \right\}$$

is an ellipsoidal approximation of the $1-\alpha$ percentile confidence set for the minimizer where

$$W_{k+1} = \left(\hat{H}_k + \mu_k W_k\right)^{\mathrm{T}} V_k^{-1} \left(\hat{H}_k + \mu_k W_k\right).$$

Strictly using the updates for W_{k+1} above can lead to degenerate ellipsoids. To obtain useful design ellipsoids the constraints $\gamma^{-1} I_p \preceq W_{k+1} \preceq \gamma I_p$ and $\det(W_{k+1}) = 1$ are enforced by modifying the eigenvalues— hence, the definition of $\mathrm{W}_\gamma \ni W_{k+1}$.

2.9.5 Algorithm Overview. The QNSTOP implementation used in this thesis is summarized in Algorithm 1. Each run of the algorithm in the experiments was terminated when a budget of function evaluations B had been exhausted.

Algorithm 1. *QNSTOP-GLOBAL*

Step 0 (initialization) : *Fix* $\tau = 1$, $\rho = 1$, $N = 100$, *and* $\gamma \geq 1 = 20$. *Fix scaling matrix* $W_0 = I_p$ *and model Hessian matrix* $\hat{H}_0 = I_p$. *Choose an initial iterate* X_0 *and set* $k = 0$.

Step 1 (regression experiment) : *Uniformly sample* $\{X_{k1}, \ldots, X_{kN}\} \subset \mathrm{E}_k(\tau) \cap \Theta$. *Observe the response vector* $Y_k = (y_{k1}, \ldots, y_{kN})^{\mathrm{T}}$. *Compute* $\hat{g}_k$ *by solving (2.9).*

Step 2 (secant update) : *If* $k > 0$, *compute the model Hessian matrix* $\hat{H}_k$ *using BFGS.*

Step 3 (update iterate) : *Compute* μ_k *using the method described in Section 9.3, solve* $[\hat{H}_k + \mu_k W_k]s_k = -\hat{g}_k$, *and compute* $X_{k+1} = X_k + s_k$.

Step 4 (update subsequent design ellipsoid) : *Compute* $W_{k+1} \in \mathrm{W}_\gamma$ *using the approach described in section 2.9.4.*

Step 5 : *If* $(k+2)N < B$ *then increment* k *by 1 and go to* Step 1. *Otherwise, the algorithm terminates.*

Figure 4 shows a typical progression of QNSTOP over 20 iterations. The solid line represents the lowest value found among 200 design sites for that iteration, while the dotted line represents the corresponding minimum found by the minimizer of the quadratic model. Note that while at times the model will give an imperfect minimum, the overall downward trend is significant.

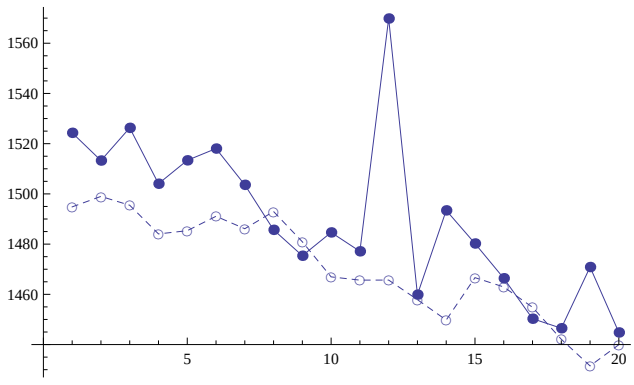

Figure 4. A typical QNSTOP progression.

2.10 Experimental Results and Discussion

Each of the optimization algorithms here is run 50 times using starting points selected from a Latin hypercube design based on the calculated bounds for each of the three problems. For the biomechanics problem, the bounds on each variable are $-1, 1$. While the nonconvex quadratic minimization problem dual is unconstrained and the reflection annihilation problem has nonnegative variables, both have bounds that can be calculated a priori that allow the Latin hypercube design to be constructed. For the nonconvex quadratic minimization problem dual, the bounds on each variable are ± 41.569. For the reflection annihilation problem, the lower and upper bounds are vectors of 0 and 40, respectively. Fortran 90 codes for these test problems may be found in the supplementary material provided.

DIRECT (with $\epsilon = 10^{-4}$), given the constraints of the initial box, is run for each of the problems centered on a point derived from a variation of the Latin hypercube design used for the starting points of the other algorithms: the fifty starting points are all divided by 100 to allow as much of the space to be explored as possible while still differentiating the starting points from each other. The naive simulated annealing algorithm uses the Latin hypercube starting points for only one of the eight processors run in parallel. SPAN similarly uses only eight processors per experiment, since the utilization of more processors results in extreme communication overhead; the number of processors for the naive implementation was limited to the same number of processors that SPAN employed to directly see the advantage of one longer annealing versus the eight shorter ones. The naive parallel implementation of SPSA uses the Latin hypercube starting points in only one of the 640 processors used in parallel for each experiment. KNITRO 8.0 uses the Latin hypercube starting point for only one of its randomly generated multistart points per experiment, and the rest are generated by the built-in pseudorandom generation. Each of the fifty experiments was run on only a single processor for KNITRO 8.0. In a slight difference from the other algorithms, QNSTOP uses 10,000 samples for each of 100 Latin hypercube starting points per experiment, forgoing the standard Latin hypercube starting point in order to provide a more global search strategy. This version of QNSTOP is implemented only as a serial algorithm, so going beyond the simplest parallelization of 100 processors (given the N-parallel nature of step 1) per experiment is not possible.

Table 1
Problem summary.

	No. Unknowns	Classification
Biomech	57	C^0, ill-conditioned
Quad Dual	57	C^0
Wave	56	C^1

It must be noted that the SPAN implementation, while using the same algorithm as the naive implementation, chooses its evaluation points based on a number of different seeds for the pseudorandom number generator equal to the number of processors, to avoid duplicating values. The naive implementation, which uses the same parameters as the SPAN implementation and performs eight annealings (at 125,000 function evaluations each), therefore examines different points than the single annealing performed over multiple processors for the SPAN implementation. In fact, without changing the random seed input for the SPAN implementation, simply changing the number of processors used can change the outcome based on the change to the local seeds. Of course, given the identical algorithms, the tradeoff here is simply one of time for a single annealing performed in parallel versus the time for multiple serial annealings performed simultaneously.

Table 2
Average function evaluations per experiment for each problem.

	DIRECT	SPAN	SA (naive)	SPSA	QNSTOP	KNITRO
Biomech	1008497	1000000	1000000	1000000	1000000	1010492
Quad Dual	1000593	1000000	1000000	1000000	1000000	1075451
Wave	1001585	1000000	1000000	1000000	1000000	1000781

Table 1 displays, for each problem, the number of unknowns and the classification (C^0 means continuous but not differentiable). Note that the problems presented here do not contain any equality constraints, although each problem has bound constraints. The constraints for the biomechanics problem define a convex feasible set, which is handled by conformally mapping $[-1, 1]^{57}$ onto the feasible set.

Table 2 displays the number of function evaluations used by each optimization algorithm as it pursues the global minimum over each of the 50 experiments. Each optimization algorithm is given a function evaluation budget of 10^6 for each experiment and run until it reaches the function evaluation budget or terminates according to the rules of the algorithm.

Table 3
Results for the biomechanics problem.
Best known minimum value: 1222.05 *(QNSTOP/DIRECT).*

	Minimum	1st quartile	2nd quartile	3rd quartile	Maximum
DIRECT	8501	24227	28594	34321	77567
SA (naive)	12606	14414	15677	16352	17723
SPAN	3295	4460	4833	5567	82297
SPSA	33447	46040	56503	61939	84676
KNITRO	19545	28180	30688	35390	40234
QNSTOP	10134	13359	14710	17185	45828

Table 4
Results for the nonconvex quadratic minimization problem dual.
Best known minimum value: -1866.01 *(DIRECT).*

	Minimum	1^{st} quartile	2^{nd} quartile	3^{rd} quartile	Maximum
DIRECT	-1864.32	-1863.03	-1862.21	-1861.79	-1860.00
SA (naive)	-1146.16	-1110.06	-1095.66	-1084.64	-1030.77
SPAN	-1861.53	-1859.93	-1859.20	-1858.69	-1857.43
SPSA	253.30	604.44	688.00	759.65	893.00
KNITRO	-1864.74	-1827.05	-1825.67	-1808.06	-1609.60
QNSTOP	-1863.90	-1862.63	-1862.21	-1861.37	-1860.52

Table 5
Results for the wave annihilation problem.
Best known minimum value: 0 *(DIRECT).*

	Minimum	1^{st} quartile	2^{nd} quartile	3^{rd} quartile	Maximum
DIRECT	$8.19 * 10^{-7}$	$1.02 * 10^{-3}$	$5.76 * 10^{-3}$	$5.74 * 10^{-2}$	$2.7 * 10^{-1}$
SA (naive)	26.87	27.26	27.36	27.53	27.76
SPAN	2.71	3.35	25.20	26.25	26.62
SPSA	12.94	523.35	2902.51	8031.26	206193.00
KNITRO	27.09	28.00	28.00	28.00	28.00
QNSTOP	26.64	27.10	27.19	27.30	27.48

Tables 3–5 display, for each of the problems described, the minimum, maximum, first, second, and third quartile objective function values for each of the algorithms over the fifty experiments, along with the best known minimum for each of the problems and the method of discovery. Note that the experiments that discovered the best known minimum for these problems are not part of the set of experiments listed here, as they tend to have higher function evaluation limits to allow for exhaustive searching.

The worst performer was the naive parallel SPSA, for several reasons. First, SPSA is designed to be a local optimization algorithm, here used in three global optimization applications; attempts to increase the number of starting points to compensate for this shortcoming were rather unsuccessful in the face of the number of dimensions involved in each problem space. Second, the approach taken by SPSA suffers in any case when a large number of variables is encountered, causing it to be much slower to discover local minima. Third, the naive parallel implementation utilized here simply divided the number of function

evaluations by the number of processors; while this may have made the algorithm more likely to start close to a good minimum, the tradeoff was arguably not favorable in any instance except the wave annihilation problem, the most amenable to traditional derivative-based solution methods. Finally, the injected noise technique, while arguably useful for the result obtained for the wave annihilation problem, was certainly not helpful for the quadratic dual. SPSA's best performance was merely mediocre. To determine the best performer, however, requires a look at each problem individually.

Biomechanics problem. SPAN did very well on the biomechanics problem compared to the naive simulated annealing algorithm. In fact, the vast majority of the solutions found by SPAN beat even the best solutions found by the rest of the algorithms for that problem. It's of interest to note the impact that local searching has on improving the minimums for the biomechanics problem, since the only difference between SPAN and the naive annealing is the longer local search time, where the annealing is done over a fairly small neighborhood and the temperature is quite "cool." DIRECT similarly benefits from the transition to local searching inherent in its execution, resulting in a good best result and reasonable results for most of its experiments. The QNSTOP global strategy employed here, Latin hypercube sampling, is perhaps not the best strategy to use to show the strengths of this algorithm when applied to this problem; the best known minimum for this problem is the result of a previous QNSTOP experiment. Even so, the admittedly inferior global strategy utilized here yielded reasonable results compared to the multistart KNITRO, SPSA, and naive parallel simulated annealing strategies, and consistently beat the experiments performed by DIRECT. Note, however, that DIRECT found the best known value in one large run, and none of the other methods, in these or other experiments, ever found this best value. The deterministic DIRECT is guaranteed to monotonically decrease the objective function with more work, whereas the nondeterministic methods (SA, SPAN, SPSA, QNSTOP) are only *likely* to do so.

Quadratic dual problem. Recall that for the nonconvex quadratic minimization problem dual, all the local minima are within 0.5% of the global minimum -1866.01. The layout of this problem is particularly devastating for SPSA, which has difficulty settling into the nondifferentiable local minimum points at which the function is nondifferentiable, resulting in its worst showing; the injected noise technique, which was employed in an attempt to escape the vast number of local minima, did not encourage SPSA to settle for any minimum

within the number of function evaluations employed. DIRECT, QNSTOP, and KNITRO are almost tied for the best results, although DIRECT and QNSTOP are much more consistent in their performance, as KNITRO encounters difficulties as well with the nondifferentiability of the function around the local minima. Finally, once again the lack of local search in the naive parallel simulated annealing is revealed, as the naive annealing experiments found the broad basins that held the local minimum points but failed to refine to a local minimum point. Again DIRECT, in a larger run, found the best known value (theoretical global minimum, for this problem), and similar comments as for the biomechanics problem also apply here.

Wave annihilation problem. DIRECT consistently found the optimum solution for the wave annihilation problem, while the other algorithms (with the notable exception of SPSA) were consistent in finding local minima near 28. SPSA benefited the most with this problem from the "shotgun" approach utilized here, coming in third for the best result found, with the notable downside that its performance was otherwise abysmal. QNSTOP, KNITRO, and the naive simulated annealing all found local minima near 28, while SPAN found good minima in about half of its experiments, while the other half found minima near 28.

The following general conclusions are immediately obvious. First, SPSA is entirely unfit for an optimization problem with a large number of dimensions and a large number of minima. This is not surprising, as this is not the purpose for which SPSA was developed. Secondly, any approach that relied on local gradient information or approximations, which includes SPSA and KNITRO, had an unfortunate tendency to restrict their searches prematurely and thus lost significantly in global exploration compared to the other algorithms presented here, particularly in the biomechanics and wave annihilation problems. While the multistart strategy allowed for some automatic global searching, it was clearly not enough to overcome the difficulties inherent in these problems. Finally, the multistart strategy employed by QNSTOP needs refinement before drawing any conclusions about the true value of this algorithm.

Some general conclusions are also in order about the two newest algorithms considered here — the massively parallel implementation pVTdirect of DIRECT, and the quasi-Newton stochastic algorithm QNSTOP. Because pVTdirect maintains a history of all samples, it makes more efficient use of samples than highly parallel independent sampling stochastic algorithms do, and thus is likely to scale better with more processors. Deterministic algorithms like DIRECT may perform very well on noisy functions (like the biomechanics problem here), and local stochastic algorithms like QNSTOP may perform very well on global optimization problems (as here). In the context of ever increasing parallelism, higher dimensions, and global optimization, algorithms like (deterministic) pVTdirect and (stochastic) QNSTOP, and hybrids thereof, seem well worth pursuing.

Chapter 3.

PROBABILITY-ONE HOMOTOPY METHODS FOR CONSTRAINED CLUSTERING

3.1 Introduction

As machine learning permeates multiple fields of science and engineering, new objective functions are continually being proposed to suit the demands of new application domains. Multicriteria objective functions especially are becoming more prevalent in areas such as mixing labeled and unlabeled data (Balcan & Blum, 2010), (Chapelle et al., 2008), (Sinha & Belkin, 2008), incorporating constraints (Demiriz et al., 2008), (Wang & Davidson, 2010), (Yang & Callan, 2009), and transfer learning (Luo et al., 2008), (Taylor et al., 2008), (Yang et al., 2009b), (Zhang et al., 2011). The difficulty of solving such problems, and even interpreting the solution in a meaningful way, is likewise a growing field of research. A mechanism for solving problems in one field may or may not be adaptable to solving problems in another, and the information yielded by the method may not contain everything needed by the modern researcher.

One such multiobjective formulation is in the area of constrained clustering. In constrained clustering (Basu et al., 2008), the goal is not just to obtain clusters that are local in their respective spaces but that also obey a discrete set of a priori must-link (ML) and cannot-link or must-not-link (MNL) constraints between points. More complicated constraint sets can be represented in simpler form by these ML and MNL constraints, as well; in particular, the conventional cluster hypothesis may be enforced through the use of ϵ– and δ- constraints (Davidson & Ravi, 2005). Although there are many powerful constrained clustering algorithms published in the literature (Dai et al., 2007), (Sato & Iwayama, 2009), (Hossain et al., 2010), (Baghshah & Shouraki, 2011), (He et al., 2012), (Sese et al., 2004), there is currently a lack of a systematic mathematical theory to guide the design of formulations and understand the tradeoffs that invariably result as each algorithm attempts to serve two masters.

The fundamental problem in algorithm design for constrained clustering problems is the tradeoff between conventional clustering objectives and the requirements of the linking constraints. Broadly speaking, there have been two types of algorithms designed to deal with this problem (Davidson & Ravi, 2007). The first uses the constraints to learn a distance function. The second strictly enforces the constraints as the algorithm iterates to a useful solution. The primary problem motivating the development of these two algorithmic approaches is that determining the feasibility of a set of constraints that contains both MNL and ML constraints is an NP-complete problem, being equivalent to the graph coloring problem. When the existence of a feasible solution can not be determined in polynomial time, the usual approach is to fall back on heuristics, with the hope that the resulting solution will be good enough. This dovetails nicely with one of the dominant viewpoints of big-data ML, where rigorous solutions are usually impractical due to the sheer amount of data involved, NP-complete or not, and as such heuristic approaches are the norm for reaching reasonable solutions in a reasonable time.

One traditional solution to such heuristically solved biobjective problems is to introduce a parameter λ that balances or weights competing considerations, in this case cluster locality versus constraint satisfaction. Although there are interesting theoretical insights into the complexity of constrained clustering problems (Davidson, 2012), there is little existing theory available that can deal with (1) how to efficiently compute solutions parametrically as λ varies, (2) how to find and deal with multiple solutions for a fixed λ, and (3) how to canonically define the best choice of λ. Furthermore, using such λ as an independent variable often poses insurmountable problems to the researcher, as will be shown.

Homotopy methods are systematic approaches to characterize solution sets by smoothly tracking solutions from one formulation to another (in this case, from an unconstrained formulation to a constrained formulation). This can allow the effect of changing λ on the quality and nature of the solutions to be mathematically characterized. Smoothly tracking solutions as λ varies provides a holistic understanding of the interplay between the algorithm and a dataset. The resulting tradeoff curve can yield information about the nature of the problem and the probability of improvement offered by constraints.

Corduneanu and Jaakkola (2002) used classical continuation to study how two diverse information sources should be combined in order to arrive at an integrated model. The first application of modern homotopy methods to machine learning was by Ji et al. (2009), who

showed that a general semisupervised formulation for hidden Markov models (HMMs) can be realized using a probability-one homotopy as well.

The key contributions here are:

(1) The *first* homotopy maps, which combine quadratic loss functions with discrete evaluations of constraint violations, for constrained clustering problems are presented. This is a nontrivial task since there are several discrete aspects to the constrained clustering problem (e.g., discrete assignments of points to clusters, discrete satisfactions or violations of constraints) that need to be accommodated in a traditional homotopy framework.

(2) The construction of homotopy maps typically requires careful problem specific tweaking to ensure convergence. The general map constructed here applies to any constrained clustering problem where a distance function is meaningful, similar to existing algorithms for this purpose.

(3) Use of the theory of nonlinear complementarity problem (NCP) functions and the Kreisselmeier-Steinhauser envelope function is new.

(4) Numerous experimental results demonstrating the scalability, viability, usefulness, superiority, and interpretability of the homotopy map approach to constrained clustering are presented, as well as results for a new map applied to ϵ– and δ-style constraints, which constrain intercluster and intracluster distances.

3.2 Mathematical Background

Let superscripts denote vector indices and subscripts denote components of vectors and scalar indices unless otherwise indicated. Let all norms be 2-norms unless otherwise indicated and let all distances be Euclidean distances. Let $\mathbb{R}^n$ denote n-dimensional real Euclidean space and let $\mathbb{R}^{n \times m}$ be the set of real $n \times m$ matrices. Let the ith row of a matrix $A \in \mathbb{R}^{n \times m}$ be denoted by $A_{i\cdot}$ and the jth column by $A_{\cdot j}$. Finally, for a vector $x \in \mathbb{R}^n$, $x > 0$ means all $x_i > 0$, $x \geqq 0$ means all $x_i \geqq 0$, and $x \geq 0$ means $x \geqq 0$ but $x \neq 0$.

Given a set $\hat{X} = \{x^i \mid x^i \in \mathbb{R}^d,\ i = 1, 2, \ldots, k\}$ of k points (cluster representatives) in d dimensions, let $X = \mathrm{vec}\big(x^1, x^2, \ldots, x^k\big) \in \mathbb{R}^{kd}$. Given a set $\hat{Y} = \{y^i \mid y^i \in \mathbb{R}^d,\ i = 1, 2, \ldots, n\}$ of n data points in d dimensions, let $Y = \mathrm{vec}\big(y^1, y^2, \ldots, y^n\big) \in \mathbb{R}^{nd}$. Represent a constraint by the vector $c = (a, b, z, w) \in \mathbb{R}^{2d+2}$ of two data points $a, b \in \hat{Y}$, an identifier $z = \pm 1$, and a degree-of-belief weight $\mathbb{R} \ni w > 0$, where an identifier of $z = 1$ means that

a and b are bound by a must-link constraint (i.e., must be in the same cluster) and an identifier of $z = -1$ means that a and b are bound by a cannot-link constraint (can not be in the same cluster). Given a set $\hat{C} = \{c^i \mid c^i \in \mathbb{R}^{2d+2},\ i = 1,\ 2,\ \ldots,\ q\}$ of q constraints, let $C = \mathrm{vec}\big(c^1, c^2, \ldots, c^q\big) \in \mathbb{R}^{q(2d+2)}$.

3.2.1 Penalty function and constraints. For a data point $y \in \hat{Y}$ and two cluster prototypes $x^i, x^j \in \hat{X}$ define the comparator function $D : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ by

$$D(x^i, x^j, y) = \left(\max\left\{0, \|x^i - y\|^2 - \|x^j - y\|^2\right\}\right)^4.$$

Note that D is three times continuously differentiable, $D \geqq 0$, and $D(x^i, x^j, y) > 0$ if and only if the distance between y and x^i is larger than the distance between y and x^j.

Given $a, b \in \hat{Y}$, let the must-link function $F_m : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^{kd} \to \mathbb{R}$ be defined by

$$F_m(a, b, X) = \prod_{i=1}^{k} \left(\sum_{j=1, j\neq i}^{k} D(x^i, x^j, a) + D(x^i, x^j, b) \right)$$

and let the cannot-link function $F_c : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^{kd} \to \mathbb{R}$ be defined by

$$F_c(a, b, X) = \sum_{i=1}^{k} \left(\prod_{j=1, j\neq i}^{k} D(x^j, x^i, a) D(x^j, x^i, b) \right).$$

Then the following observations are easily verified.

Observation 1. F_m and F_c are nonnegative and three times continuously differentiable.

Observation 2. For any must-link constraint $c = (a, b, 1, w) \in \hat{C}$, the must-link function $F_m(a, b, X) = 0$ if and only if constraint c is satisfied.

Observation 3. For any cannot-link constraint $c = (a, b, -1, w) \in \hat{C}$, the cannot-link function $F_c(a, b, X) = 0$ if and only if constraint c is satisfied.

Observation 4. The penalty function

$$F(C, X) = \sum_{\{i : z_i = 1\}} w_i F_m(a^i, b^i, X) + \sum_{\{i : z_i = -1\}} w_i F_c(a^i, b^i, X)$$

is zero if and only if all the constraints in $\hat{C}$ are satisfied.

By Observation 4, if it is possible to satisfy all of the constraints, then there exists a vector of cluster representatives $\mathcal{X}$ such that $F(C, \mathcal{X}) = 0$. This vector of cluster representatives represents a global minimum point of the function F at which $\nabla_X F(C, \mathcal{X}) = 0$. Unfortunately, if multiple $x^i \in \hat{X}$ coalesce, the resulting D values will result in zero even though there is no clear clustering arising from this case, and $\min_X F(C, X)$ has the trivial solutions $x^1 = \cdots = x^k$. Thus it is necessary to constrain the optimization problem $\min_X F(C, X)$ to prevent such a degenerate case from occurring. In addition, X should be bounded, as $\lim_{\|X\| \to \infty} F(C, X) = 0$ is possible.

First, consider the bounding constraint. A straightforward concave function $\Psi : \mathbb{R}^{kd} \to \mathbb{R}$ to achieve bounding is $\Psi(X) = B - \sum_{i=1}^{n} \|x^i\|^2 \geqq 0$, where $B \in \mathbb{R}$ is a given large constant. Second, to prevent the degenerate condition noted above, a set of constraints $g_i : \mathbb{R}^{kd} \to \mathbb{R}$ can be constructed as $g_i(X) = \epsilon_g - \|x^{i_1} - x^{i_2}\|^2 \leqq 0$, where $1 \leqq i \leqq \binom{k}{2}$, $x^{i_1}, x^{i_2} \in \hat{X}$ are different cluster representatives and $\epsilon_g > 0$ is a small constant. Note that these constraints are differentiable everywhere, and satisfy the reverse convex constraint qualification at $\mathcal{X}$ if $\Psi(\mathcal{X}) > 0$ is inactive. If the active constraints at $\mathcal{X}$ satisfy a constraint qualification (e.g., Arrow-Hurwicz-Uzawa), then the resulting optimization problem

$$\begin{aligned} &\min_X F(C, X) \\ \text{subject to } &-\Psi(X) \leqq 0, \\ &g_i(X) \leqq 0, \quad 1 \leqq i \leqq \binom{k}{2} \end{aligned} \tag{3.1}$$

satisfies the Karush-Kuhn-Tucker (KKT) necessary conditions at $\mathcal{X}$.

Now that the problem has been defined, it remains to show how homotopy methods can be used to effect a smooth transition from a traditional clustering to a clustering that solves the above optimization problem, yielding a useful tradeoff curve. First, however, several tools necessary to utilize the homotopy method are described.

3.2.2 Kreisselmeier-Steinhauser function. Since there are $\binom{k}{2}$ separation constraints in the optimization problem, an aggregation function that can reduce these to a single inequality constraint is of benefit. The Kreisselmeier-Steinhauser envelope function (Kreisselmeier & Steinhauser, 1979)

$$Z(X) = \frac{1}{\kappa} \ln \left[\sum_{i=1}^{\binom{k}{2}} \exp(-\kappa g_i(X)) \right],$$

where $\kappa > 0$ is a regularization parameter, is a common aggregation function used in optimization to reduce the number of inequality constraints to one. Let $g_{\max}(X) = \max_{1 \leq i \leq \binom{k}{2}} g_i(X)$. Note that

$$g_{\max}(X) \leqq Z(X) \leqq g_{\max}(X) + \frac{\ln(k(k-1)/2)}{\kappa},$$

which means that if $Z(X) \leqq 0$ then $g_i(X) \leqq 0$ for all i. As an approximation function, however, there are some drawbacks. The selection of κ is important; a large κ will result in a small difference between the value of Z and $g_{\max}$, but may also cause some numerical difficulties. Furthermore, the feasible region defined by Z is generally smaller than the feasible region defined by $g_{\max}$, as should be obvious from the above inequalities. While each g_i is concave, Z is neither quasiconcave nor pseudoconvex. Nevertheless, except for the degenerate case

$$\nabla\Psi(\mathcal{X})(\nabla Z(\mathcal{X}))^T = \|\nabla\Psi(\chi)\| \, \|\nabla Z(\chi)\|$$

the constraints $-\Psi(X) \leqq 0$, $Z(X) \leqq 0$ satisfy the Arrow-Hurwicz-Uzawa constraint qualification at $\mathcal{X}$. A KKT point $\bar{X}$ for

$$\begin{aligned} &\min_X F(C, X) \\ \text{subject to } &-\Psi(X) \leqq 0, \\ &Z(X) \leqq 0 \end{aligned} \tag{3.2}$$

is generally not a KKT point for (3.1). However, since ϵ_g is a fairly arbitrary value to separate cluster representatives, the distinction between formulations (3.1) and (3.2) is minimal. For a relatively small number of clusters, say $k < 10$, it is possible to select a large κ, say $\kappa = 100$, without encountering numerical difficulties summing the $\binom{k}{2}$ terms in $Z(X)$. Thus, for a moderate number of cluster representatives, $Z(X)$ is a practical way to combine the g_i constraints, reducing the number of dual variables and hence the dimension of the homotopy map.

3.2.3 Positively oriented nonlinear complementarity functions. A continuous function $\hat{\Psi} : R \times R \longrightarrow R$ is called an *NCP function* if $\hat{\Psi}(a, b) = 0 \iff 0 \leqq a \perp b \geqq 0$, and it is *positively oriented* if $\hat{\Psi}(a, b) \geqq 0 \iff a \geqq 0$ and $b \geqq 0$. The positively oriented NCP function of interest here, first introduced by Mangasarian (1976), is

$$\hat{\Phi}(a, b) = -|a - b|^3 + a^3 + b^3.$$

Observe that $\hat{\Phi}$ is C^2; moreover, for $b > 0$, $\hat{\Phi}(\cdot, b)$ is strictly increasing and onto $\mathbb{R}$. NCP functions are used to represent the complementarity conditions within the KKT necessary conditions: for each inequality constraint $g_i \leqq 0$ with associated Lagrange multiplier μ_i, $\hat{\Phi}(-g_i,\mu_i) = 0 \Longleftrightarrow -g_i\mu_i = 0,\ -g_i \geqq 0,\ \mu_i \geqq 0$. Thus, for an optimization problem (3.1) containing only inequality constraints, finding a KKT point is equivalent to solving the nonlinear system of equations

$$\begin{aligned} \nabla_X F(C,X) - \mu_0 \nabla \Psi(X) + \sum_{i=1}^{\binom{k}{2}} \mu_i \nabla g_i(X) &= 0, \\ \hat{\Phi}(\Psi, \mu_0) &= 0, \\ \hat{\Phi}(-g_i, \mu_i) = 0, \quad i = 1, \ldots, \binom{k}{2}&. \end{aligned}$$

3.2.4 Homotopy theory. Standard continuation methods (Watson, 1979, Watson, 2002) find a root $\bar{x}$ of a differentiable function $f(x)$ using a known root x_0 of a simple differentiable function $g(x)$ by solving

$$\rho(\lambda, x) = (1-\lambda)g(x) + \lambda f(x) = 0$$

as λ is increased from 0 to 1, starting with the known solution x_0 at $\lambda = 0$. λ is the continuation parameter, g is called the 'start' function, and f is called the 'target' function. Given a solution (λ, x_λ), standard local methods (such as Newton's method) are used to solve $\rho(\lambda + \delta\lambda, x) = 0$ for fixed small $\delta\lambda > 0$. This yields a series of solutions along a zero curve γ of $\rho(\lambda, x)$. However, there is no guarantee that a given starting function g will yield a zero of f, as the algorithm may fail at some intermediate $\tilde{\lambda}$ as continuation progresses.

Continuation can fail if the zero curve γ of ρ emanating from $(0, x_0)$ fails to exist past some $\tilde{\lambda} < 1$. γ can just stop at $\tilde{\lambda}$, turn back toward $\lambda = 0$ at $\tilde{\lambda}$, or go to infinity. γ may exist past $\tilde{\lambda}$, but bifurcate at $\tilde{\lambda}$, causing the local iteration to fail because $D_x\rho(\lambda, x)$ is singular at the bifurcation point $(\tilde{\lambda}, x_{\tilde{\lambda}})$.

Homotopy methods deal with bifurcation and turning points through a local parametrization of the zero curve $(\lambda, x) = (\lambda(t), x(t))$. Most importantly, homotopy methods treat λ as an independent variable, and do not increase λ monotonically from 0 to 1. The issues of nonexistence, bifurcation, and divergence to infinity are addressed by modern *probability-one homotopy* methods (Watson, 1979a), (Watson, 2002), (Chow et al., 1978), which guarantee

almost surely (in the probability measure theoretic sense) the existence of a smooth, nonbifurcating, bounded zero curve γ of a homotopy map $\rho_a(\lambda, x)$ that connects a start point $(0, x_0)$ to a point $(1, \bar{x})$, where $f(\bar{x}) = 0$.

These algorithms are implemented in FORTRAN 77 as HOMPACK (Watson et al., 1987), and extended in Fortran 90 as HOMPACK90 (Watson et al., 1997). The following theorems about probability-one homotopy maps and the associated zero curves γ are central.

THEOREM 1: PARAMETRIZED SARD'S THEOREM. *Let $U \subset \mathbb{R}^m$, $V \subset \mathbb{R}^n$ be nonempty open sets, $\rho : U \times [0,1) \times V \to \mathbb{R}^n$ be a C^2 map, and define*

$$\rho_a(\lambda, x) = \rho(a, \lambda, x).$$

If ρ is transversal to zero (rank $D\rho = n$ *on* $\rho^{-1}(0)$*), then for almost all $a \in U$ the map ρ_a is also transversal to zero.*

THEOREM 2. *Let $F : \mathbb{R}^n \to \mathbb{R}^n$ and $\rho : \mathbb{R}^m \times [0,1) \times \mathbb{R}^n \to \mathbb{R}^n$ be C^2, and define $\rho_a(\lambda, x) = \rho(a, \lambda, x)$. Assume that*

(1) *ρ is transversal to zero;*

(2) *for each fixed $a \in \mathbb{R}^m$, $\rho_a(0, x) = 0$ has a unique solution x_a at which* rank $D_x \rho_a(0, x_a) = n$*;*

(3) *$\rho_a(1, x) = F(x)$;*

(4) *for each $a \in \mathbb{R}^m$, the connected component of the zero set $\rho_a^{-1}(0)$ containing $(0, x_a)$ is bounded.*

Then for almost all $a \in \mathbb{R}^m$ there exists a zero curve γ of $\rho_a(\lambda, x)$, emanating from $(0, x_a)$, along which the $n \times (n+1)$ Jacobian matrix $D\rho_a(\lambda, x)$ has full rank, that does not intersect itself and is disjoint from any other zeros of ρ_a, and accumulates at a point $(1, \bar{x})$ for which $F(\bar{x}) = 0$. Furthermore, if rank $D\rho_a(1, \bar{x}) = n$*, then the curve γ connecting $(0, x_a)$ to $(1, \bar{x})$ has finite arc length.*

THEOREM 3. *Let $F : \mathbb{R}^n \to \mathbb{R}^n$ be C^2, and suppose there exist $r_0, r > 0$, $r > r_0$, such that for any $a \in \mathbb{R}^n$ with $\|a\|_2 < r_0$, $x - a$ and $F(x)$ do not point in opposite directions on $\{x \in \mathbb{R}^n \mid \|x\|_2 = r\}$. Define $\rho : \mathbb{R}^n \times [0,1) \times \mathbb{R}^n \to \mathbb{R}^n$ by*

$$\rho(a, \lambda, x) = (1 - \lambda)(x - a) + \lambda F(x),$$

and let $\rho_a(\lambda, x) = \rho(a, \lambda, x)$. *Then for almost all vectors* $a \in \mathbb{R}^n$ *with* $\|a\|_2 < r_0$ *there exists a zero curve* γ *of* $\rho_a(\lambda, x)$, *emanating from* $(0, a)$, *along which the* $n \times (n+1)$ *Jacobian matrix* $D\rho_a(\lambda, x)$ *has full rank, that does not intersect itself and is disjoint from any other zeros of* ρ_a, *and accumulates at a point* $(1, \bar{x})$ *for which* $F(\bar{x}) = 0$. *Furthermore, if* rank $D\rho_a(1, \bar{x}) = n$, *then the curve* γ *connecting* $(0, a)$ *to* $(1, \bar{x})$ *has finite arc length.*

Theorem 1 means that the set of points (λ, x) where $\rho_a(\lambda, x) = 0$ looks like the curves in Figure 5 for almost all points $a \in U$. The hypotheses in Theorems 2 and 3 guarantee that the curve γ in Figure 5 is the only curve emanating from $\lambda = 0$ and that γ must accumulate at $\lambda = 1$. Thus, a probability-one homotopy algorithm simply tracks the zero curve γ_a of ρ_a, which is guaranteed to reach a solution $\bar{x}$ of $F(x) = 0$ at $\lambda = 1$, with probability one (almost surely) so long as the hypothesis of Theorems 2 and 3 are met. (Theorem 3 is simply a special case of Theorem 2.)

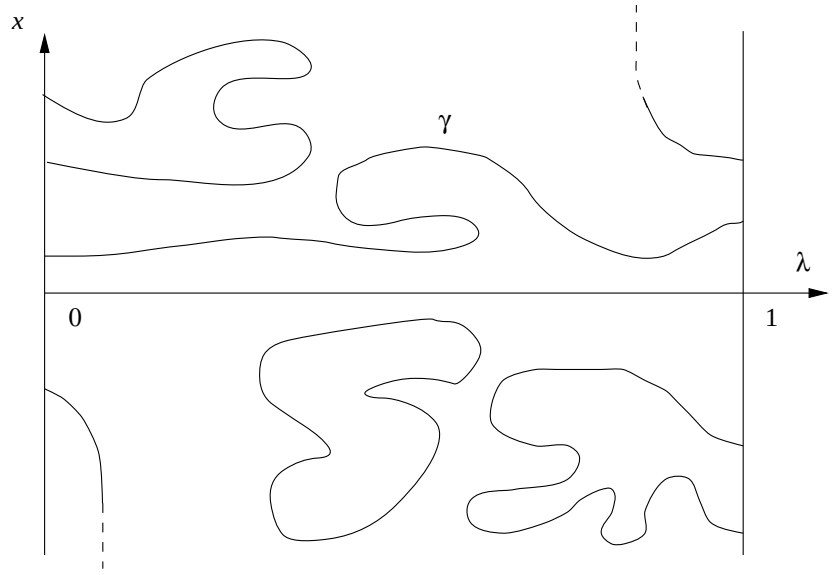

Figure 5. The inverse image $\rho_a^{-1}(0)$ for ρ_a transversal to zero.

In practice, the full rank of the Jacobian matrix $D_{\rho_a}(1, \bar{x})$ is not necessary, as the zero curve usually approaches a solution as $\lambda \to 1$ with finite arc length. This is especially true when applied to the semisupervised clustering problem, as the desired clustering (which satisfies the given constraints) is expected to be present at some point along γ before

$\lambda = 1$. Homotopy maps that fulfill the theorems' assumptions are called *globally convergent probability-one homotopy maps*. Proving a map to be globally convergent with probability one reduces to proving that it meets the given assumptions. Given time to trace the finite arc length of the solution curve with a robust enough tracker, such curves will inevitably yield a useful solution. More valuable, tracing the curve yields the entire parametrized solution trace for analysis, generating a tradeoff curve that can be further analyzed.

It is possible to modify the given "natural" homotopy map $(1-\lambda)g(x) + \lambda f(x)$ by manipulating the λ parameter to yield more useful maps. In particular, where $g(x)$ lacks a unique zero (such as the clustering problem case here), a unique zero x_0 at $\lambda = 0$ can be enforced by modifying the map to

$$\rho_a(\lambda, x) = (1 - \tanh(60\lambda))(x - x_0) \\ + \tanh(60\lambda)\left[(1-\lambda)g(x) + \lambda f(x)\right],$$

where tanh is the hyperbolic tangent function. $\tanh(60\lambda) \approx 1$ for $\lambda > 0.3$, to within 64-bit machine accuracy. Thus $\rho_a(\lambda, x) = 0$ has a unique solution $x = x_0$ at $\lambda = 0$, but for $\lambda > 0.3$ the map looks essentially like $(1-\lambda)g(x) + \lambda f(x)$. Semisupervised learning problems often have such "natural" start functions g with multiple zeros, making this a useful general trick in homotopy map generation. (The rigorous convergence theory for this map actually uses $\tanh\big(60\lambda/(1-\lambda)\big)$, which is computationally indistinguishable from $\tanh(60\lambda)$ for $\lambda > 0.3$.)

3.3 Clustering Application

Let $k^0 \in \mathbb{R}^{kd}$ be some (presumably poor) solution to the unsupervised clustering problem for k clusters in d dimensions, generated by a traditional clustering approach, such as the K-Means algorithm. For the present discussion, consider each cluster assignment to be a "hard" assignment, that is, each data point is assigned to a single cluster determined by its distances from the cluster representatives, not assigned a probability of belonging to each cluster based on those distances.

It is worth noting at this juncture that disjunctive and conjunctive combinations of constraints can be represented by the penalty functions F_m and F_c defined in Section 3.2, which are of particular value when ϵ– and δ–constraints are considered. ϵ– and δ–constraints act upon groups of instances. ϵ-constraints dictate that any data point in a cluster must have another data point in that cluster within ϵ distance, or be the only data point in the

cluster. δ–constraints dictate that any datapoint in a cluster must be at least δ distance from every datapoint that resides in a different cluster. Both of these types of constraints can be represented as disjunctions and conjunctions of must-link constraints (Davidson & Ravi, 2005).

Let C^1 and C^2 be constraints (must-link, cannot-link, or combinatorial) and let F^1 and F^2 be the corresponding penalty functions. Then $C^3 = C^1 \vee C^2$ has the corresponding penalty function $F^3 = F^1 F^2$. Similarly, $C^4 = C^1 \wedge C^2$ has the corresponding penalty function $F^4 = F^1 + F^2$. Observe that $F^3 = 0$ if and only if C^3 is satisfied, and $F^4 = 0$ if and only if C^4 is satisfied. Finally, observe that any number of must-link and cannot-link constraints can thus be combined in conjunctive normal form by summing products of these penalty functions. As such, these penalty functions can easily be adapted to represent penalty functions for ϵ– and δ–constraints.

By Observation 4 in Section 3.2, if it is possible to satisfy all of the constraints defined by C, then there exists a vector of cluster representatives $\mathcal{X}$ such that the penalty function $F(C, \mathcal{X}) = 0$. This vector of cluster representatives represents a global minimum point of the function F at which $\nabla_X F(C, \mathcal{X}) = 0$. This suggests the homotopy map (where $a = k^0$)

$$\check{\rho}_a(\lambda, X) = (1 - \lambda)(X - a) + \lambda \left(\nabla_X F(C, X)\right)^T.$$

This homotopy map is appealing: when $\lambda = 0$, the solution is simply the solution k^0 to the unsupervised clustering problem. When $\lambda = 1$, the solution, if one exists, represents a local minimum point (or stationary point) of the penalty function, which is based on the violation of constraints. This is not to say that the solution generated will satisfy all the constraints if such a solution is possible, as it is fairly easy to construct a degenerate set of constraints so that there is a local solution close to $X = k^0$. However, in practice this has not proven to be a problem.

$\check{\rho}_a$ is a probability-one homotopy map, but while it satisfies conditions (1), (2), and (3) in Theorem 2, it fails to satisfy condition (4), bounded γ. Furthermore, there is a trivial solution to all constraints at $x^1 = x^2 = \cdots = x^k$, where all cluster representatives are equal. Thus, modifications must be made to the above map to accommodate the constraints outlined earlier in Section 3.2.

3.3.1 First Homotopy Map. First, consider the bounding constraint $\Psi(X) = B - \sum_{i=1}^{n} \|x^i\|^2 \geqq 0$, and let $B > \|k^0\|^2$ be a given constant. The Lagrangian of the new

bounded penalty function is $\hat{L}(X,\mu) = F(C,X) - \mu\Psi(X)$, and its derivative, replacing $\nabla_X F(C,X)$, is $\nabla_X \hat{L}(X,\mu) = \nabla_X F(C,X) - \mu\nabla_X \Psi(X)$. This yields a new variable, the Lagrangian multiplier μ, which in turn adds a new function to the map (since the map must be from $\mathbb{R}^{p+1} \to \mathbb{R}^{p}$ for some p), along with the requirement that $\mu \geqq 0$, $\Psi \geqq 0$, and $\mu\Psi = 0$. This naturally leads to the use of the Mangasarian NCP function presented in (Mangasarian, 1976) and modified in (Watson, 1979b), (Watson, 2001).

Define the function $\Phi : [0,1] \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ by

$$\Phi(\lambda, \mu, \Psi(X), h_0) = -|\mu - \Psi(X)|^3 + \mu^3 + \Psi(X)^3 - (1-\lambda)h_0$$

for some constant $h_0 > 0$. The constant term h_0 is designed to force the remaining terms to remain positive for $\lambda < 1$, which enforces the bounding of X for $\lambda < 1$, since $\Psi(X)$ must remain positive when $\Phi = 0$. When $\lambda = 1$, $\Phi(1,\mu,\Psi,h_0) = 0 \iff \mu \geqq 0,\ \Psi \geqq 0,\ \mu\Psi = 0$. The previous homotopy map $\check{\rho}_a$ is then modified to (where now $a = (k^0, h_0)$)

$$\hat{\rho}_a(\lambda, X, \mu) = \begin{pmatrix} (1-\lambda)(X - k^0) + \lambda(\nabla_X \hat{L}(X,\mu))^T \\ \Phi(\lambda, \mu, \Psi(X), h_0) \end{pmatrix}.$$

Note that $\hat{\rho}_a(1, X, \mu) = 0$ is equivalent to the KKT conditions: $\nabla_X \hat{L} = 0,\ \Psi \geqq 0,\ \mu \geqq 0,$ $\mu\Psi = 0$. Unfortunately, while the stated Φ enforces a lower bound on μ, it does not enforce an upper bound on μ; in fact, if $\Psi(X) \to 0$ as $\lambda \to \tilde{\lambda} < 1$, μ must potentially become arbitrarily large to compensate for this. While this map is better than the previous one in that it prevents cluster representatives from migrating arbitrarily far from the data set, it does not prevent μ from growing arbitrarily large, although in practice this is not a common occurrence.

To avoid the trivial solution $x^1 = \cdots = x^k$, an alternative to the constraint functions $g_i(X)$ and $Z(X)$ described in Section 3.2 is to define the function $G : \mathbb{R}^{kd} \to \mathbb{R}$ by

$$G(X) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \max(0, \ell - \|x^i - x^j\|^2)^4, \quad x^i, x^j \in \hat{X}.$$

Then $G \geqq 0$, $G \in C^3$, and $G = 0$ unless two cluster representatives x^i and x^j are less than a distance $\sqrt{\ell}$ from each other, where $\ell > 0$ is a given regularization constant. This represents an equality constraint on the original problem. The updated Lagrangian becomes $\bar{L}(X,\mu,\nu) = F(C,X) - \mu\Psi(X) + \nu G(X)$. In turn, $\nabla_X \bar{L}(X,\mu,\nu) = \nabla_X F(C,X) - \mu\nabla_X \Psi(X) + \nu\nabla_X G(X)$. The additional function is much simpler here, as $G(X)$ is obviously bounded above by

$k(k-1)\ell^4/2$ and below by 0. Let $G(X)$ serve as the final regularization function when $\lambda = 1$, thus fulfilling the equality constraint, and let ν be uniquely determined at $\lambda = 0$ by some initial $\mathbb{R} \ni \nu_0 > 0$. Since it can be assumed that $G(k^0) = 0$ for any reasonable ℓ, and $\Psi(k^0) > 0$, the final hard clustering map is (where now $a = (k^0, h_0, \nu_0)$)

$$\bar{\rho}_a(\lambda, X, \mu, \nu) = \begin{pmatrix} (1-\lambda)(X - k^0) + \lambda(\nabla_X \bar{L}(X, \mu, \nu))^T \\ \Phi(\lambda, \mu, \Psi(X), h_0) \\ (1-\lambda)(\nu - \nu_0) + G(X) \end{pmatrix}.$$

This map is also a probability-one homotopy map. Taking $a = \left(k^0, h_0, \nu_0\right)$ the map $\bar{\rho}(a, \lambda, X, \mu, \nu) = \bar{\rho}_a(\lambda, X, \mu, \nu)$ is transversal to zero — $D_a\bar{\rho} = (\lambda - 1)I$, a multiple of the identity matrix, hence $D\bar{\rho}$ has full rank. For $0 \leqq \lambda \leqq 1$ and $\Psi(k^0) > 0$, Φ is a strictly increasing function of μ, unbounded above, and therefore $\Phi(0, \mu, \Psi(k^0), h_0) = 0$ uniquely determines μ. Thus $\bar{\rho}_a = 0$ has a unique solution at $\lambda = 0$, and a straightforward calculation shows that $D_{(X,\mu,\nu)}\bar{\rho}_a(0, X, \mu, \nu)$ is invertible at this solution. It is also clear from the construction of $\bar{\rho}_a$ that $\bar{\rho}_a(1, X, \mu, \nu) = 0$ is equivalent to the KKT conditions for the problem of minimizing $F(X, C)$ subject to the bounding constraint $\Psi \geqq 0$ and the regularization constraint $G = 0$. Therefore, $\bar{\rho}_a$ satisfies conditions (1), (2), and (3) of Theorem 2, but the bounded γ condition (4) is not satisfied without further assumptions. Conditions for the zero curve γ being bounded (and hence reaching a solution at $\lambda = 1$) are addressed in the next lemma.

LEMMA 1. *Let $\Psi(k^0) > 0$, $G(k^0) = 0$, γ be a zero curve of $\bar{\rho}_a(\lambda, X, \mu, \nu)$ emanating from $\left(0, k^0, \mu_0, \nu_0\right)$ along which $D\bar{\rho}_a$ has full rank, and assume that ν is bounded along γ. Then γ itself is bounded.*

Proof. $\Phi(\lambda, \mu, \Psi(X), h_0) = 0$ along γ implies that, for $\lambda < 1$, $\mu > 0$ and $\Psi(X) > 0$, which in turn implies that X is bounded along γ. Since $0 \leqq \lambda \leqq 1$, and ν is assumed to also be bounded along γ, it suffices to prove that μ is bounded along γ. Assume otherwise, so there exists a sequence of points $\left(\lambda_i, X^i, \mu_i, \nu_i\right)$ on γ with $\mu_i \geqq 0$, $\mu_i \to \infty$. Passing to a subsequence if necessary, it may be assumed (by compactness) that $\left(\lambda_i, X^i, \nu_i\right) \to (\bar{\lambda}, \bar{X}, \bar{\nu})$.

Suppose $\Psi(\bar{X}) > 0$. Then because Φ is strictly increasing and unbounded above ((Mangasarian, 1976), (Watson, 1979b), (Watson, 2001)) $\Phi\left(\bar{\lambda},\ \mu_i,\ \Psi(\bar{X}),\ h_0\right) \to 0$ implies $\{\mu_i\}$ is bounded, a contradiction. Hence $\Psi(\bar{X}) = 0$.

Suppose then that $\Psi(\bar{X}) = B - \|\bar{X}\|^2 = 0$ and $\bar{\lambda} > 0$. In this case $\nabla\Psi(\bar{X}) = -2\bar{X} \neq 0$ and (from the first component of $\bar{\rho}_a = 0$)

$$\mu_i \nabla\Psi(\bar{X}) \to \frac{1-\bar{\lambda}}{\bar{\lambda}}\big(\bar{X} - k^0\big)^T + \nabla_X F(C, \bar{X}) + \bar{\nu}\nabla G(\bar{X})$$

$\Longrightarrow \{\mu_i\}$ is bounded, a contradiction.

The remaining case is $\Psi(\bar{X}) = 0$ and $\bar{\lambda} = 0 \Longrightarrow \nabla\Psi(\bar{X}) \neq 0$ and $\lambda_i\mu_i\big(\nabla\Psi(\bar{X})\big)^T \to \bar{X} - k^0 \Longrightarrow w\big(-\nabla\Psi(\bar{X})\big)^T = k^0 - \bar{X}$ for some $w \geqq 0$. Since $-\Psi(X)$ is convex, $-\nabla\Psi(\bar{X})(k^0 - \bar{X}) \leqq -\Psi(k^0) - (-\Psi(\bar{X})) < 0$. Then $0 \geqq (k^0 - \bar{X})^T\big(-\nabla\Psi(\bar{X})\big)^T w = (k^0 - \bar{X})^T(k^0 - \bar{X}) > 0$, a contradiction. Therefore μ is bounded along γ. ■

Lemma 1 directly yields the next homotopy convergence theorem.

THEOREM 4. *Using the notation of this section, define* $\bar{\rho} : \mathbb{R}^{kd} \times (0,\infty) \times (0,\infty) \times [0,1) \times \mathbb{R}^{kd} \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}^{kd+2}$ *by*

$$\bar{\rho}\big(k^0, h_0, \nu_0, \lambda, X, \mu, \nu\big) = \begin{pmatrix} (1-\lambda)(X - k^0) + \lambda(\nabla_X \bar{L}(X, \mu, \nu))^T \\ \Phi\left(\lambda, \mu, \Psi(X), h_0\right) \\ (1-\lambda)(\nu - \nu_0) + G(X) \end{pmatrix}.$$

Let $\Psi(k^0) > 0$, $G(k^0) = 0$, $a = \big(k^0, h_0, \nu_0\big)$ *and* $\bar{\rho}_a(\lambda, X, \mu, \nu) = \bar{\rho}\big(k^0, h_0, \nu_0, \lambda, X, \mu, \nu\big)$. *Then* $\bar{\rho}$ *is transversal to zero, and for almost all* $a \in \mathbb{R}^{kd} \times (0,\infty) \times (0,\infty)$ *there exists a zero curve* γ *of* $\bar{\rho}_a$, *emanating from* $\big(0, k^0, \mu_0, \nu_0\big)$, *along which the* $(kd+2) \times (kd+3)$ *Jacobian matrix* $D\bar{\rho}_a$ *has full rank, that does not intersect itself and is disjoint from any other zeros of* $\bar{\rho}_a$. *If* ν *is bounded along* γ, *then* γ *accumulates at a point* $(1, \bar{X}, \bar{\mu}, \bar{\nu})$, *where* $(\bar{X}, \bar{\mu}, \bar{\nu})$ *is a KKT point for the constrained clustering problem*

$$\min_X F(C, X) \quad \textit{subject to} \quad -\Psi(X) \leqq 0, \quad G(X) = 0.$$

Furthermore, if $\operatorname{rank} D\bar{\rho}_a(1, \bar{X}, \bar{\mu}, \bar{\nu}) = kd + 2$, *then the curve* γ *connecting* $\big(0, k^0, \mu_0, \nu_0\big)$ *to* $(1, \bar{X}, \bar{\mu}, \bar{\nu})$ *has finite arc length.*

3.3.2 K-Means approximation. In order for the tradeoff curve to reflect an accurate picture of the differences between clustering based solely on the cluster hypothesis and clustering based on the satisfaction of cluster constraints, it is important that the start function $g(x)$ accurately represent the state of the clustering as determined by the cluster hypothesis and that $f(x)$ accurately represents the state of the clustering as determined by the clustering

constraints. The latter case is handled by the optimization problem given above, but the former case requires a clustering formulation that will work in the context of a homotopy map.

The traditional K-Means clustering algorithm is the most popular clustering algorithm based on the cluster hypothesis available. However, the K-Means function $K : \mathbb{R}^{kd} \to \mathbb{R}$ to be minimized,

$$K(X) = \sum_{i=1}^{k} \sum_{y \in S_i} \|y - x^i\|^2,$$

where S_i is the set of all points in cluster i, with representative x^i, is not C^2, since cluster assignment is not differentiable. Ideally, besides being a fair approximation of the K-Means clustering algorithm, the approximation $\hat{K}(X) : \mathbb{R}^{kd} \to \mathbb{R}$ will have two additional qualities: It must be C^3, and $\nabla_X \hat{K}(X) : \mathbb{R}^{kd} \to \mathbb{R}^{kd}$ must be bounded in the feasible region.

The most common such approximation (Phillips, 2009) for a given data set $\hat{Y}$ is

$$\hat{K}(X) = \sum_{i=1}^{|\hat{Y}|} \frac{k}{\sum_{j=1}^{k} \frac{1}{\|y^i - x^j\|^2}}.$$

This continuously differentiable approximation arises from posing the original K-Means clustering problem as a sum of products of the weight or probability $P_{i,j}(X)$ that a data point $y^i \in \hat{Y}$ belongs to a particular cluster represented by $x^j \in \hat{X}$, measured here as

$$P_{i,j} = \frac{\dfrac{1}{\|y^i - x^j\|^2}}{\displaystyle\sum_{m=1}^{k} \frac{1}{\|y^i - x^m\|^2}},$$

and a measure $\tilde{D}_{i,j}$ of the distance that the data point resides from the cluster representative, taken here as $\tilde{D}_{i,j} = \|y^i - x^j\|^2$, a simple square of the Euclidean distance from x^j to y^i. Thus, points very close to their cluster representatives have high probability of belonging to that cluster, but bring a corresponding low value to the final optimization function, since such points are considered ideal. Summing the products yields the approximation $\hat{K}(X) = \sum_{i=1}^{|\hat{Y}|} \sum_{j=1}^{k} P_{i,j} \tilde{D}_{i,j}$ after cancellation. Note, however, that this cancellation may yield overflow if $\|y^i - x^j\| \approx 0$, in which case the summand for that index i is taken as zero. These

singularities are removable, and $\hat{K}(X)$ is an entire function (in each of the components of X, viewed as a complex vector). Minimizing $\hat{K}(X)$ thus yields an approximation of the minimum of the original K-Means function. Furthermore, $\nabla_X \hat{K}(X)$ is bounded if X is bounded, and the components of $\nabla_X \hat{K}(X)$ are also entire functions in each of the components of (complex) X. Precisely,

$$\frac{\partial \hat{K}(X)}{\partial x_c^b} = \sum_{i=1}^{|\hat{Y}|} \frac{2k(x_c^b - y_c^i)}{\|y^i - x^b\|^4 \left(\sum_{j=1}^{k} \frac{1}{\|y^i - x^j\|^2}\right)^2}$$

is bounded if X is bounded, with the same removable singularities as $\hat{K}(X)$.

Note that while the problem has been represented here using the squared 2-norm as the measure of distance, points can be "spread out" by using higher order (even) p-norms raised to the pth power. In fact, any C^3 nonnegative function could be used as the distance measure between data points; the squared 2-norm is simply the most convenient one for the present purpose.

3.3.3 Second Homotopy map. Generally inequality constraints are easier to deal with than equality constraints, so consider replacing the equality constraint $G(X) = 0$ used for the first homotopy map $\bar{\rho}_a$ by the inequality constraint $Z(X) \leqq 0$ discussed earlier. Keep the same bounding constraint $\Psi(X) \geqq 0$. Using the same modified NCP function Φ as before, the equation

$$\Phi(\lambda, \mu, \Psi(X), h_0) = 0$$

for $h_0 > 0$, $\Psi(k^0) > 0$, $\Phi(0, \mu_0, \Psi(k^0), h_0 = 0$, and $0 \leqq \lambda < 1$ forces $\Psi(X) > 0$ along the zero curve γ. Similarly the equation

$$\Phi(\lambda, \nu, -Z(X), h_1) = 0$$

for $h_1 = 0$, $Z(k^0) < 0$, $\Phi(0, \nu_0, -Z(k^0), h_1) = 0$, and $0 \leqq \lambda < 1$ forces $Z(X) < 0$ along γ. When $\lambda = 1$, these two equations enforce the KKT conditions for the constraints $-\Psi(X) \leqq 0$, $Z(X) \leqq 0$ and their Lagrange multipliers μ, ν, respectively.

The Lagrangian function associated with (3.2) is $\tilde{L}(X, \mu, \nu) = F(C, X) - \mu\Psi(X) + \nu Z(X)$, and a KKT point $(\bar{X}, \bar{\mu}, \bar{\nu})$ for (3.2) satisfies

$$\nabla_X \tilde{L}(X, \mu, \nu) = 0,$$

$$0 \leqq \mu \perp \Psi(X) \geqq 0,$$

$$0 \leqq \nu \perp -Z(X) \geqq 0.$$

Furthermore, should $Z(\bar{X}) < 0$, the KKT point $(\bar{X}, \bar{\mu}, 0)$ for (3.2) yields a KKT point $(\bar{X}, \bar{\mu}, 0, \ldots, 0)$ for (3.1).

Finally, putting all the pieces together, the proposed constrained clustering homotopy map is

$$\tilde{\rho}_a(\lambda, X, \mu, \nu) = \begin{pmatrix} (1 - \tanh(60\lambda))(X - k^0) + \tanh(60\lambda)\varphi(\lambda, X, \mu, \nu) \\ \Phi\left(\lambda, \mu, \Psi(X), h_0\right) \\ \Phi\left(\lambda, \nu, -Z(X), h_1\right) \end{pmatrix},$$

where

$$\varphi(\lambda, X, \mu, \nu) = \left((1-\lambda)\nabla_X \hat{K}(X) + \lambda \nabla_X \tilde{L}(X, \mu, \nu)\right)^T,$$

$a = (k^0, h_0, h_1)$ and k^0 is any point for which $\Psi(k^0) > 0$, $Z(k^0) < 0$, and $\nabla_X \hat{K}(k^0) \approx 0$, e.g., a K-Means solution (locally) minimizing $K(X)$.

The tanh terms in the above construction arise because $\nabla_X \hat{K}(X) = 0$ has multiple possible solutions. At the very least, permutations of the cluster representatives in X will yield identical values for $\nabla_X \hat{K}$. The tanh terms ensure that $\tilde{\rho}_a(0, X, \mu, \nu) = 0$ has a unique solution as required by Theorem 2. Since $h_0 > 0$, $\Phi(0, \mu, \Psi(k^0), h_0) = 0$ uniquely determines $\mu = \mu_0 > 0$, and similarly $h_1 > 0$, $\Phi(0, \nu, -Z(k^0), h_1) = 0$ uniquely determines $\nu = \nu_0 > 0$.

Computationally, as mentioned earlier, $\tanh(60\lambda) = 1$ in 64-bit arithmetic for $\lambda > 0.3$, and thus, for $\lambda > 0.3$, this map functions identically to

$$\begin{pmatrix} \left((1-\lambda)\nabla_X \hat{K}(X) + \lambda \nabla_X \tilde{L}(X, \mu, \nu)\right)^T \\ \Phi\left(\lambda, \mu, \Psi(X), h_0\right) \\ \Phi\left(\lambda, \nu, -Z(X), h_1\right) \end{pmatrix}.$$

3.3.4 Convergence proof. (1) Taking $a = \left(k^0, h_0, h_1\right)$ the map $\tilde{\rho}(a, \lambda, X, \mu, \nu) = \tilde{\rho}_a(\lambda, X, \mu, \nu)$ is transversal to zero: Observe that

$$D_a \tilde{\rho}(a, \lambda, X, \mu, \nu) = \operatorname{diag}\big(-(1 - \tanh(60\lambda))I, -(1-\lambda), -(1-\lambda)\big),$$

which has rank $kd + 2$ for $0 \leqq \lambda < 1$.

(2) $\tilde{\rho}_a = 0$ has a unique solution at $\lambda = 0$: For $0 \leqq \lambda \leqq 1$ and $\Psi(k^0) > 0$, $\Phi(\lambda, \mu, \Psi(k^0), h_0)$ is a strictly increasing function of μ, unbounded above, and therefore $\Phi_0(0, \mu, k^0, h_0) = 0$ uniquely determines $\mu = \mu_0$. Similarly, for $0 \leqq \lambda \leqq 1$ and $Z(k^0) < 0$, $\Phi(\lambda, \nu, -Z(k^0), h_1)$ is a strictly increasing function of ν, unbounded above, and therefore

$\Phi(0, \nu, -Z(k^0), h_1) = 0$ uniquely determines $\nu = \nu_0$. A straightforward calculation shows that $D_{(X,\mu,\nu)}\tilde{\rho}_a(0, k^0, \mu_0, \nu_0)$ is invertible.

(3) It is clear from the construction of $\tilde{\rho}_a$ that $\tilde{\rho}_a(1, X, \mu, \nu) = 0$ is equivalent to the KKT necessary conditions for the problem (3.2).

Therefore, $\tilde{\rho}_a$ satisfies conditions (1), (2), and (3) of Theorem 2, but the bounded γ condition (4) is not satisfied without further assumptions. Conditions for the zero curve γ being bounded (and hence reaching a solution at $\lambda = 1$) are addressed in the next lemma.

LEMMA 2. *Let $\Psi(k^0) > 0$, $Z(k^0) < 0$, γ be a zero curve of $\tilde{\rho}_a(\lambda, X, \mu, \nu)$ emanating from $\big(0, k^0, \mu_0, \nu_0\big)$ along which $D\tilde{\rho}_a$ has full rank, and assume that ν is bounded along γ. Then γ itself is bounded for $0 \leqq \lambda \leqq 1$.*

Proof. For $0 \leqq \lambda < 1$, $\Phi(\lambda, \mu, \Psi(X), h_0) = 0$ along γ implies that $\mu > 0$ and $\Psi(X) > 0$, which in turn implies that X is bounded along γ. Since $0 \leqq \lambda \leqq 1$, and ν is assumed to also be bounded along γ, it suffices to prove that μ is bounded along γ. Assume otherwise, so there exists a sequence of points $\big(\lambda_i, X^i, \mu_i, \nu_i\big)$ on γ with $\mu_i \geqq 0$, $\mu_i \to \infty$. Passing to a subsequence if necessary, it may be assumed (by compactness) that $\big(\lambda_i, X^i, \nu_i\big) \to (\bar{\lambda}, \bar{X}, \bar{\nu})$.

Suppose $\Psi(\bar{X}) > 0$. Then because Φ is strictly increasing in μ and unbounded above ((Mangasarian, 1976), (Watson, 1979b), (Watson, 2001)) $\Phi\big(\bar{\lambda},\ \mu_i,\ \Psi(\bar{X}),\ h_0\big) \to 0$ implies $\{\mu_i\}$ is bounded, a contradiction. Hence $\Psi(\bar{X}) = 0$.

Suppose then that $\Psi(\bar{X}) = B - \|\bar{X}\|^2 = 0$ and $\bar{\lambda} > 0$. In this case $\nabla_X \Psi(\bar{X}) = -2\bar{X}^T \neq 0$ and (from the first component of $\tilde{\rho}_a = 0$) $\mu_i \nabla_X \Psi(\bar{X}) = -2\mu_i \bar{X}^T \to \frac{1-\tanh(60\bar{\lambda})}{\bar{\lambda}\tanh(60\bar{\lambda})}\big(\bar{X} - k^0\big)^T + \frac{1-\bar{\lambda}}{\bar{\lambda}}\nabla_X \hat{K}(\bar{X}) + \nabla_X F(C, \bar{X}) + \bar{\nu}\nabla_X Z(\bar{X}) \Longrightarrow \{\mu_i\}$ is bounded, a contradiction.

The remaining case is $\Psi(\bar{X}) = 0$ and $\bar{\lambda} = 0 \Longrightarrow \nabla_X \Psi(\bar{X}) = -2\bar{X}^T \neq 0$ and

$$\tanh(60\lambda_i)\lambda_i\mu_i\big(\nabla_X \Psi(\bar{X})\big)^T \to \bar{X} - k^0$$

$\Longrightarrow w\big(-\nabla_X \Psi(\bar{X})\big)^T = k^0 - \bar{X} \neq 0$ for some $w > 0$, since $0 = \Psi(\bar{X}) \neq \Psi(k^0) > 0$. Since $-\Psi(X)$ is convex, $-\nabla_X \Psi(\bar{X})(k^0 - \bar{X}) \leqq -\Psi(k^0) - (-\Psi(\bar{X})) < 0$. Then $0 > (k^0 - \bar{X})^T\big(-\nabla_X \Psi(\bar{X})\big)^T w = (k^0 - \bar{X})^T(k^0 - \bar{X}) > 0$, a contradiction. Therefore μ is bounded along γ for $0 \leqq \lambda \leqq 1$. ∎

Note that ν and μ can both go to infinity as $\lambda \to 1$ and the h_0 and h_1 terms vanish from $\Phi(\lambda, \mu, \Psi(X), h_0)$ and $\Phi(\lambda, \nu, -Z(X), h_1)$, which corresponds to two or more mean prototypes approaching each other as both approach the boundary of the region. This indicates multiple active constraints. The solution $\bar{X}$ approached as $\lambda \to 1$ will still yield

a KKT point for (3.2), although the Lagrange multipliers will not be available (however, they may be easily verified to be greater than zero in such a case!).

Lemma 2 and the earlier discussion of $\tilde{\rho}$ directly yield the next homotopy convergence theorem.

THEOREM 5. *Using the notation of this section, define* $\tilde{\rho} : \mathbb{R}^{kd} \times (0,\infty) \times (0,\infty) \times [0,1) \times \mathbb{R}^{kd} \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}^{kd+2}$ *by*

$$\tilde{\rho}\big(k^0, h_0, h_1, \lambda, X, \mu, \nu\big) = \begin{pmatrix} (1-\tanh(60\lambda))(X-k^0) + \tanh(60\lambda)\varphi(\lambda, X, \mu, \nu) \\ \Phi\left(\lambda, \mu, \Psi(X), h_0\right) \\ \Phi\left(\lambda, \nu, -Z(X), h_1\right) \end{pmatrix},$$

where

$$\varphi(\lambda, X, \mu, \nu) = \big((1-\lambda)\nabla_X K(X) + \lambda \nabla_X \tilde{L}(X, \mu, \nu)\big)^T.$$

Let $\Psi(k^0) > 0$, $Z(k^0) < 0$, $a = \big(k^0, h_0, h_1\big)$, *and*

$$\tilde{\rho}_a(\lambda, X, \mu, \nu) = \tilde{\rho}\big(k^0, h_0, h_1, \lambda, X, \mu, \nu\big).$$

Then for almost all $a \in \mathbb{R}^{kd} \times (0,\infty) \times (0,\infty)$ *there exists a zero curve* γ *of* $\tilde{\rho}_a$, *emanating from* $\big(0, k^0, \mu_0, \nu_0\big)$, *along which the* $(kd+2) \times (kd+3)$ *Jacobian matrix* $D\tilde{\rho}_a$ *has full rank, that does not intersect itself and is disjoint from any other zeros of* $\tilde{\rho}_a$. *If* ν *is bounded along* γ , *then* γ *accumulates at a point* $(1, \bar{X}, \bar{\mu}, \bar{\nu})$, *where* $(\bar{X}, \bar{\mu}, \bar{\nu})$ *is a KKT point for the constrained clustering problem*

$$\min_X F(C, X) \quad \textit{subject to} \quad -\Psi(X) \leqq 0, \quad Z(X) \leqq 0.$$

Furthermore, if $\operatorname{rank} D\tilde{\rho}_a(1, \bar{X}, \bar{\mu}, \bar{\nu}) = kd+2$, *then the curve* γ *connecting* $\big(0, k^0, \mu_0, \nu_0\big)$ *to* $(1, \bar{X}, \bar{\mu}, \bar{\nu})$ *has finite arc length.*

3.4 Experimental Results

Experiments to discover the effectiveness of the homotopy tracking algorithm with the proposed homotopy map, as compared to popular existing constrained clustering algorithms, are presented here. The constraints used involve combinations of ML and MNL constraints

(solving problems involving solely ML constraints are fairly straightforward polynomial time graph problems).

The existence of MNL constraints in the constraint sets is crucial to understanding the complexity of the test problems. Davidson et al. (2006) state that as a rough rule of thumb a set of constraints can be understood as fundamentally "difficult" for these iterative K-Means approaches if any single datapoint appears in k or more MNL constraints. As such, for each dataset presented here, both an "easy" and a "difficult" set of constraints were generated. The "easy" constraint set involves one hundred constraints such that no datapoint appears more than $k-1$ times in a mix of ML and MNL constraints. The "difficult" constraint set, also one hundred constraints, involves predominately MNL constraints, and guarantees that at least one datapoint is involved in k MNL constraints. In both cases, the generated constraints were completely random, with no a priori knowledge about how well the generated constraints would guide the algorithms to a correct solution.

The datasets involved are all taken from the UCI machine learning dataset repository (Bache & Lichman, 2013). They represent a balanced selection of moderately easy clustering problems without constraints, and should demonstrate some of the key differences between the homotopy algorithms utilizing the maps $\bar{\rho}$ and $\tilde{\rho}$ developed here and the K-Means algorithms used previously. The datasets are "Liver Disorders" (liver), "Pima Indians Diabetes" (pima), "Steel Plates Faults" (faults), "Wine" (wine), "Iris" (iris), "Ionosphere" (iono), "Glass Identification" (glass), and "PAMAP2 Physical Activity Monitoring" (pamap). The datasets "faults" and "pamap" were modified in the following manner: The first three classification categories of the dataset "faults" were treated as additional data, and the last classification category was used for classification. The dataset "pamap" was modified by eliminating all data points of the "0" classification (as recommended by the contributors)

TABLE 6
Dataset Summary

	No. Instances	No. Features	No. Categories
liver	345	6	2
pima	768	7	2
faults	1941	31	2
wine	178	12	3
iris	150	4	3
iono	351	34	2
glass	214	10	6
pamap	175498	53	12

and any data point with a "NaN" data value. See Table 6 for the relevant details for each dataset.

The K-Means algorithms used for the comparison are those presented by Bilenko et al. (2004): metric pairwise constrained K-Means (MPCK-Means), metric learning K-Means without pairwise constraints (MK-Means), and pairwise constrained K-Means without metric learning (PCK-Means). The standard K-Means result is also presented, to be used as a baseline. These algorithms were chosen for several reasons. First, K-Means is by far the most popular clustering algorithm, if only because of its intuitive approach and ease of programming; thus, K-Means algorithms modified for constrained clustering are the most likely to be consulted by researchers who are interested in constrained clustering problems. Second, these constrained K-Means algorithms minimize a summed penalty function based on the distance from the cluster centroids to the data points assigned to that cluster. While this penalty function may be discrete, it is still similar enough to the penalty function presented here to make comparisons between these algorithms and the homotopy approach feasible.

It is worth noting immediately that three things set the homotopy algorithms apart from the K-Means algorithms presented here. First, for the K-Means algorithms, the ordering of the constraints plays a nonnegligible role in the quality of the final result, meaning that finding the best result theoretically involves searching every permutation of a given constraint set (which is not computationally feasible). For a homotopy algorithm, the ordering of the constraints is unimportant. Second, not only are problems involving concentrations of MNL

constraints involving the same datapoint not qualitatively more "difficult" for homotopy algorithms, but, since distances only need to be calculated once per iteration, problems involving concentrations of datapoints are computationally less intense than problems where the constraints are more diverse, at least until each datapoint is involved in at least one constraint. Finally, the homotopy algorithms, like the K-Means algorithm, is limited to convex clusterings, which for some datasets can be potentially debilitating. In contrast, the adapted K-Means algorithms presented here distinguish between cluster assignment and cluster centroids, which allows for nonconvex clusterings.

Ten experiments are conducted for each algorithm on each dataset, and the tables report the minimum, median, and maximum of the stated index for the ten experiments. Table 7 shows the adjusted Rand index of each dataset measured against the proper classification, for each algorithm discussed here, for 100 "easy" constraints. Table 8 shows the same data for 100 "hard" constraints. The exception is the "pamap" dataset, which used 250 of each constraint to allow for more differentiation (due to the massive size of the dataset). Note that these two tables show the adjusted Rand index for the best cluster found along the homotopy method's reported trace, excluding the K-Means solution starting point. For both of these tables the simpler homotopy map $\bar{\rho}$ was used.

3.5 ϵ- and δ-constraints

One reasonable question that arises in semisupervised learning is what kinds of information would need to be present in a clustering problem that could not be represented by the datapoints themselves. One answer makes reference to the cluster hypothesis itself.

The cluster hypothesis states that if two datapoints are close to each other (for a vague notion of closeness), then they should belong to the same cluster; if they are far apart, they should belong to different clusters. The constraints that formalize this statement are known as ϵ– and δ–constraints; ϵ-constraints are constraints that dictate that any data point in a cluster must have another data point in that cluster within ϵ distance, or represent the entire cluster, while δ–constraints state that any datapoint in a given cluster must be at least δ distance from every datapoint in a different cluster. Both of these constraints can be represented as disjunctions and conjunctions of the classic "must-link" and "must-not-link" constraints (Davidson & Ravi, 2006). The number of such constraints can grow quite large as the number of datapoints in the set increases (depending on the values assigned to ϵ and

TABLE 7
adjusted Rand index, "easy" constraints

	K-Means	MK-Means	PCK-Means	MPCK-Means	Homotopy $\bar{\rho}$
liver	−0.0064	−0.0036	−0.0042	−0.0045	−0.0040
	−0.0064	−0.0036*	−0.0042	−0.0045	−0.0040
	−0.0064	−0.0036	−0.0042	−0.0045	−0.0040
pima	0.0744	0.0720	0.0510	0.0164	0.1322
	0.0744	0.0870	0.0510	0.0164	0.1322*
	0.0744	0.1040	0.0510	0.0164	0.1322
faults	0.1358	−0.1028	0.1109	−0.0863	0.1127
	0.1358*	−0.1028	0.1133	−0.0849	0.1127
	0.1358	−0.1028	0.1159	−0.0837	0.1127
wine	0.3711	0.7549	0.3420	0.6211	0.4377
	0.3711	0.7692*	0.3420	0.6211	0.4377
	0.3394	0.8309	0.3420	0.6211	0.4377
iris	0.4225	0.4290	0.5195	0.5234	0.8841
	0.7163	0.8857*	0.5195	0.5234	0.8841*
	0.7302	0.8857	0.5195	0.5234	0.8841
iono	0.1728	0.1776	0.1122	0.1122	0.2450
	0.1776	0.1776	0.1122	0.1122	0.2450*
	0.1776	0.1776	0.1122	0.1122	0.2450
glass	0.1790	0.2023	0.1720	0.1824	0.1967
	0.1790	0.2023*	0.1720	0.1824	0.1967*
	0.2258	0.2482	0.1720	0.1824	0.2258
pamap	0.6457	0.3046	0.5560	0.2695	0.6457
	0.6457*	0.3046	0.5560	0.2695	0.6457*
	0.6457	0.3046	0.5560	0.2695	0.6457

δ), but the entire set of constraints need not be brought to bear for the solution to show improvement. One advantage of such constraints is that they don't depend on the "real" clustering, which is to say the classification, of a given dataset, which is often unknown in practice. Thus, applying these constraints to test problems can yield tests of improvement in whatever measure of cluster hypothesis satisfaction is desired (of which there are many).

100 random constraints were generated using ϵ– and δ–constraints based on reasonable values for the given data sets. Since the adjusted Rand index is useless in this context, the Davies-Bouldin index (DBI) was used instead. The DBI is a nonnegative measure of conformity to the cluster hypothesis; a lower DBI indicates closer conformity to the clustering hypothesis. The "pamap" dataset was not used due to the difficulty in generating meaningful differences in the clusterings with this sort of constraint. Table 9 shows these results. The

TABLE 8
adjusted Rand index, "hard" constraints

	K-Means	MK-Means	PCK-Means	MPCK-Means	Homotopy $\bar{\rho}$
liver	−0.0064	−0.0046	−0.0109	−0.0077	0.0400
	−0.0064	−0.0046	−0.0080	−0.0052	0.0400*
	−0.0064	−0.0046	0.0102	0.0253	0.0400
pima	0.0744	0.0722	0.0652	0.0114	0.0775
	0.0744	0.0722	0.0696	0.0340	0.0775*
	0.0744	0.0722	0.0696	0.0422	0.0775
faults	0.1358	−0.1028	0.1324	−0.0839	0.1119
	0.1358	−0.1028	0.1420*	−0.0839	0.1119
	0.1358	−0.1028	0.1459	−0.0832	0.1119
wine	0.3394	0.7692	0.3265	0.6731	0.8666
	0.3711	0.7840	0.3818	0.7031	0.8666*
	0.3711	0.8170	0.4451	0.8636	0.8666
iris	0.4225	0.4290	0.5127	0.5411	0.9216
	0.7163	0.8857	0.6779	0.8017	0.9216*
	0.7302	0.8857	0.8015	0.9222	0.9216
iono	0.1728	0.1776	0.1049	0.1122	0.1943
	0.1776	0.1776	0.1413	0.1122	0.1943*
	0.1776	0.1776	0.1413	0.1122	0.1943
glass	0.0162	0.0000	0.1849	0.1560	0.0162
	0.1790	0.2023	0.2422*	0.1741	0.1790
	0.2258	0.2482	0.2608	0.2102	0.2258
pamap	0.6457	0.2929	0.6454	0.2700	0.6513
	0.6457	0.2929	0.6454	0.2700	0.6513*
	0.6457	0.2929	0.6454	0.2700	0.6513

(more computationally expensive) homotopy map $\tilde{\rho}$ was used here to demonstrate the utility of incorporating the K-Means approximation function into the map. In these experiments the best cluster found by the homotopy algorithm was also uniformly the last one.

3.6 Discussion

For the experiments that use the true classifications of these datasets to show the validity of the "easy" constraints (Table 7), the homotopy method performed well for the pima, iris, and iono datasets, and performed well for the pamap dataset when compared to the other constrained clustering metrics for the "easy" constraint set, although none of them

TABLE 9
Davies-Bouldin index, ϵ– and δ–constraints

	K-Means	MK-Means	PCK-Means	MPCK-Means	Homotopy $\bar{\rho}$
liver	1.7349	2.3067	1.7679	1.2516	0.8706
	1.7349	1.8801	1.4568	1.2516	0.8706*
	1.7349	1.6682	1.3542	1.2516	0.8706
pima	1.9995	0.9883	0.8762	0.8681	0.8094
	1.5653	1.9403	1.0585	1.4436	0.8601*
	1.5387	1.9316	1.0585	1.4436	0.8601
faults	0.9392	0.9883	0.8762	0.8681	0.8094
	0.9392	0.9652	0.8762	0.8681	0.8094*
	0.9392	0.9652	0.8637	0.8681	0.8094
wine	1.5126	1.6650	0.8185	1.5393	0.6604
	1.5126	1.5507	0.6542	1.4515	0.6097*
	1.5126	1.4506	0.6101	1.3447	0.4948
iris	0.7373	1.5023	1.4662	0.9612	0.9379
	0.7373	0.9455	0.8877	0.7175	0.6453*
	0.7373	0.7445	0.7041	0.6585	0.5776
iono	2.0706	2.0512	1.6898	1.6898	1.6188
	2.0706	1.8936	1.8936	1.6898	1.6188*
	2.0706	1.8936	1.8919	1.6898	1.6188
glass	3.4599	1.8348	1.0414	1.8284	2.2789
	2.2910	1.4204	1.0414*	1.2820	1.2204
	1.7415	1.0621	1.0414	1.0038	0.2403

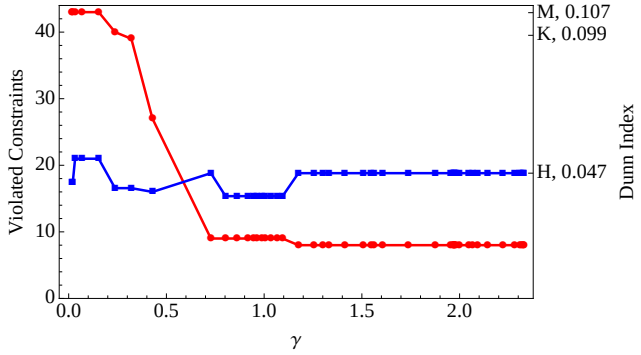

Figure 6. The iris dataset with "easy" constraints. The Dunn index is tracked against the arc length of γ in blue, while the satisfied constraints are tracked in red. The Dunn Indices for the final homotopy ($\bar{\rho}$) clustering ("H"), MK-Means clustering ("M"), and K-Means clustering ("K") are also shown.

managed to improve on the K-Means starting points for that problem. The metric learning without pairwise constraints algorithm (M-Kmeans) performed about as well, performing the best in the liver, wine, iris, and glass datasets. It is worth noting that for the two largest datasets in terms of instance numbers (faults and pamap), no constrained clustering algorithm approached the actual classification better than the straightforward K-Means

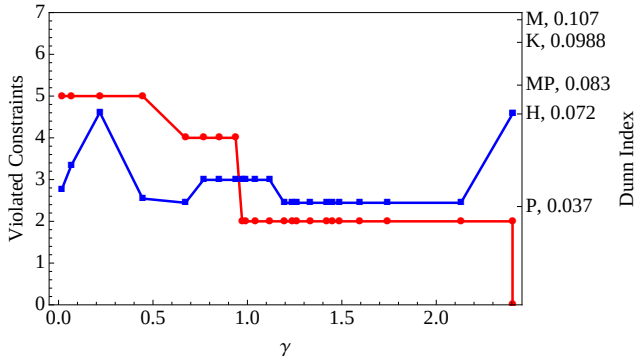

Figure 7. The iris dataset with "hard" constraints. The Dunn Indices for the final homotopy ($\bar{\rho}$) clustering ("H"), MK-Means clustering ("M"), PK-Means clustering ("P"), MPK-Means clustering ("MP"), and K-Means clustering ("K") are also shown.

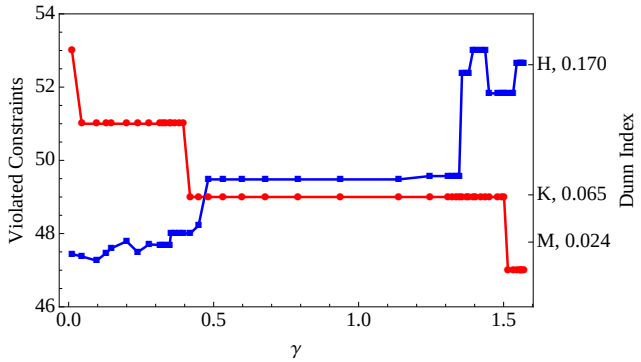

Figure 8. The liver dataset with "easy" constraints.

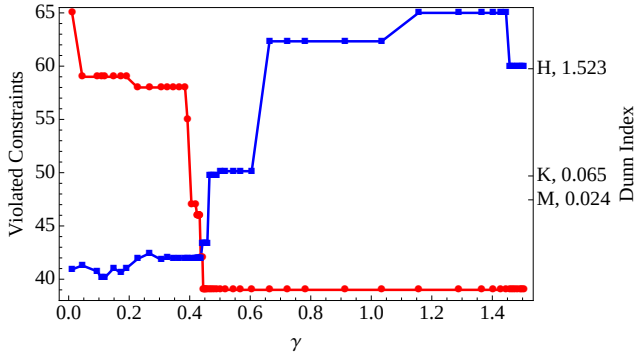

Figure 9. The liver dataset with "hard" constraints.

solution. This isn't entirely surprising, as a dense population of datapoints means that any rearrangement of clustering, even one based on constraints known to be correct, is going to inevitably cause a transfer of instances from a cluster where they satisfy the clustering hypothesis to one where they do not, with an expected degradation of quality in the resulting partition.

The "hard" dataset (Table 8) saw a much better showing by the homotopy method, which performed the best for the liver, pima, wine, iris, iono, and pamap datasets. The faults and glass datasets were, in this case, best captured by the pairwise constrained K-Means algorithm. This demonstrates in particular the power of the homotopy map when faced with this kind of "hard" problem. The ordering of the constraints is quite important to the other algorithms, especially when a large number of MNL constraints are employed, as is the case here. While these constraints were randomly generated, they were randomly generated to satisfy the "hardness" of the constraint set, which meant that the first $k+1$ constraints of each list were MNL constraints all involving the same datapoint. This is exactly the kind of ordering guaranteed to give the other algorithms the most difficulty in satisfying the constraints. The homotopy algorithm, on the other hand, is unbiased by constraint ordering, and largely indifferent to constraint type.

The adjusted Rand index is a good tool for a posteriori judgment of clusters, but semisupervised clustering problems don't have the classification a priori. The tools of the researcher are (usually) limited to intercluster and intracluster distances, with limited extra information not presented as a dimension of the clustering. For this sort of situation, the homotopy method shines in the ϵ– and δ–constraint experiments (Table 9), due to the use of the K-Means approximation $\hat{K}(X)$ in the homotopy formulation. The net effect of this approximation is to cause the homotopy method to account for local minima of the K-Means approximation as λ increases. Assuming that $\Psi(\hat{X}) > 0$ and $Z(\hat{X}) < 0$, with a small ν and μ, which is almost always the case in practice, an $\hat{X}$ at some $\hat{\lambda} < 1$ that would satisfy $\nabla_X F(C, \hat{X}) \approx 0$, but violate $\nabla_X \hat{K}(\hat{X}) \approx 0$, would not lie along γ. Thus, γ contains those solutions that do not strongly violate the clustering hypothesis as arc length increases along γ , resulting in the end point at $\lambda = 1$ being generally favorable to the cluster hypothesis.

All of this is an aside to the true purpose of the development of the homotopy map. The Dunn index is a reasonable tool for measuring the validity of multiple clusterings of the same dataset, although it is one that can not handle nonconvex clusters (hence it was not used in the ϵ– and δ– experiments). Figures 6–9 show the utility of the homotopy map $\bar{\rho}$ without reference to the "correct" clustering, simulating the constraints that a researcher may reasonably discover on their own. Note that generally, the Dunn index improves over the original K-Means clustering as constraints are satisfied. Since satisfying the given set of constraints improves the quality of the discovered partitions, the improvement to the Dunn

index (or other cluster metric) can be viewed as providing a standard to judge imposed constraints. In this case, of course, the constraints are all known to be valid. In addition, these figures make it easy to show how valid constraints can guide the homotopy algorithm through regions of poor clustering to establish better partitions. For example, in Figure 6 it is easy to see that several regions of poor clustering are encountered as arc length increases along γ, but the final clustering in this case, with all constraints satisfied, happens to be the best one. Of course, this need not be the case with these problems, but it would appear to speak to the reasonable nature of these constraints.

The new homotopy approach for constrained clustering problems uses state-of-the-art mathematical software to characterize multicriteria problems in constrained clustering. Just as in other applications of homotopy methods to science and engineering, the application of homotopy methods to machine learning problems can usher in greater understanding of solution sets and the value of constraints. Besides the strong mathematical foundations and rigorous formalisms brought to classical machine learning problems, this homotopy approach has the potential to greatly reduce the ad hoc nature of methodological experimentation that is prevalent in practice. The approach given here not only helps extract better patterns from data, but also helps formally understand the internal workings of machine learning techniques. Future work includes homotopy maps for other multicriteria machine learning problems such as information bottleneck, time series segmentation, and transfer learning.

Chapter 4.

CONCLUSION AND FUTURE WORK

This thesis investigates the field of optimization in two major ways. The first part shows the application of several powerful modern optimization methods to the kinds of cutting-edge real-world optimization problems facing researchers today. The second part shows the development and theoretical underpinnings of a novel, powerful method for attempting to solve an NP-hard modern machine learning problem. While this new method is of course not perfect, it compares competitively with other cutting-edge approaches to the problem in terms of raw power to solve and, unlike the other approaches available, efficiently generates a solution trace that serves the researcher as a tradeoff curve, revealing information both about the problem at hand and the constraint set chosen.

Most of the future work beyond this thesis involves refining the homotopy map $\tilde{\rho_a}$. The map $\tilde{\rho_a}$ is unable to extend beyond convex clusterings at this time, but there's no reason this should remain the case, and some map should be able to encompass the same set of solutions available to the methods that can generate nonconvex partitions. While restricting the map to convex clusters may actually aid it in those cases where ϵ- and δ-constraints are the primary concern, this is no excuse for not developing the map further.

Other possible applications involve modifying the penalty function to allow for other optimization methods to be brough to bear, such as those discussed in the beginning of this section. The weakness of the penalty function F, that it falls apart when cluster centroids are allowed to stack, requires care, but the fact that many of these algorithms, such as DIRECT, do not require continuous problems, and others, such as QNSTOP, fare well in the face of stochastic noise, means that with some tinkering the penalty function presented here can become a powerful tool for constrained clustering without the tyranny of distance functions or absolute certainty of values.

REFERENCES

[1] K. Bache and M. Lichman. UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science. 2013.

[2] D. E. Anderson, M. L. Madigan and M. A. Nussbaum, "Maximum voluntary joint torque as a function of joint angle and angular velocity: model development and application to the lower limb", *Journal of Biomechanics*, 40, no. 14, 3105–3113, 2007.

[3] M. S. Baghshah and S. B. Shouraki, "Learning low-rank kernel matrices for constrained clustering", *Neurocomput.*, 74, no. 12–13, 2201–2211, 2011.

[4] M. F. Balcan and A. Blum, "A discriminative model for semi-supervised learning", *J. ACM.*, 57, no.3, 19:1–19:46, 2010.

[5] S. Basu, I. Davidson, and K. Wagstaff, eds, *Constrained Clustering: Advances in Algorithms, Theory, and Applications*, Chapman and Hall, Boca Raton, FL, 2008.

[6] M. Bilenko, S. Basu, and R. J. Mooney, "Integrating constraints and metric learning in semi-supervised clustering", in *ICML '04*, 11–18, 2004.

[7] K. A. Bieryla, "An investigation of perturbation-based balance training as a fall prevention intervention for older adults", *Ph.D. thesis*, Department of Mechanical Engineering, VPI & SU, Blacksburg, VA, 2009.

[8] R. H. Byrd, M. E. Hribar, and J. Nocedal, "An interior point method for large scale nonlinear programming", *SIAM Journal on Optimization*, 9, no. 4, 877–900, 1999.

[9] R. H. Byrd, J.C. Gilbert, and J. Nocedal, "A trust region method based on interior point techniques for nonlinear programming", *Mathematical Programming A*, 89, 149–185, 2000.

[10] R. H. Byrd, J. Nocedal, and R. A. Waltz, *Large-Scale Nonlinear Optimization*, Springer-Verlag, 35–59, 2006.

[11] B. S. Castle, *Quasi-Newton Methods for Stochastic Optimization and Proximity-Based Methods for Disparate Information Fusion*, PhD thesis, Indiana University, 2012.

[12] O. Chapelle, B. Scholkopf, and A. Zien, *Semi-Supervised Learning*, 1st ed., Cambridge, MA, MIT Press. 2008.

[13] K. B. Cheng, "The relationship between joint strength and standing vertical jump performance", *Journal of Applied Biomechanics*, 24, no. 3, 224–233, 2008.

[14] A. R. Conn, N. I. M. Gould, and P. L. Toint, "Trust-Region Methods", *MPS-SIAM Series on Optimization*, SIAM, Philadelphia, 2000.

[15] J.E. Dennis, Jr. and R.B. Schanbel, *Numerical methods for unconstrained optimization and nonlinear equations (2nd ed.)*, SIAM, Philadelphia, 1996.

[16] S. N. Chow, J. Mallet-Paret, and J. A. Yorke, "Finding zeros of maps: homotopy methods that are constructive with probability-one", *Math. Comput.*, 32, 887–899, 1978.

[17] A. Corduneanu and T. Jaakkola, "Continuation methods for mixing heterogeneous sources", in *UAI '02*, 111–118, 2002.

[18] B. R. Dai, C. R. Lin, and M. S. Chen, "Constrained data clustering by depth control and progressive constraint relaxation", *The VLDB Journal*, vol. 16, 201–217, 2007.

[19] I. Davidson and S. S. Ravi, "Clustering with constraints: feasibility issues and the K-Means algorithm", in *SDM '05*, 201–211, 2005.

[20] I. Davidson and S. S. Ravi, "Identifying and generating easy sets of constraints for clustering", in *AAAI '06*, 336–341, 2006.

[21] I. Davidson and S. S. Ravi, "The complexity of non-hierarchical clustering with instance and cluster level constraints", *Data Min. Knowl. Discov.*, 14, 25–61, 2007.

[22] I. Davidson, "Two approaches to understanding when constraints help clustering", in *KDD '12*, 1312–1320, 2012.

[23] A. Demiriz, K. Bennett, and P. Bradley, "Chapter 9: Using assignment constraints to avoid empty clusters in K-Means clustering." In *Constrained Clustering: Advances in Algorithms,*

Theory, and Applications, 1st ed, K. Wagstaff, I.Davidson, and S. Basu, eds. 1st ed. Chapman & Hall/CRC, Boca Raton, FL, 2008.

[24] D.R. Easterling, L.T. Watson, and M.L. Madigan, "The DIRECT algorithm applied to a problem in biomechanics with conformal mapping", *in Proc. 2010 International Conference on Scientific Computing, CSC '10*, H. Arabnia and G. Grawanis (eds.), CSREA Press, USA, 2010, 128–133, 2010.

[25] D.R. Easterling, L.T. Watson, M.L. Madigan, B.S. Castle, and M.W. Trosset, "Parallel deterministic and stochastic global minimization of functions with very many minima", *Comput. Otim. Appl.*, 57, 469–492, 2014.

[26] D. Y. Gao, *Duality Principles in Nonconvex Systems: Theory, Methods, and Applications*, Kluwer Academic Publishers, 472 pp, 2000a.

[27] D. Y. Gao, "Canonical dual transformation method and generalized triality theory in nonsmooth global optimization", *Journal of Global Optimization*, 17(1/4), 127–160, 2000b.

[28] W.L. Goffe, G.D. Ferrier, and J. Rogers, "Global optimization of statistical functions with simulated annealing", *Journal of Econometrics*, 60, 65–100, 1994.

[29] W. W. Hager, R. Rostamian, and D. Wang, "The wave annihilation technique and the design of nonreflective coatings", *SIAM Journal on Applied Mathematics*, 60, no. 4, 1388–1424, 2000.

[30] J. He; A. Verstak, L. T. Watson, and M. Sosonkina, "Design and implementation of a massively parallel version of DIRECT", *Computational Optimization and Applications*, 40, no. 2, 217–245, 2008.

[31] J. He, L.T. Watson, N. Ramakrishnan, C. A. Shaffer, A. Verstak, J. Jiang, K. Bae, and W.H. Tranter, "Dynamic data structures for a direct search algorithm", *Computational Optimization and Applications*, 23, no. 1, 5–25, 2002.

[32] J. He, A. Verstak, L.T. Watson, and M. Sosonkina, "Performance modeling and analysis of a massively parallel DIRECT: part 1", *International Journal of High Performance Computing Applications*, 23, no. 1, 14–28, 2009a.

[33] J. He, L. T. Watson, and M. Sosonkina, "Algorithm 897: VTDIRECT95: serial and parallel codes for the global optimization algorithm DIRECT", *ACM Transactions on Mathematical Software*, 36, no. 3, Article 17, 1–24, 2009b.

[34] P. He, X. Xu, and L. Chen, "Constrained clustering with local constraint propagation", in *ECCV '12*, 223–232, 2012.

[35] J. S. Higginson, R.R. Neptune, and F.C. Anderson, "Simulated parallel annealing within a neighborhood for optimization of biomechanical systems.", *Journal of Biomechanics.*, 38, no. 9, 1938–1942, 2004.

[36] M. S. Hossain, S. Tadepalli, L. T. Watson, I. Davidson, R. Helm and N. Ramakrishnan, "Unifying dependent clustering and disparate clustering for non-homogeneous data", in *KDD '10*, 593–602, 2010.

[37] M. G. Hoy, F. E. Zajac, and M. E. Gordon, "A musculoskeletal model of the human lower extremity: the effect of muscle, tendon, and moment arm on the moment-angle relationship of musculotendon actuators at the hip, knee, and ankle", *Journal of Biomechanics*, 23, no. 2, 157–169, 1990.

[38] L. Ingber, "Simulated annealing: practice versus theory", *Mathematical Computer Modeling*, 18, no. 11, 29–57, 1993.

[39] S. Ji, L. T. Watson, and L. Carin, "Semisupervised learning of hidden Markov models via a homotopy method", *IEEE Trans. Pattern Anal. Machine Intell.*, 31, 275–287, 2009.

[40] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, "Lipschitzian optimization without the Lipschitz constant", *Journal of Optimization Theory and Applications*, 79, no. 1, 157–181, 1993.

[41] D. R. Jones, "The DIRECT global optimization algorithm", *Encyclopedia of Optimization, Vol. 1*, Dordrecht : Kluwer Academic Publishers, 431–440, 2001.

[42] J. Kiefer and J. Wolfowitz, "Stochastic estimation of a regression function", *Annals of Mathematical Statistics*, 23, 462–466, 1952.

[43] S. Kirkpatrick, C. D. Gelatt, and M.P. Vecchi, "Optimization by simulated annealing", *Science, New Series*, 220, no. 4598, 671–680, 1983.

[44] G. Kreisselmeier and R. Steinhauser, "Systematic control design by optimizing a vector performance index.", *International Federation of Active Controls Symposium on Computer-Aided Design of Control Systems*, Zurich, Switzerland, 1979.

[45] P. Luo, F. Zhang, H. Xiong, Y. Xiong, and Q. He, "Transfer learning from multiple source domains via consensus regularization", in *CIKM '08*, 103–112, 2008.

[46] O. Mangasarian, "Equivalence of the complementarity problem to a system of nonlinear equations", *SIAM Journal on Applied Mathematics*, vol. 31, no. 1, 89–92, 1976.

[47] J. L. Maryak and D. C. Chin, "Global random optimization by simultaneous perturbation stochastic approximation", *IEEE Transactions on Automatic Control*, 53, no. 3, 780–783, 2008.

[48] M. J. Pavol, T. M. Owings, and M.D. Grabiner, "Body segment inertial parameter estimation for the general population of older adults", *Journal of Biomechanics*, 35, 707–712, 2002.

[49] R. Phillips, "A Probabilistic Classification Algorithm with Soft Classification Output", PhD Thesis, Virginia Tech, Blacksburg, VA, 2009.

[50] N. R. Radcliffe, D. R. Easterling, L. T. Watson, M. L. Madigan, and K. A. Bieryla, "Results of two global optimization algorithms applied to a problem in biomechanics.", *in Proc. 2010 Spring Simulation Multiconference, High Performance Computing Symp*, A. Sandu, L. Watson, and W. Thacker (eds), Soc. for Modelling and Simulation Internat., Vista, CA, 2010, 117–123, 2010.

[51] D. J. Ram; T. H. Sreenivas; K. G. Subramaniam, "Parallel simulated annealing algorithms", *Journal of Parallel and Distributed Computing*, 37, 207–212, 1996.

[52] R. Riener and T. Edrich, "Identification of passive elastic joint moments in the lower extremities", *Journal of Biomechanics*, 32, no. 5, 539–544, 1999.

[53] Y. Sato and M. Iwayama, "Interactive constrained clustering for patent document set", in *PaIR '09*, 17–20, 2009.

[54] B. W. Schulz, J. A. Ashton-Miller, and N. B. Alexander, "Can initial and additional compensatory steps be predicted in young, older, and balance-impaired older females in response to anterior and posterior waist pulls while standing?", *Journal of Biomechanics*, 39, no. 8, 1444–1453, 2006.

[55] W. S. Selbie and G. E. Caldwell, "A simulation study of vertical jumping from different starting postures", *Journal of Biomechanics*, 29, no. 9, 1137–1146, 1996.

[56] J. Sese, Y. Kurokawa, M. Monden, K. Kato, and S. Moroshita, "Constrained clusters of gene expression profiles with pathological features", *Bioinformatics*, 20, no. 17, 3137–3145, 2004.

[57] K. Sinha and M. Belkin, "The value of labeled and unlabeled examples when the model is imperfect", in *Advances in Neural Information Processing Systems 20*, 2008.

[58] J. C. Spall, "A stochastic approximation technique for generating maximum likelihood parameter estimates", *in Proc. American Control Conference (Minneapolis, MN, June 10-12)*, 1161–1167, 1987.

[59] J. C. Spall, "Multivariate stochastic approximation using simultaneous perturbation gradient approximation", *IEEE Trans. Autom. Control*, 37, no. 3, 332–341, 1992.

[60] J. C. Spall, "An overview of the simultaneous perturbation method for efficient optimization", *John Hopkins APL Tech. Digest*, 19, no. 4, 482–492, 1998a.

[61] J. C. Spall, "Implementation of the simultaneous perturbation algorithm for stochastic optimization", *IEEE Transactions on Aerospace and Electronic Systems*, 34, no. 3, 817–823, 1998b.

[62] D. M. Stablein, W. H. Carter, Jr., and G. L. Wampler, "Confidence regions for constrained optima in response-surface experiments", *Biometrics*, 39, 759–763, 1983.

[63] M. E. Taylor, G. Kuhlmann, and P. Stone, "Autonomous transfer for reinforcement learning", in *AAMAS '08*, vol.1, 283–290, 2008.
[64] X. Wang and I. Davidson, "Flexible constrained spectral clustering", in *KDD '10*, 563–572, 2010.
[65] L. T. Watson, "A globally convergent algorithm for computing fixed points of C^2 maps", *Appl. Math. Comput.*, 5, 297–311, 1979.
[66] L. T. Watson, "Solving the nonlinear complimentarity problem by a homotopy method", *SIAM J. Control Optimization*, 17, 36–46, 1979.
[67] L. T. Watson, "Numerical linear algebra aspects of globally convergent homotopy methods", *SIAM Rev.*, 28, 529–545, 1987.
[68] L. T. Watson, S. Billups, and A.P. Morgan, "Algorithm 652: HOMPACK: a suite of codes for globally convergent homotopy algorithms", *ACM Trans. Math. Software*, vol. 13, 281–310, 1987.
[69] L.T. Watson, M. Sosonkina, R. C. Melville, A. Morgan, and H. Walker, "Algorithm 777: HOMPACK90: a suite of Fortran 90 codes for globally convergent homotopy algorithms", *ACM Trans. Math. Software.*, 23, 514–549, 1997.
[70] L. T. Watson, "Theory of globally convergent probability-one homotopies for nonlinear programming", *SIAM J. Optimization*, 11, no. 3, 761–780, 2000.
[71] L. T. Watson and C. A. Baker, "A fully-distributed parallel global search algorithm", *Engineering Computations*, 18, no. 1–2, 155–169, 2001.
[72] L. T. Watson, "Probability-one homotopies in computational science", *J. Comput. Appl. Math.*, 140, 785–807, 2002.
[73] F. Yang, F. C. Anderson, and Y. C. Pai, "Predicted threshold against backward balance loss in gait", *Journal of Biomechanics*, 40, no. 4, 804–811, 2007.
[74] F. Yang, F. C. Anderson, and Y. C. Pai, "Predicted threshold against backward balance loss following a slip in gait", *Journal of Biomechanics*, 41, no. 9, 1823–1831, 2008.
[75] H. Yang and J. Callan, "A metric-based framework for automatic taxonomy induction", in *ACL '09*, 1, 271–279, 2009.
[76] Q. Yang, Y. Chen, G.R. Xue, W. Dai, and Y. Yu, "Heterogeneous transfer learning for image clustering via the social web", in *ACL '09*, 1–9, 2009.
[77] D. Zhang, J. He, Y. Liu, L. Si, and R. Lawrence, "Multi-view transfer learning with a large margin approach", in *KDD '11*, 1208–1216, 2011.

Appendix A: Biomechanics problem details.

The first problem under consideration is a two-dimensional musculoskeletal model utilizing forward dynamic simulations (Bieryla, 2009). The task investigated involves maintaining bipedal balance without stepping after an abrupt backwards displacement of the supporting surface. A penalty function $f(x)$ is derived from the 57 dimensional x derived from the torque activation level function $A(t)$ discretized to nineteen distinct nodes for three joints.

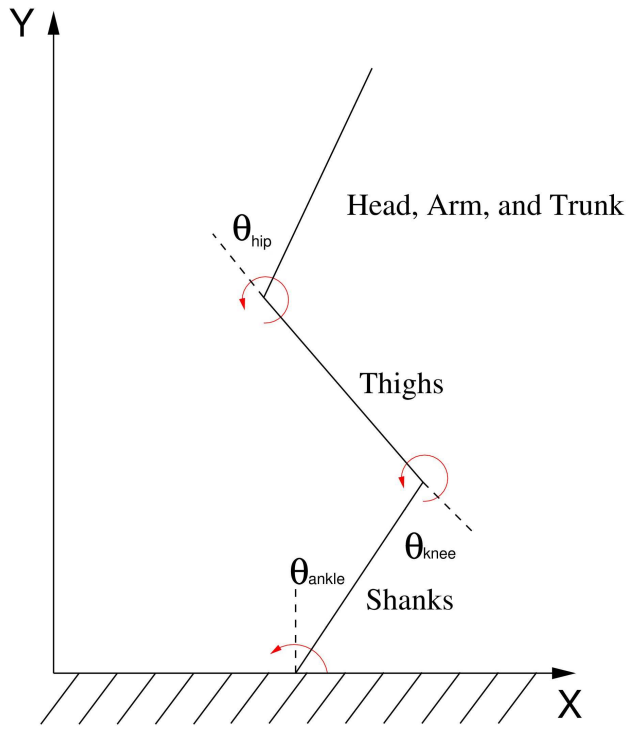

Figure A1. Schematic drawing of the three segment sagittal plane model representing the human body.

The musculoskeletal model is a sideview sagittal plane representation of the volunteer including three rigid segments representing the shanks, thighs, and head-arms-trunk (HAT) connected by frictionless pin joints (see Figure A1) activated by three joint torques representing the torques generated by muscles crossing the ankles, knees, and hips. The joint torques are the sum of the passive and active joint torques $T = T_p + T_a$ and represent all flexor and extensor contributions to the joints. The feet are neglected in the model because the volunteer from which experimental data was derived exhibited minimal heel rise during

trials. As such, the joint representing the ankle is assumed to simply connect the distal end of the shanks to the supporting surface. The inputs to the musculoskeletal model are the joint torques and the time-varying position of the ankle, which is rigidly connected to the supporting surface.

Equations of motion for the model are derived from Lagrange dynamics, and are uniquely determined from the segments' length, mass, center of mass, and moments of inertia. These constants are calculated from the subject's height (1.6 m) and weight (60 kg) by the formulas presented in (Pavol et al., 2002), and are given here for convenience. The segment masses are calculated as 5.39 kg for the shank segment, 13.0 kg for the thigh segment, and 37.3 kg for the trunk segment. The length of the segments are 0.408 m for the shank, 0.402 m for the thigh, and 0.475 m for the trunk. The center of mass for each segment, in distance from the proximal joint, is 0.171 m for the shank, 0.157 m for the thigh, and $2.5 \cdot 10^{-4}$ m for the trunk. The moments of inertia are calculated as 0.0584 $\mathrm{kg} \cdot \mathrm{m}^2$ for the shank, 0.228 $\mathrm{kg} \cdot \mathrm{m}^2$ for the thigh, and 2.07 $\mathrm{kg} \cdot \mathrm{m}^2$ for the trunk. Finally, the initial angles for the model are -0.0114 rad for the ankle, 3.152 rad for the knee, and 3.272 rad for the hip.

Passive torque ($\mathrm{N} \cdot \mathrm{m}$)

$$T_p = 2(T_{p,a},\ T_{p,k},\ -T_{p,h})^{\mathrm{T}}$$

is calculated using equations taken from (Riener & Edrich, 1999), which generate passive torque with respect to the ankle $T_{p,a}$, knee $T_{p,k}$, and hip $T_{p,h}$. Since these equations are in degrees, joint angles must be converted to degrees in order to use them. Given that $\theta_{a,\circ}$ represents the angle between the ground and the shank, $\theta_{k,\circ}$ represents the angle between the shank and the thigh, and $\theta_{h,\circ}$ represents the angle between the hip and the torso, $T_{p,a} = e^{(a - b\,\theta_{a,\circ} - c\,\theta_{k,\circ})} - e^{(-d + f\,\theta_{a,\circ} + g\,\theta_{k,\circ})} - 1.792$ represents the passive torque associated with a single ankle joint, where $a = 2.1016$, $b = 0.0843$, $c = 0.0176$, $d = 7.97634$, e is Euler's constant, $f = 0.1949$, and $g = 0.0008$. $T_{p,k} = e^{(h - j\,\theta_{a,\circ} - k\,\theta_{k,\circ} + l\,\theta_{h,\circ})} - e^{(-m + n\,\theta_{a,\circ} + o\,\theta_{k,\circ} - p\,\theta_{h,\circ})} + e^{(q - r\,\theta_{k,\circ})} - 4.820$ represents the passive torque associated with a single knee joint, where $h = 1.8$, $j = 0.0460$, $k = 0.0352$, $l = 0.0217$, $m = 3.971$, $n = 0.0004$, $o = 0.0495$, $p = 0.0128$, $q = 2.220$, and $r = 0.150$. $T_{p,h} = e^{(s - t\,\theta_{k,\circ} - u\,\theta_{h,\circ})} - e^{(v - w\,\theta_{k,\circ} + y\,\theta_{h,\circ})} - 8.072$ represents the passive torque associated with a single hip joint, where $s = 1.4655$, $t = 0.0034$, $u = 0.0750$, $v = 1.3403$, $w = 0.0226$, and $y = 0.0305$.

Active torque ($\mathrm{N} \cdot \mathrm{m}$)

$$T_a = 4(T_{a,a},\ T_{a,k},\ T_{a,h})^{\mathrm{T}}$$

is defined as the maximum isometric torque scaled by three functions that are known to influence torque production. (The value "4" is that used here and by Bieryla (Bieryla, 2009), but should be "2" in a correct model.) Each active torque (ankle $T_{a,a}$, knee $T_{a,k}$, and hip $T_{a,h}$) is the result of the torques generated by forces applied by muscles in two directions (extension and flexion). The active torque with respect to an individual joint j and a force direction f generated at time t (s) with joint angle θ_j (rad) and angular velocity ω_j (rad/s) is

$$T_{a,j,f}(t, \theta_j, \omega_j) = T_{j,f,\max}\, r_{j,f}(\theta_j)\, h_j(\omega_j)\, A_j(t).$$

Depending on the activation at a given moment in time $A_j(t)$, either the extension or flexion formulas will be used to calculate $T_{a,j,f}$. Positive activation for a joint corresponds to extension for the ankle and hip and flexion for the knee, and vice versa for negative activation.

$T_{a,a,e,\max} = 0.125hwt_s$ is the maximum isometric torque ($\mathrm{N} \cdot \mathrm{m}$) for the ankle in the extension direction, where h is the height of the subject (m), w is the weight of the subject (N), and $t_s = 1.2$ is a (unitless) variable based on the strength of the subject. Similarly, $T_{a,a,f,\max} = 0.022hwt_s$, $T_{a,k,e,\max} = 0.124hwt_s$, $T_{a,k,f,\max} = 0.060hwt_s$, $T_{a,h,e,\max} = 0.138hwt_s$, and $T_{a,h,f,\max} = 0.081hwt_s$. These maximum isometric torques are determined for a single lower extremity from a strength model of female older adults (Anderson et al., 2007).

The torque-angle relation $r_{j,f}(\theta_j)$ (unitless) is obtained from previously published experimental data (Hoy et al., 1990) and varies from zero to one. If the values for the angles fall outside the range allowed by the model during the simulation, they are set to the limit of the model. Extension or flexion formulas are used depending on the sign of the angle of the corresponding joint. The angle limits (rad) and relation formulas are

$$-0.52 < \theta_a < 0.61,\ 0.0 < \theta_k < 2.27,\ -0.17 < \theta_h < 2.27,$$

$$r_{a,f} = -0.1731(\theta_a^3) - 0.5882(\theta_a^2) + 0.3357(\theta_a) + 0.9502,$$

$$r_{a,e} = 2.742(\theta_a^4) + 1.6115(\theta_{a^3}) - 2.8579(\theta_a^2) - 0.4996(\theta_a) + 0.9699,$$

$$r_{k,f} = -0.2543(\theta_k^5) + 1.5215(\theta_k^4) - 2.9033(\theta_k^3) + 1.4916(\theta_k^2) + 0.2539(\theta_k) + 0,7643,$$

$$r_{k,e} = 0.2334(\theta_k^4) - 0.4944(\theta_k^3) - 1.0148(\theta_k^2) + 2.051(\theta_k) + 0.1865,$$

$$r_{h,f} = 0.4450(\theta_h^7) - 3.1958(\theta_h^6) + 8.5726(\theta_h^5) - 10.2750(\theta_h^4) + 4.7283(\theta_h^3) - 0.1678(\theta_h^2) + 0.2293(\theta_h) + 0.6449,$$

$$r_{h,e} = 0.2056(\theta_h^4) - 0.5625(\theta_h^3) - 0.2723(\theta_h^2) + 0.9446(\theta_h) + 0.6095.$$

The torque-angular velocity relation

$$h(\omega_j) = \begin{cases} (\omega_0 - \omega_j)/(\omega_0 + \Gamma\omega_j), & \omega_j/\omega_0 \leq 1, \\ 0, & \omega_j/\omega_0 > 1, \end{cases}$$

also varies from zero to one and is obtained from (Selbie & Caldwell, 1996), where ω_j is the angular velocity of the joint j (rad/s), ω_0 (± 20 rad/s) is the maximum angular velocity for all joints, and $\Gamma = 2.5$ is the shape factor describing the torque-angular velocity curve (Selbie & Caldwell, 1996). If the angular velocity and activation level have opposite signs, indicative of eccentric muscle contraction, $h(\omega_j)$ is increased to a maximum value of 1.5.

$A_j(t)$ (unitless) is allowed to vary from -1 to 1 to apply a flexion or extension torque at a joint. Because activation dynamics are not instantaneous, joint torque activation rate of change is limited to a maximum absolute value of 1/0.08 per second (Cheng, 2008). To enforce this rate of change, a bijective conformal mapping is employed (Easterling et al., 2010). Activation $A_j(t)$ for the ankles, knees, and hips for the entire 1.8s simulation is defined by nineteen joint activation "nodes" spaced 100ms apart and whose values vary from -1 to 1 (57 nodes total). Linear interpolation is used to define the activation levels between consecutive nodes. These nodes represent the variables for the objective function f.

The four optimization algorithms under consideration are used to attempt to determine the values for the joint activation nodes that minimize the objective function

$$\begin{aligned} f = {} & w_1 \int_{t_0}^{t_f} (X_C(t) - X_A)dt + w_2 \int_{t_0}^{t_f} e(\theta(t))dt \\ & + w_3 \int_{t_0}^{t_f} e(\dot{\theta}(t))dt + w_4 \int_{t_0}^{t_f} \left(\sum_{i=1}^{3} \dot{q}_i(t)^2 \right)^{1/2} dt \\ & + w_5 \int_{t_0}^{t_f} \dot{X}_C(t)dt + w_6 \int_{t_0}^{t_f} \ddot{X}_C(t)dt \\ & + w_7 \sum_{i=1}^{3} \int_{t_0}^{t_f} (\tau_i(t)^2)dt \end{aligned}$$

adapted from (Yang et al., 2007), (Yang et al., 2008), where $e(s(t)) = \sum_{i=1}^{3} \phi(s_i(t))$ and

$$\phi(s_i(t)) = \begin{cases} s_i(t)^- - s_i(t), & s_i(t) < s_i(t)^-, \\ 0, & s_i(t)^- \leq s_i(t) \leq s_i(t)^+, \\ s_i(t) - s_i(t)^+, & s_i(t) > s_i(t)^+, \end{cases}$$

with $s_i(t)^-$ and $s_i(t)^+$ representing the lower and upper physical bounds of the joint angles, respectively.

The first term in the objective function minimizes the maximum horizontal displacement of the center of mass $X_C(t) - X_A$, where $X_C(t)$ is the center of mass of the body on the displacement platform (m) and X_A is the position of the ankle on the displacement platform (m). These values are taken from experimental data. The second and third terms restrict joint angle $\theta(t)$ (rad) and angular velocity $\dot{\theta}(t)$ (rad/s) to remain within previously published physiologic limits. The joint angle minimums are -0.873 rad for the ankle, 0 rad for the knee and -0.524 rad for the hip. The joint angle maximums are 0.524 rad for the ankle, 2.269 rad for the knee and 2.182 rad for the hip. The angular velocity minimums are -6.2 rad/s for the ankle, -7.3 rad/s for the knee, and -8.5 rad/s for the hip. The angular velocity maximums are 8 rad/s for the ankle, 15 rad/s for the knee, and 10 rad/s for the hip. The fourth, fifth, and sixth terms minimize segment angular velocity $\dot{q}_i(t)$ (rad/s), center of mass velocity $\dot{X}_C(t)$ in (m/s), and center of mass acceleration $\ddot{X}_C(t)$ in (m/s^2), respectively, over the entire simulation. The seventh term minimizes the integral of the square of the joint torques $\tau_i(t)$, the sum of active and passive torques. The weights for f are $w_1 = 1000$, $w_2 = 500$, $w_3 = 500$, $w_4 = 50$, $w_5 = 100$, $w_6 = 25$, and $w_7 = 0.025$. The initial joint angle configuration is derived from experimental data, and the initial joint angular velocities are set to zero. The duration of the simulation time is $t_f = 1.8$ s, allowing for full recovery from the perturbation.

The minimum time to boundary (TTB) of the center of mass is used to quantify model performance with respect to balance. TTB is calculated as the instantaneous anterior-posterior (A/P) distance from the center of mass to the base of support divided by the instantaneous absolute value of the center of mass A/P velocity (Schulz et al., 2006). The base of support boundary is defined as the position of the first metatarsal based on the volunteer's anthropometry. The minimum TTB value describes the smallest amount of time for the participant to reach their limit of stability. Loss of balance occurs for TTB ≤ 0 s. A higher TTB indicates a longer period of time until the participant reaches their limit of stability. If the participant reaches the base of support a step occurs. Therefore, a decrease in TTB is seen as a degradation in balance.

In summary, each optimization algorithm attempts to minimize the stated $f(x)$, subject to the constraints that $-1 \leq x \leq 1$ and that within a single activation profile, the activation level between two consecutive discrete nodes may differ by no more than 1.25.

Appendix B: Fortran code for Homotopy map $\tilde{\rho}_a$.

```
!  The following RHO() and JAC() functions are to be called by FIXPNF() in
!    the HOMPACK90 module.  Note that these are not intended to be called
!    directly by the user.  Note also that before calling FIXPNF(),
!    CCHomotopyLoadData() should be used to set up the module.
!    CCHomotopyFinalize() should be called after calling FIXPNF() to free up
!    system resources.
!  CCHomotopyLoadData() returns a 2-element array containing u0 and v0, which
!    should be put after the 1-dimensional array of initial values "X" in a
!    1-dimensional array of size "size(X) + 2" before the call to FIXPNF().
!  All two-dimensional arrays passed to CCHomotopyLoadData() should be of the
!    form (dimension, index).
!  After calling CCFinalize(), remember to restore the returned values from
!    FIXPNF() by multiplying them by CCHomotopyDatanorm if normalization was
!    employed.
!  For more information see the CCHomotopyLoadData() and CCHomotopyFinalize()
!    headers.
SUBROUTINE RHO(A, LAMBDA, X, Y)
  !  RHO wrapper for use with FIXPNF.
  USE CC_HOMOTOPY_MOD
  IMPLICIT NONE
  REAL(KIND=R8), INTENT(IN) ::  A(:), X(:)
  REAL(KIND=R8), INTENT(IN OUT) ::  LAMBDA
  REAL(KIND=R8), INTENT(OUT) ::  Y(:)

  Y=CCHomotopyRHO(A, LAMBDA, X, size(X))
  RETURN
END SUBROUTINE RHO

SUBROUTINE RHOJAC(A,LAMBDA,X,Y,K)
  !  RHOJAC wrapper for use with FIXPNF.
  USE CC_HOMOTOPY_MOD
  IMPLICIT NONE
  REAL(KIND=R8), INTENT(IN)::  A(:),X(:)
  REAL(KIND=R8), INTENT(IN OUT)::  LAMBDA
  REAL(KIND=R8), INTENT(OUT)::  Y(:)
  INTEGER, INTENT(IN)::  K

  Y = CCHomotopyJAC(A, LAMBDA, X, K, size(X))
  RETURN
END SUBROUTINE RHOJAC

!  This file (CC_HOMOTOPY.F90) contains the module CC_HOMOTOPY_MOD that
!    declares variables and subroutines used in generating the homotopy
!    map for constrained clustering problems.    To be used with the HOMPACK90
!    homotopy tracker.

MODULE CC_HOMOTOPY_MOD
!  USE REAL_PRECISION ! Declaration of IEEE 754 arithmetic.
!         Uncomment the above line if the module is available.
  IMPLICIT NONE
!
  PRIVATE
  PUBLIC ::  CCHomotopyLoadData, CCHomotopyFinalize, CCHomotopyRHO, &
      CCHomotopyJAC, CCHomotopyDatanorm, CCHomotopyML, CCHomotopyMNL
!  If the REAL_PRECISION module is available, comment the following two lines.
  PUBLIC ::  R8
  INTEGER, PARAMETER ::  R8 = SELECTED_REAL_KIND(13)
```

```
!
!  Modify the following parameter to change the p-norm used for distance
!    measurements.    (Must be a positive and even integer.    R8 value used
!    to prevent minor inaccuracies from automatic conversion in arithmetic.)
  REAL(KIND=R8), PARAMETER ::  p = 2.0_R8
!  Module variables.
  INTEGER ::  dims !  The number of dimensions of the problem.
  INTEGER ::  numcons !  The number of problem constraints.
  INTEGER ::  numdata !  The number of datapoints.
  INTEGER ::  numprots !  The number of mean prototypes (k).
  REAL(KIND=R8) ::  bL ! Boundary value "B".
  REAL(KIND=R8) ::  CCHomotopyDatanorm !  The normalization factor.    Public
   !  so that it can be referenced after completion to restore actual prototype
   !  values.
  REAL(KIND=R8) ::  kappa !  KS kappa value.
  REAL(KIND=R8) ::  h0, h1 !  h_0 and h_1 variables.
  REAL(KIND=R8) ::  regParam !  regularization parameter for KS function.
  REAL(KIND=R8) ::  scaleFactor !  Scales all degree of belief factors.    Use in
   !  cases of overlarge numbers, which can throw off derivative calculations
   !  and homotopy curve tracking.
  REAL(KIND=R8) ::  u, v !  Mu and Nu values.
  INTEGER, DIMENSION(:,:), ALLOCATABLE ::  constraint !  Array of constraints.
   !  Constraints should be of the form (+-1, mean prototype, mean prototype).
  REAL(KIND=R8), DIMENSION(:, :), ALLOCATABLE ::  aInit !  Initial position
   !  of mean prototypes.
  REAL(KIND=R8), DIMENSION(:), ALLOCATABLE ::  degOfBelief !  Degree of Belief
   !  array.
  REAL(KIND=R8), DIMENSION(:,:), ALLOCATABLE ::  datapoint !  Array of
   !  datapoints.    Array should be of the form (dimension, index).
  REAL(KIND=R8), DIMENSION(:,:), ALLOCATABLE ::  meanProt  !  Array of mean
   !  prototypes.    Array should be of the form (dimension, index).
  REAL(KIND=R8), DIMENSION(:), ALLOCATABLE ::  xInit !  Initial x (1 dimensional).
CONTAINS
  FUNCTION CCHomotopyLoadData(datapoints, meanPrototypes, constraints, &
      rgprm, scaleExp, boundL, kappaIn, normalize, dob) RESULT(uv0)
    !  Allocates and loads the data into the module variables.
    !  Note:    2-dimensional arrays are accessed as (dimension, index).
    !    On input:
    !
    !    datapoints(1:dim, 1:d) is the array of datapoints, where "d" is the
    !        number of datapoints and "dim" is the dimension.
    !    meanPrototypes(1:dim, 1:k) is the array of mean prototypes, where "k"
    !        is the number of clusters and "dim" is the dimension.
    !    constraints(1:3, 1:c) is the array of constraints, where "c" is the
    !        number of constraints.
    !
    !    On output:
    !
    !    uv0(1) contains the initial u0 value, to be loaded into
    !          x for the tracker.
    !
    !    uv0(2) contains the initial v0 value, to be loaded into
    !          x for the tracker.
    !
    !    Optional arguments:
```

```
!
!    rgprm (if present) is the desired regParam value.
!
!    scaleExp (if present) is the scaling factor for all constraints.
!
!    boundL (if present) is the boundary value.
!
!    kappaIn (if present) is the desired kappa value.
!
!    normalize (if present) is the logical normalization flag.
!
!    dob (if present) is the array of degrees of belief for each constraint.
!        (These should all be positive weights.)
REAL(KIND=R8), DIMENSION(:, :), INTENT(IN) ::  datapoints
REAL(KIND=R8), DIMENSION(:, :), INTENT(IN) ::  meanPrototypes
INTEGER, DIMENSION(:, :), INTENT(IN) ::  constraints
REAL(KIND=R8), OPTIONAL, INTENT(IN) ::  rgprm
REAL(KIND=R8), OPTIONAL, INTENT(IN) ::  scaleExp
REAL(KIND=R8), OPTIONAL, INTENT(IN) ::  boundL
REAL(KIND=R8), OPTIONAL, INTENT(IN) ::  kappaIn
LOGICAL, OPTIONAL, INTENT(IN) ::  normalize
REAL(KIND=R8), DIMENSION(1), OPTIONAL, INTENT(IN) ::  dob
REAL(KIND=R8), DIMENSION(2) ::  uv0
!  Local variables.
INTEGER ::  i, j  !  Index variables.
INTEGER ::  ind !  Index variable for 1D -> 2D array transforms.
REAL(KIND=R8), DIMENSION(:), ALLOCATABLE ::  bounds !  Used in calculating
!    bL if not directly provided.
!  Initial data loading.
dims = size(datapoints, 1)
numprots = size(meanPrototypes, 2)
numdata = size(datapoints, 2)
numcons = size(constraints, 2)
ALLOCATE(datapoint(dims, numdata))
datapoint = datapoints
CCHomotopyDatanorm = 1.0_R8
!  Decide whether normalization will be used to prevent overflowing values
!  based on datapoint values (i.e., normalize the scale).
IF (PRESENT(normalize)) THEN
  IF (normalize) THEN
    CCHomotopyDatanorm = MAXVAL(ABS(datapoint))
  END IF
END IF
datapoint = datapoint/CCHomotopyDatanorm
ALLOCATE(meanProt(dims, numprots))
meanProt = meanPrototypes
meanProt = meanProt/CCHomotopyDatanorm
ALLOCATE(constraint(3, numcons))
constraint = constraints
!  If the user accidentally uses "0" instead of "-1" for the MNL constraints,
!    fix it here.
DO i = 1, numcons
  IF (constraint(3, i) == 0) THEN
    constraint(3, i) = -1
  END IF
```

```
END DO
ALLOCATE(aInit(dims, numprots))
aInit = meanProt
ALLOCATE(xInit(numprots * dims))
ind = 0
DO i = 1, numprots
   DO j = 1, dims
      ind = ind + 1
      xInit(ind) = aInit(j, i) !  xInit is used in calculating u0 and v0.
   END DO
END DO
IF (PRESENT(rgprm)) THEN
  regParam = rgprm
  IF (regParam < 0.0_R8) THEN
     regParam = 0.025_R8
  END IF
ELSE
  regParam = 0.025_R8 !  Utterly arbitrary.
END IF
regParam=regParam*CCHomotopyDatanorm
!  Set scale factor.
IF (PRESENT(scaleExp)) THEN
   scalefactor = scaleExp
   IF (scalefactor < 0.0_R8) THEN
      scaleFactor = 1.0_R8
   END IF
ELSE
   scaleFactor = 1.0_R8
END IF
!  Find boundary value.
ALLOCATE(bounds(dims))
IF (PRESENT(boundL)) THEN
   bL = boundL
   IF (bL < 0.0_R8) THEN
      bounds = findBounds(dims)
      bL = 2.0_R8 + MAXVAL(bounds)**2 * numprots * dims
   END IF
ELSE
   bounds = findBounds(dims)
   bL = 2.0_R8 + MAXVAL(bounds)**2 * numprots * dims
END IF
DEALLOCATE(bounds)
!  Degree-of-belief.
ALLOCATE(degOfBelief(numcons))
degOfBelief = 1.0_R8
IF (PRESENT(dob)) THEN
   degOfBelief = dob
END IF
!  Envelope function "rho" value.    (Name changed to avoid confusion.)
IF (PRESENT(kappaIn)) THEN
   kappa = kappaIn
ELSE
   kappa = 2.0_R8
END IF
h0 = 1.0_R8  !h0 and h1 can be altered here at user's desire.
h1 = 1.0_R8
```

```
  !  Initialize u, v.
  uv0(1) = calcU0()
  uv0(2) = calcV0()
END FUNCTION CCHomotopyLoadData

SUBROUTINE CCHomotopyFinalize()
  !  Deallocates all arrays.
  DEALLOCATE(datapoint, meanProt, constraint, aInit, &
      degOfBelief, xInit)
END SUBROUTINE CCHomotopyFinalize

FUNCTION norm(a, b) RESULT(outv)
  !  Finds ||datapoint(:, a) - datapoint(:, b)||_p**p.
  INTEGER, INTENT(IN) ::  a, b
  INTEGER ::  i
  REAL(KIND=R8) ::  outv

  outv = SUM((datapoint(:, a) - datapoint(:, b))**p)
END FUNCTION norm

FUNCTION normProts(a, b) RESULT(outv)
  !  Finds ||meanProt(:, a) - meanprot(:, b)||_p**p
  INTEGER, INTENT(IN) ::  a, b
  REAL(KIND=R8) ::  outv
  !  Local variables.
  INTEGER ::  i

  outv = SUM((meanProt(:, a) - meanProt(:, b))**p)
END FUNCTION normProts

FUNCTION d_normProts(a, b, dm1, dm2) RESULT(outv)
  !  Finds the first derivative of normProts with respect to
  !    meanProt(dm2, dm1).
  INTEGER, INTENT(IN) ::  a, b
  INTEGER, INTENT(IN) ::  dm1, dm2
  REAL(KIND=R8) ::  outv
  !  Local variables.
  INTEGER ::  i

  IF (a == dm1) THEN
    outv = p * (meanProt(dm2, a) - meanProt(dm2, b))**(p-1.0_R8)
  ELSE IF (b == dm1) THEN
    outv = -p * (meanProt(dm2, a) - meanProt(dm2, b))**(p-1.0_R8)
  ELSE
    outv = 0.0_R8
  END IF
END FUNCTION d_normProts

FUNCTION dd_normProts(a, b, dM11, dM12, dM21, dM22) RESULT(outv)
  !  Finds the second derivative of normProts with respect to
  !    meanProt(dM12, dM11) and meanProt(dM22, dM21).
  INTEGER, INTENT(IN) ::  a, b
  INTEGER, INTENT(IN) ::  dM11, dM12, dM21, dM22
  REAL(KIND=R8) ::  outv
  !  Local variables.
  INTEGER ::  i

  outv = 0.0_R8
  IF (a == b) THEN
    outv = 0.0_R8
  ELSE IF (a == dM11) THEN
    IF (a == dM21) THEN
```

```
      outv = p
    ELSE IF (b == dM21) THEN
      outv = -p
    END IF
  ELSE IF (b == dM11) THEN
    IF (a == dM21) THEN
      outv = -p
    ELSE IF (b == dM21) THEN
      outv = p
    END IF
  END IF
END FUNCTION dd_normProts

FUNCTION findBounds(d) RESULT(outp)
  !  In case a bounding value is not provided, findBounds calculates one based
  !    on the positions of the datapoints.    d is the dimensionality.
  INTEGER, INTENT(IN) ::  d
  REAL(KIND=R8), DIMENSION(d) ::  outp
  !  Local variables.
  INTEGER ::  i

  DO i = 1, dims
    outp(i) = MAXVAL(ABS(datapoint(i, :)))
  END DO
END FUNCTION findBounds

FUNCTION distance(meanProtInd, dataInd) RESULT(outp)
  !  Calculates the L-P norm (to the Pth power) distance between the
  !    datapoint(dataInd) and the meanProt(meanProtInd).
  INTEGER, INTENT(IN) ::  dataInd
  INTEGER, INTENT(IN) ::  meanProtInd
  REAL(KIND=R8) ::  outp
  !  Local variables.
  INTEGER ::  i

  outp = SUM((meanProt(1:dims, meanProtInd) - datapoint(1:dims, dataInd))**p)
END FUNCTION distance

FUNCTION d_distance(meanProtInd, dataInd, dM11, dM12) RESULT(outp)
  !  Calculates the first derivative of the distance function with respect
  !    to meanProt(dM12, dM11).
  INTEGER, INTENT(IN) ::  dataInd, meanProtInd, dM11, dM12
  REAL(KIND=R8) ::  outp

  outp = 0.0_R8
  IF (meanProtInd == dM11) THEN
    outp = p * &
        (meanProt(dM12, meanProtInd) - datapoint(dM12, dataInd))**(p-1.0_R8)
  END IF
END FUNCTION d_distance

FUNCTION dd_distance(meanProtInd, dataInd, dM11, dM12, &
    dM21, dM22) RESULT(outp)
  !  Calculates the second derivative of the distance function with respect
  !    to meanProt(dM12, dM11) and meanProt(dM22, dM21).
  INTEGER, INTENT(IN) ::  meanProtInd
  INTEGER, INTENT(IN) ::  dataInd
  INTEGER, INTENT(IN) ::  dM11, dM12, dM21, dM22
  REAL(KIND=R8) ::  outp

  outp = 0.0_R8
```

```
  IF (meanProtInd == dM11) THEN
    IF (meanProtInd == dM21) THEN
      IF (dM12 == dM22) THEN
        outp = p
      END IF
    END IF
  END IF
END FUNCTION dd_distance

FUNCTION closestMeanProt(a) RESULT(outp)
  !  Determines the index of the closest mean prototype to datapoint(a).
  !  Used to shortcut ML and mNL constraint calculations.
  INTEGER, INTENT(IN) ::  a
  INTEGER ::  outp
  !  Local variables.
  INTEGER ::  i, j
  REAL(KIND=R8) ::  temp, temp2

  j = 1
  temp = distance(1, a)
  DO i = 2, numprots
    temp2 = distance(i, a)
    IF (temp2 < temp) THEN
      temp = temp2
      j = i
    END IF
  END DO
  outp = j
END FUNCTION closestMeanProt

FUNCTION distdiff(m1, m2, d) RESULT(outp)
  !  D function, using mean prototypes m1 and m2 and datapoint d.
  INTEGER, INTENT(IN) ::  m1, m2, d
  REAL(KIND=R8) ::  outp

  IF (m1 == m2) THEN
    outp = 0.0_R8
  ELSE
    outp = (MAX(0.0_R8, distance(m1, d) - distance(m2, d)))**4
  END IF
END FUNCTION distdiff

FUNCTION d_distdiff(m1, m2, d, dM11, dM12) RESULT(outp)
  !  First derivative of D with respect to meanProt(dM12, dM11).
  INTEGER, INTENT(IN) ::  m1, m2, d
  INTEGER, INTENT(IN) ::  dM11, dM12
  REAL(KIND=R8) ::  outp

  outp = 0.0_R8
  IF (dM11 == m1) THEN
    outp = 4.0_R8 * MAX(0.0_R8, distance(m1, d) - distance(m2, d))**3 * &
        d_distance(m1, d, dM11, dM12)
  ELSE IF (dM11 == m2) THEN
    outp = -4.0_R8 * MAX(0.0_R8, distance(m1, d) - distance(m2, d))**3 * &
        d_distance(m2, d, dM11, dM12)
  END IF
END FUNCTION d_distdiff

FUNCTION dd_distdiff(m1, m2, d, dM11, dM12, dM21, dM22) RESULT(outp)
  !  Second derivative of D with respect to meanProt(dM12, dM11),
  !      meanProt (dM21, dM22).
```

```
  INTEGER, INTENT(IN) ::  m1, m2, d
  INTEGER, INTENT(IN) ::  dM11, dM12, dM21, dM22
  REAL(KIND=R8) ::  outp

  outp = 0.0_R8
  IF (dM11 == m1) THEN
    IF (dM21 == m1) THEN
      outp = 4.0_R8 * MAX(0.0_R8, distance(m1, d) - distance(m2, d))**3 * &
          dd_distance(m1, d, dM11, dM12, dM21, dM22) + &
          12.0_R8 * MAX(0.0_R8, distance(m1, d) - distance(m2, d))**2 * &
          d_distance(m1, d, dM11, dM12) * d_distance(m1, d, dM21, dM22)
    ELSE IF (dM21 == m2) THEN
      outp = (-12.0_R8) * MAX(0.0_R8, distance(m1, d) - &
          distance(m2, d))**2 * d_distance(m2, d, dM21, dM22) * &
          d_distance(m1, d, dM11, dM12)
    END IF
  ELSE IF (dM11 == m2) THEN
    IF (dM21 == m2) THEN
      outp = 12.0_R8 * MAX(0.0_R8, distance(m1, d) - &
          distance(m2, d))**2 * d_distance(m2, d, dM11, dM12) * &
          d_distance(m2, d, dM21, dM22) + &
          (-4.0_R8) * MAX(0.0_R8, distance(m1, d) - &
          distance(m2, d))**3 * &
          dd_distance(m2, d, dM11, dM12, dM21, dM22)
    ELSE IF (dM21 == m1) THEN
      outp = (-12.0_R8) * MAX(0.0_R8, distance(m1, d) - &
          distance(m2, d))**2 * &
          d_distance(m1, d, dM21, dM22)  * d_distance(m2, d, dM11, dM12)
    END IF
  END IF
END FUNCTION dd_distdiff

FUNCTION CCHomotopyML(consind) RESULT(outp)
  !  Must link penalty function for a given constraint.
  !  Input:    Index of constraint
  !  This function is not called in normal tracking.    Use it only
  !      to observe the size of values involved in the mL derivatives
  !      to determine if scaling is necessary.
  INTEGER, INTENT(IN) ::  consind
  REAL(KIND=R8) ::  outp
  !  Local variables.
  REAL(KIND=R8) ::  temp
  INTEGER ::  i, c, a, b

  a = constraint(1, consind)
  b = constraint(2, consind)
  outp = 0.0_R8
  IF (closestMeanProt(a) .NE. closestMeanProt(b)) THEN
    outp = 1.0_R8
    DO c = 1, numprots
      temp = 0.0_R8
      DO i = 1, numprots
        IF (i .NE. c) THEN
          temp = temp + distdiff(c, i, a) + distdiff(c, i, b)
        END IF
      END DO
      outp = outp * temp
    END DO
```

```
  END IF
END FUNCTION CCHomotopyML

FUNCTION mLTest(consind) RESULT(outp)
  !  Must link constraint test.
  !  Input:    Index of constraint.
  !  Output:    TRUE if the constraint is VIOLATED.
  !  Note:    Like all constraint functions, this one grows
  !    with the number of mean prototypes, independent of
  !    the number of data points.
  INTEGER, INTENT(IN) ::  consind
  LOGICAL ::  outp
  !  Local variables.
  REAL(KIND=R8) ::  temp
  INTEGER ::  i, c, a, b

  a = constraint(1, consind)
  b = constraint(2, consind)
  outp = .FALSE.
  IF (closestMeanProt(a) .NE. closestMeanProt(b)) THEN
     outp = .TRUE.
  END IF
END FUNCTION mLTest

FUNCTION mLpart(c, i, a, b) RESULT(outp)
  !    Part of the mL constraint (used in calculating derivative).
  !    C and I are mean prot indices.
  !    a and b are datapoint indices.
  INTEGER, INTENT(IN) ::  c, i, a, b
  REAL(KIND=R8) ::  outp

  outp = distdiff(c, i, a) + distdiff(c, i, b)
END FUNCTION mLpart

FUNCTION d_mLpart(c, i, a, b, dM1, dM2) RESULT(outp)
  !  Derivative of mLpart with respect to meanProt(dM1, dM2).
  INTEGER, INTENT(IN) ::  c, i, a, b
  INTEGER, INTENT(IN) ::  dM1, dM2
  REAL(KIND=R8) ::  outp

  outp = d_distdiff(c, i, a, dM1, dM2) + d_distdiff(c, i, b, dM1, dM2)
END FUNCTION d_mLpart

FUNCTION dd_mLpart(c, i, a, b, dM11, dM12, dM21, dM22) RESULT(outp)
  !    Second derivative of mLpart, etc.
  INTEGER, INTENT(IN) ::  c, i, a, b
  INTEGER, INTENT(IN) ::  dM11, dM12, dM21, dM22
  REAL(KIND=R8) ::  outp

  outp = dd_distdiff(c, i, a, dM11, dM12, dM21, dM22) + &
      dd_distdiff(c, i, b, dM11, dM12, dM21, dM22)
END FUNCTION dd_mLpart

FUNCTION mLsum(c, a, b) RESULT(outp)
  !  Summation function within the ML constraint penalty.
  !  c is the current "considered" mean prot index.
  !  a and b are datapoints.
  INTEGER, INTENT(IN) ::  c, a, b
  REAL(KIND=R8) ::  outp
  !  Local variables.
  INTEGER ::  i

  outp = 0.0_R8
```

```
  DO i = 1, numprots
    IF (i .NE. c) THEN
      outp = outp + mLpart(c, i, a, b)
    END IF
  END DO
END FUNCTION mLsum

FUNCTION d_mLsum(c, a, b, dM1, dM2) RESULT(outp)
  !  Derivative of the summation function within the mL constraint penalty.
  INTEGER, INTENT(IN) ::  c, a, b
  INTEGER, INTENT(IN) ::  dM1, dM2
  REAL(KIND=R8) ::  outp
  !  Local variables.
  INTEGER ::  i

  outp = 0.0_R8
  DO i = 1, numprots
    IF (i .NE. c) THEN
      outp = outp + d_mLpart(c, i, a, b, dM1, dM2)
    END IF
  END DO
END FUNCTION d_mLsum

FUNCTION dd_mLsum(c, a, b, dM11, dM12, dM21, dM22) RESULT(outp)
  !  Second derivative of the summation function within the
  !    mL constraint penalty.
  INTEGER, INTENT(IN) ::  c, a, b
  INTEGER, INTENT(IN) ::  dM11, dM12, dM21, dM22
  REAL(KIND=R8) ::  outp
  !  Local variables.
  INTEGER ::  i

  outp = 0.0_R8
  DO i = 1, numprots
    IF (i .NE. c) THEN
      outp = outp + dd_mLpart(c, i, a, b, dM11, dM12, dM21, dM22)
    END IF
  END DO
END FUNCTION dd_mLsum

FUNCTION d_mL(consind, dM1, dM2) RESULT(outp)
  !  First derivative of must link constraint(consind) with respect to
  !       meanProt(dM1, dM2).
  INTEGER, INTENT(IN) ::  consind
  INTEGER, INTENT(IN) ::  dM1, dM2
  REAL(KIND=R8) ::  outp
  !  Local variables.
  LOGICAL ::  temp
  INTEGER ::  a, b, c, i, j
  REAL(KIND=R8) ::  z, prod, summ
  REAL(KIND=R8), DIMENSION(numprots) ::  f, df

  outp = 0.0_R8
  a = constraint(1, consind)
  b = constraint(2, consind)
  DO c = 1, numprots
    !  Populate f and df with the function inside the product and
    !    its derivative, respectively.
    f(c) = mLsum(c, a, b)
    df(c) = d_mLsum(c, a, b, dM1, dM2)
```

```
   END DO
   summ = 0.0_R8
   DO c = 1, numprots
      prod = 1.0_R8
      DO i = 1, numprots
         IF (i == c) THEN
            z = df(i)
         ELSE
            z = f(i)
         END IF
         IF (ABS(z) < (summ * EPSILON(1.0_R8))/prod) THEN
            !  Basically, z = 0, so this term is useless.
            GOTO 201
         ELSE
            prod = prod * z
         END IF
      END DO
      summ = summ + prod
201    CONTINUE
   END DO
   outp = summ
  END FUNCTION d_mL

  FUNCTION dd_mL(consind, dM11, dM12, dM21, dM22) RESULT(outp)
   !  Second derivative of the mL constraint(consind), w.r.t.
   !    meanProt(dM12, dM11) and meanProt(dM22, dM21).
   INTEGER, INTENT(IN) ::  consind
   INTEGER, INTENT(IN) ::  dM11, dM12, dM21, dM22
   REAL(KIND=R8) ::  outp
   !  Local variables.
   INTEGER ::  i, j, a, b, c
   LOGICAL ::  temp
   REAL(KIND=R8) ::  summ, prod, z
   REAL(KIND=R8), DIMENSION(numprots) ::  f, df1, df2, ddf

   outp = 0.0_R8
   temp = mLTest(consind)
   IF (temp) THEN
      a = constraint(1, consind)
      b = constraint(2, consind)
      DO c = 1, numprots
         f(c) = mLsum(c, a, b)
         df1(c) = d_mLsum(c, a, b, dM11, dM12)
         df2(c) = d_mLsum(c, a, b, dM21, dM22)
         ddf(c) = dd_mLsum(c, a, b, dM11, dM12, dM21, dM22)
      END DO
      summ = 0.0_R8
      DO c = 1, numprots
         DO j = 1, numprots
            prod = 1.0_R8
            DO i = 1, numprots
               IF ((i == j) .AND. (j == c)) THEN
                  z = ddf(i)
               ELSE IF (i == c) THEN
                  z = df1(i)
               ELSE IF (i == j) THEN
                  z = df2(i)
```

```
            ELSE
              z = f(i)
            END IF
            IF (ABS(z) > (EPSILON(1.0_R8) * summ)/prod) THEN
              prod = prod * z
            ELSE
              GOTO 101
            END IF
          END DO
          summ = summ + prod
101       CONTINUE
        END DO
      END DO
      outp = summ
    END IF
  END FUNCTION dd_mL

  FUNCTION CCHomotopyMNL(consind) RESULT(outp)
    !    Must not link constraint(consind).
    !    Not used in normal tracking.    Use to determine penalty
    !        function size for scale computations.
    INTEGER, INTENT(IN) ::  consind
    REAL(KIND=R8) ::  outp
    !  Local variables.
    INTEGER ::  a, b, c, i
    REAL(KIND=R8) ::  temp

    a = constraint(1, consind)
    b = constraint(2, consind)
    outp = 0.0_R8
    IF (closestMeanProt(a) == closestMeanProt(b)) THEN
      DO c = 1, numprots
        temp = 1.0_R8
        DO i = 1, numprots
          IF (i .NE. c) THEN
            temp = temp * distdiff(i, c, a) * distdiff(i, c, b)
          END IF
        END DO
        outp = outp + temp
      END DO
    END IF
  END FUNCTION CCHomotopyMNL

  FUNCTION mNLTest(consind) RESULT(outp)
    !Must-not-link constraint(consind).
    !  Input:    Index of constraint.
    !  Output:    TRUE if the constraint is VIOLATED.
    INTEGER, INTENT(IN) ::  consind
    LOGICAL ::  outp
    !  Local variables.
    REAL(KIND=R8) ::  temp
    INTEGER ::  a, b

    a = constraint(1, consind)
    b = constraint(2, consind)
    outp = .FALSE.
    IF (closestMeanProt(a) == closestMeanProt(b)) THEN
      outp = .TRUE.
```

```
  END IF
END FUNCTION mNLTest

FUNCTION mNLpart(i, c, a, b) RESULT(outp)
  !    Part of the mNL constraint (used in calculating derivative).
  !    c and i are mean prot indices.
  !    a and b are data point indices.
  INTEGER, INTENT(IN) ::  i, c, a, b
  REAL(KIND=R8) ::  outp

  outp = distdiff(c, i, a) * distdiff(c, i, b)
END FUNCTION mNLpart

FUNCTION d_mNLpart(i, c, a, b, dM1, dM2) RESULT(outp)
  !  Derivative of mNLpart with respect to meanProt(dM2, dM1).
  INTEGER, INTENT(IN) ::  i, c, a, b
  INTEGER, INTENT(IN) ::  dM1, dM2
  REAL(KIND=R8) ::  outp

  outp = d_distdiff(c, i, a, dM1, dM2) * distdiff(c, i, b) &
      + distdiff(c, i, a) * d_distdiff(c, i, b, dM1, dM2)
END FUNCTION d_mNLpart

FUNCTION dd_mNLpart(i, c, a, b, dM11, dM12, dM21, dM22) RESULT(outp)
  !    Second derivative of mNLpart, etc.
  INTEGER, INTENT(IN) ::  i, c, a, b
  INTEGER, INTENT(IN) ::  dM11, dM12, dM21, dM22
  REAL(KIND=R8) ::  outp

  outp = dd_distdiff(c, i, a, dM11, dM12, dM21, dM22) * distdiff(c, i, b) + &
      d_distdiff(c, i, a, dM11, dM12) * d_distdiff(c, i, b, dM21, dM22) + &
      d_distdiff(c, i, a, dM21, dM22) * d_distdiff(c, i, b, dM11, dM12) + &
      distdiff(c, i, a) * dd_distdiff(c, i, b, dM11, dM12, dM21, dM22)
END FUNCTION dd_mNLpart

FUNCTION mNLprod(c, a, b) RESULT(outp)
  !  Product function within the mNL constraint penalty.
  !  c is the current "considered" mean prot index.
  !  a and b are data points.
  INTEGER, INTENT(IN) ::  c, a, b
  REAL(KIND=R8) ::  outp
  !  Local variables.
  INTEGER ::  i

  outp = 1.0_R8
  DO i = 1, numprots
    IF (i .NE. c) THEN
      outp = outp * mLpart(c, i, a, b)
    END IF
  END DO
END FUNCTION mNLprod

FUNCTION d_mNLprod(c, a, b, dM1, dM2) RESULT(outp)
  !  Derivative of the product function within the mNL constraint penalty.
  INTEGER, INTENT(IN) ::  c, a, b
  INTEGER, INTENT(IN) ::  dM1, dM2
  REAL(KIND=R8) ::  outp
  !  Local variables.
  INTEGER ::  i, j
  REAL(KIND=R8) ::  prod, summ, z
  REAL(KIND=R8), DIMENSION(:), ALLOCATABLE ::  f, df

  outp = 0.0_R8
```

```
   ALLOCATE(f(numprots))
   ALLOCATE(df(numprots))
   f = 0.0_R8
   df = 0.0_R8
   DO i = 1, numprots
      IF (i .NE. c) THEN
         f(i) = mNLpart(c, i, a, b)
         df(i) = d_mNLpart(c, i, a, b, dM1, dM2)
      END IF
   END DO
   summ = 0.0_R8
   DO j = 1, numprots
      IF (j == c) THEN
         GOTO 103
      END IF
      prod = 1.0_R8
      DO i = 1, numprots
         IF (i == c) THEN
            GOTO 102
         END IF
         z = 1.0_R8
         IF (i == j) THEN
            z = df(i)
         ELSE
            z = f(i)
         END IF
         IF (ABS(z) > (EPSILON(1.0_R8) * summ)/prod) THEN
            prod = prod * z
         ELSE
            GOTO 103  !z = 0.0_R8; no increase to summ.
         END IF
102       CONTINUE !Advance I loop without update.
      END DO
      summ = summ + prod
103    CONTINUE  !Advance J loop without update.
   END DO
   outp = summ
   DEALLOCATE(f, df)
 END FUNCTION d_mNLprod

 FUNCTION dd_mNLprod(c, a, b, dM11, dM12, dM21, dM22) RESULT(outp)
   !  Second derivative of the product function within the
   !  mNL constraint penalty.
   INTEGER, INTENT(IN) ::  c, a, b
   INTEGER, INTENT(IN) ::  dM11, dM12, dM21, dM22
   REAL(KIND=R8) ::  outp
   !  Local variables.
   INTEGER ::  i, j, k
   REAL(KIND=R8) ::  z, summ, prod
   REAL(KIND=R8), DIMENSION(numprots) ::  f, df1, df2, ddf

   DO i = 1, numprots
      IF (i == c) THEN
         f(i) = 0.0_R8
         df1(i) = 0.0_R8
         df2(i) = 0.0_R8
         ddf(i) = 0.0_R8
```

```
      ELSE
        f(i) = mNLpart(c, i, a, b)
        df1(i) = d_mNLpart(c, i, a, b, dM11, dM12)
        df2(i) = d_mNLpart(c, i, a, b, dM21, dM22)
        ddf(i) = dd_mNLpart(c, i, a, b, dM11, dM12, dM21, dM22)
      END IF
    END DO
    summ = 0.0_R8
    DO k = 1, numprots
      IF (k == c) THEN
        GOTO 106  !  Advance k past c.
      END IF
      DO j = 1, numprots
        IF (j == c) THEN
          GOTO 105  !  Advance j past c.
        END IF
        prod = 1.0_R8
        DO i = 1, numprots
          IF (i == c) THEN
            GOTO 104  !  Advance i past c.
          END IF
          IF (i == j) THEN
            IF (i == k) THEN
              z = ddf(i)
            ELSE
              z = df1(i)
            END IF
          ELSE IF (i == k) THEN
            z = df2(i)
          ELSE
            z = f(i)
          END IF
          IF (ABS(z) > (EPSILON(1.0_R8)*summ)/prod) THEN
            prod = prod * z
          ELSE
            GOTO 105
          END IF
104       CONTINUE  ! End of I loop.
        END DO
        summ = summ + prod
105     CONTINUE  ! End of J loop.
      END DO
106   CONTINUE  ! End of K loop.
    END DO
    outp = summ
  END FUNCTION dd_mNLprod

  FUNCTION d_mNL(consind, dM1, dM2) RESULT(outp)
    !  First derivative of must not link constraint(consind)
    !    w.r.t.  meanProt(dM1, dM2).
    INTEGER, INTENT(IN) ::  consind
    INTEGER, INTENT(IN) ::  dM1, dM2
    REAL(KIND=R8) ::  outp
    !  Local variables.
    LOGICAL ::  temp
    INTEGER ::  c, i, j, a, b
```

```
  REAL(KIND=R8) ::  z, prod, summ
  REAL(KIND=R8), DIMENSION(numprots) ::  f, df

  outp = 0.0_R8
  a = constraint(1, consind)
  b = constraint(2, consind)
  DO c = 1, numprots
    outp = outp + d_mNLprod(c, a, b, dM1, dM2)
  END DO
END FUNCTION d_mNL

FUNCTION dd_mNL(consind, dM11, dM12, dM21, dM22) RESULT(outp)
  !  Second derivative of the mL constraint.
  INTEGER, INTENT(IN) ::  consind
  INTEGER, INTENT(IN) ::  dM11, dM12, dM21, dM22
  REAL(KIND=R8) ::  outp
  !  Local variables.
  INTEGER ::  i, j, a, b, c
  LOGICAL ::  temp
  REAL(KIND=R8) ::  summ, prod1, prod2
  REAL(KIND=R8), DIMENSION(numprots) ::  f, df1, df2, ddf

  outp = 0.0_R8
  temp = mNLTest(consind)
  IF (temp) THEN
    a = constraint(1, consind)
    b = constraint(2, consind)
    DO c = 1, numprots
      outp = outp + dd_mNLprod(c, a, b, dM11, dM12, dM21, dM22)
    END DO
  END IF
END FUNCTION dd_mNL

FUNCTION d_Constraint(consind, dM1, dM2) RESULT(outp)
  !  Returns the derivative of constraint(consind)
  !    w.r.t.  meanProt(dM1, dM2).
  INTEGER, INTENT(IN) ::  consind, dM1, dM2
  REAL(KIND=R8) ::  outp

  outp = 0.0_R8
    IF (constraint(3, consind) < 0) THEN ! Must Not Link Constraint.
      IF (mNLTest(consind)) THEN
        outp = d_mNL(consind, dM1, dM2)
      END IF
    ELSE IF (constraint(3, consind) > 0) THEN ! Must Link Constraint.
      IF (mLTest(consind)) THEN
        outp = d_mL(consind, dM1, dM2)
      END IF
    ELSE
      STOP "Error:    Constraint type unidentified."
    END IF
END FUNCTION d_Constraint

FUNCTION dd_Constraint(consind, dM11, dM12, dM21, dM22) RESULT(outp)
  !  Returns the second derivative of constraint(consind)
  !    w.r.t.  meanProt(dM12, dM11), meanProt(dM22, dM21).
  INTEGER, INTENT(IN) ::  consind
  INTEGER, INTENT(IN) ::  dM11, dM12, dM21, dM22
  REAL(KIND=R8) ::  outp

  outp = 0.0_R8
```

```
  IF (constraint(3, consind) < 0) THEN ! Must Not Link Constraint.
    IF (mNLTest(consind)) THEN
      outp = dd_mNL(consind, dM11, dM12, dM21, dM22)
    END IF
  ELSE IF (constraint(3, consind) > 0) THEN ! Must Link Constraint.
    IF (mLTest(consind)) THEN
      outp = dd_mL(consind, dM11, dM12, dM21, dM22)
    END IF
  ELSE
    STOP "Error:    Constraint type unidentIFied."
  END IF
END FUNCTION dd_Constraint

FUNCTION psi(x) RESULT(outp)
  !  Bounding function.    Sum of all components squared must be less than bL.
  REAL(KIND=R8), DIMENSION(:), INTENT(IN) ::  x
  REAL(KIND=R8) ::  outp
  !  Local Variables.
  INTEGER ::  i, j, ind

  outp = 0.0_R8
  ind = 0
  DO i = 1, numprots
    DO j = 1, dims
      ind = ind + 1
      outp = outp + x(ind)**2
    END DO
  END DO
  outp = bL - outp
END FUNCTION psi

FUNCTION d_psi(dM11, dM12)  RESULT(outp)
  !  Derivative of psi w.r.t.  meanProt(dM12, dM11).
  INTEGER, INTENT(IN) ::  dM11, dM12
  REAL(KIND=R8) ::  outp

  outp = -2.0_R8 * meanProt(dM12, dM11)
END FUNCTION d_psi

FUNCTION dd_psi(dM11, dM12, dM21, dM22) RESULT(outp)
  !  Second derivative of psi w.r.t.  meanProt(dM12, dM11) and
  !    meanProt(dM22, dM21).
  INTEGER, INTENT(IN) ::  dM11, dM12, dM21, dM22
  REAL(KIND=R8) ::  outp

  outp = 0.0_R8
  IF (dM11 == dM21) THEN
    IF(dM12 == dM22) THEN
      outp = -2.0_R8
    END IF
  END IF
END FUNCTION dd_psi

FUNCTION calcU0() RESULT(outp)
  !  Calculates u0 based on initial x and h0.
  !  Uses basic binary search.
  REAL(KIND=R8) ::  outp
  !  Local variables.
  REAL(KIND=R8) ::  a, b, t, uk, npsix, res

  npsix = psi(xInit)
```

```
  IF (npsix < 1.0_R8) THEN
    write(*,*) "Error:    Boundary value too small for binsearch"
    STOP
  END IF
  a = 0.0_R8
  b = 2*npsix
  res = 1.0_R8
  DO WHILE (ABS(res) > EPSILON(1.0_R8))
    uk = (a+b)/2
    t = -1.0_R8 * ABS(npsix - uk)**3 + npsix**3 + uk**3 - h0
    IF (t > 0.0_R8) THEN
      b = uk
      res = b - a
    ELSE IF (t < 0.0_R8) THEN
      a = uk
      res = b - a
    ELSE
      outp = uk
      res = 0.0_R8
    END IF
  END DO
  outp = uk
END FUNCTION calcU0

FUNCTION calcV0() RESULT(outp)
  !  Calculates v0 based on initial x and h1.  Uses basic binary search.
  REAL(KIND=R8) ::  outp
  !  Local variables.
  REAL(KIND=R8) ::  a, b, t, vk, npsix, res

  npsix = -KS()
  IF (abs(npsix) < EPSILON(1.0_R8)) THEN
    write(*,*) "Error:    KS magnitude too small for binsearch."
    STOP
  END IF
  a = 0.0_R8
  b = 2*npsix
  res = 1.0_R8
  DO WHILE (ABS(res) > EPSILON(1.0_R8))
    vk = (a+b)/2
    t = -1.0_R8 * ABS(npsix - vk)**3 + npsix**3 + vk**3 - h1
    IF (t > 0.0_R8) THEN
      b = vk
      res = b - a
    ELSE IF (t < 0.0_R8) THEN
      a = vk
      res = b - a
    ELSE
      outp = vk
      res = 0.0_R8
    END IF
  END DO
  outp = vk
END FUNCTION calcV0

FUNCTION phi(x, lambda) RESULT(outp)
  !  Phi funciton (using mu).
  REAL(KIND=R8), INTENT(IN), DIMENSION(:)  ::  x
```

```
  REAL(KIND=R8), INTENT(IN) ::  lambda
  REAL(KIND=R8) ::  outp
  !  Local variables.
  REAL(KIND=R8) ::  psix

  psix = psi(x)
  outp = -1.0_R8 * (ABS(psix - u))**3 + (psix)**3 + u**3 - &
      (1.0_R8 - lambda) * h0
END FUNCTION phi

FUNCTION d_uPhi(x, lambda) RESULT(outp)
  !  Derivative of Phi with respect to mu.
  REAL(KIND=R8), INTENT(IN), DIMENSION(:)  ::  x
  REAL(KIND=R8), INTENT(IN) ::  lambda
  !  Local variables.
  REAL(KIND=R8) ::  outp
  REAL(KIND=R8) ::  psix

  psix = psi(x)
  IF (psix - u > 0) THEN
    outp = 3.0_R8 * (psix - u)**2 + 3*u**2
  ELSE
    outp = -3.0_R8 * (psix - u)**2 + 3*u**2
  END IF
END FUNCTION d_uPhi

FUNCTION d_phi(dM11, dM12, x) RESULT(outp)
  !  Derivative of phi with respect to meanProt(dM12, dM11)
  INTEGER, INTENT(IN) ::  dM11, dM12
  REAL(KIND=R8), DIMENSION(:)  ::  x
  REAL(KIND=R8) ::  outp
  !  Local variables.
  REAL(KIND=R8) ::  psix, dpsix

  outp = 0;
  psix = psi(x)
  dpsix = d_psi(dM11, dM12)
  IF (psix > u) THEN
    outp = -3.0_R8 * (psix - u)**2 * dpsix + 3.0_R8*(psix)**2 * dpsix
  ELSE
    outp = 3.0_R8 * (psix - u)**2 * dpsix + 3.0_R8*(psix)**2 * dpsix
  END IF
END FUNCTION d_phi

FUNCTION d_kappr(dM11, dM12) RESULT(outp)
  !  Derivative of the K approximation function w.r.t.  meanProt(dm2, dm1)
  !  Note:    Since we minimize, we can skip the "k" in the numerator.
  INTEGER, INTENT(IN) ::  dM11, dM12
  REAL(KIND=R8) ::  outp
  !  Local variables.
  INTEGER ::  i, j
  REAL(KIND=R8) ::  denominator
  REAL(KIND=R8) ::  numerator
  REAL(KIND=R8) ::  term1
  REAL(KIND=R8) ::  total

  total = 0.0_R8
  DO i = 1, numdata
    total = 0.0_R8
    numerator = 2 * meanProt(dM12, dM11) - datapoint(dM12, i)
    denominator = distance(dM11, i)**2
```

```
      DO j = 1, numprots
        term1 = distance(j, i)
        IF (term1 < EPSILON(1.0_R8)) THEN ! Distance is 0.
          !  If term1 == 0, then the contribution to kappr for this datapoint
          !    is 0.
          GOTO 107
        END IF
        total = total + 1.0_R8/term1
      END DO
      denominator = denominator * total * total
      outp = outp + numerator/denominator
107   CONTINUE
    END DO
  END FUNCTION d_kappr

  FUNCTION dd_kappr(dM11, dM12, dM21, dM22) RESULT(outp)
    !  Second derivative of kappr w.r.t.  meanProt(dM12, dM11) and
    !    meanProt(dM22, dM21).
    INTEGER, INTENT(IN) ::  dM11, dM12, dM21, dM22
    REAL(KIND=R8) ::  outp
    !  Local variables.
    INTEGER ::  i, j
    REAL(KIND=R8) ::  first, second, d_first, d_second
    REAL(KIND=R8) ::  hi, lo, d_hi, d_lo
    REAL(KIND=R8) ::  term1
    REAL(KIND=R8) ::  total

    outp = 0.0_R8
    DO i = 1, numdata
      hi = 2 * meanProt(dM12, dM11) - datapoint(dM12, i)
      first =  distance(i, dM11)**2
      d_first = 2.0_R8 * distance(dM11, i) * d_distance(dM11, i, dM21, dM22)
      second = 0.0_R8
      d_second = 0.0_R8
      DO j = 1, numprots
        term1 = distance(j, i)
        IF (term1 < EPSILON(1.0_R8)) THEN ! Distance is 0.
          !  If term1 == 0 for any mean prototype, then the
          !    contribution of this entire datapoint is 0.
          GOTO 108
        END IF
        second = second + 1.0_R8/term1
        d_second = d_second - d_distance(j, i, dM21, dM22)/(term1 * term1)
      END DO
      !  Interior to the squared is now complete for both second and d_second.
      d_second = 2.0_R8 * second * d_second !  Chain rule!
      second = second * second
      lo = first * second
      d_lo = first * d_second + second * d_first
      IF ((dM11 == dM21) .AND. (dM12 == dM22)) THEN
        d_hi = 2.0_R8
      ELSE
        d_hi = 0.0_R8
      END IF
      outp = outp + (lo * d_hi - hi * d_lo)/(lo * lo)
108   CONTINUE
    END DO
```

```
END FUNCTION dd_kappr

FUNCTION KS() RESULT(outp)
  !  Kreisselmeier-Steinhauser Envelope Function (<0 version).
  REAL(KIND=R8) ::  outp
  !  Local variables.
  INTEGER ::  i, j
  REAL(KIND=R8) ::  summ

  summ = 0.0_R8
  DO i = 1, (numprots - 1)
    DO j = (i + 1), numprots
      summ = summ + exp(-1.0_R8 * kappa * (regParam - normProts(i, j)))
    END DO
  END DO
  outp = log(summ)/kappa
END FUNCTION KS

FUNCTION d_KS(dM1, dM2) RESULT(outp)
  !  Kreisselmeier-Steinhauser Envelope Function derivative.
  INTEGER, INTENT(IN) ::  dM1, dM2
  REAL(KIND=R8) ::  outp
  !  Local variables.
  INTEGER ::  i, j
  REAL(KIND=R8) ::  numerator, denominator, e_store

  numerator = 0.0_R8
  denominator = 0.0_R8
  DO i = 1, (numprots - 1)
    DO j = (i + 1), numprots
      e_store = exp(-1.0_R8 * kappa * (regParam - normProts(i, j)))
      denominator = denominator + e_store
      IF ((dM1 == i) .OR. (dM1 == j)) THEN
        numerator = numerator + d_normProts(i, j, dM1, dM2) * e_store
      END IF
    END DO
  END DO
  outp = -numerator/denominator
END FUNCTION d_KS

FUNCTION dd_KS(dM11, dM12, dM21, dM22) RESULT(outp)
  !  Kreisselmeier-Steinhauser Envelope Function derivative.
  INTEGER, INTENT(IN) ::  dM11, dM12, dM21, dM22
  REAL(KIND=R8) ::  outp
  !  Local variables.
  INTEGER ::  i, j
  REAL(KIND=R8) ::  a, da, dba, dbda, e_store

  a = 0.0_R8
  da = 0.0_R8
  dba = 0.0_R8
  dbda = 0.0_R8
  DO i = 1, (numprots - 1)
    DO j = (i + 1), numprots
      e_store = exp(-1.0_R8 * kappa * (regParam - normProts(i, j)))
      a = a + e_store
      IF ((dM11 == i) .OR. (dM11 == j)) THEN
        da = da + d_normProts(i, j, dM11, dM12) * e_store
        IF ((dM21 == i) .OR. (dM22 == j)) THEN
          dbda = dbda + dd_normProts(i, j, dM11, dM12, dM21, dM22) *&
```

```
              (-kappa)
          END IF
        END IF
        IF ((dM21 == i) .OR. (dM22 == j)) THEN
          dba = dba + d_normProts(i, j, dM21, dM22) * e_store * (-kappa)
        END IF
      END DO
    END DO
    outp = -(da*dba - a*dbda)/(a*a)
END FUNCTION dd_KS

FUNCTION phi2(lambda) RESULT(outp)
  !  -|psi(x) - u|**3 + (psi(x))**3 + u**3 - (1 - lambda)*h0.
  REAL(KIND=R8), INTENT(IN) ::  lambda
  REAL(KIND=R8) ::  outp
  !  Local Variables.
  REAL(KIND=R8) ::  psix

  psix = -KS()
  outp = -1.0_R8 * (ABS(psix - v))**3 + (psix)**3 + v**3 - &
       (1.0_R8 - lambda) * h1
END FUNCTION phi2

FUNCTION d_nuPhi2(x, lambda) RESULT(outp)
  !  Derivative of second phi w.r.t.  nu.
  REAL(KIND=R8), INTENT(IN), DIMENSION(:)  ::  x
  REAL(KIND=R8), INTENT(IN) ::  lambda
  REAL(KIND=R8) ::  outp
  !  Local variables.
  REAL(KIND=R8) ::  psix

  psix = -KS()
  IF (psix - v > 0) THEN
    outp = 3.0_R8 * (psix - v)**2 + 3*v**2
  ELSE
    outp = -3.0_R8 * (psix - v)**2 + 3*v**2
  END IF
END FUNCTION d_nuPhi2

FUNCTION d_phi2(dM11, dM12, x) RESULT(outp)
  !  Derivative of second phi w.r.t.  meanprot(dM12, dM11).
  INTEGER, INTENT(IN) ::  dM11, dM12
  REAL(KIND=R8), DIMENSION(:)  ::  x
  REAL(KIND=R8) ::  outp
  !  Local variables.
  REAL(KIND=R8) ::  psix, dpsix

  psix = -KS()
  dpsix = -d_KS(dM11, dM12)
  IF (psix > v) THEN
    outp = -3.0_R8 * (psix - v)**2 * dpsix + 3.0_R8*(psix)**2 * dpsix
  ELSE
    outp = 3.0_R8 * (psix - v)**2 * dpsix + 3.0_R8*(psix)**2 * dpsix
  END IF
END FUNCTION d_phi2

FUNCTION d_vPhi2(x, lambda) RESULT(outp)
  !  Derivative of Phi2 with respect to nu.
  REAL(KIND=R8), INTENT(IN), DIMENSION(:)  ::  x
  REAL(KIND=R8), INTENT(IN) ::  lambda
  !  Local variables.
```

```
  REAL(KIND=R8) ::  outp
  REAL(KIND=R8) ::  psix
  psix = -KS()
  IF (psix - v > 0) THEN
    outp = 3.0_R8 * (psix - v)**2 + 3*v**2
  ELSE
    outp = -3.0_R8 * (psix - v)**2 + 3*v**2
  END IF
END FUNCTION d_vPhi2
FUNCTION CCHomotopyRHO(a, lambda, x, d) RESULT(outp)
  !  Homotopy Rho function, to be passed to HOMPACK90 and called
  !    by HOMPACK90.    Not intended to be called directly.
  !
  !    On input:
  !
  !    a is unused.
  !
  !    lambda is the lambda value, stored in Y(1) by HOMPACK.
  !
  !    x is the collection of mean prototypes, stored in Y(2:length(Y)-2).
  !
  !    d = dims * numprots + 2
  !
  !    On output:
  !
  !    outp contains the current value of rho.
  REAL(KIND=R8), DIMENSION(:), INTENT(IN) ::  a
  REAL(KIND=R8), INTENT(IN) ::  lambda
  REAL(KIND=R8), DIMENSION(:), INTENT(IN) ::  x
  INTEGER, INTENT(IN) ::  d
  REAL(KIND=R8), DIMENSION(d) ::  outp
  !  Local variables.
  INTEGER ::  ind, i, j, consind !  Various indexe.s
  REAL(KIND=R8) ::  oml, thl, omt !  Calculation placeholders.
  REAL(KIND=R8), DIMENSION(d) ::  outC3, out2, out3 !  outp components.
  REAL(KIND=R8), DIMENSION(numcons,d) ::  outt2 !  Parallelization array.
  !  Step 1:  assign meanProt positions based on x and set u and v.
  thl = tanh(60.0_R8 * lambda)
  omt = 1.0_R8 - thl
  oml = 1.0_R8 - lambda
  ind = 0
  outp = 0.0_R8
  out2 = 0.0_R8
  DO i = 1, numprots
    DO j = 1, dims
      ind = ind + 1
      meanProt(j, i) = x(ind)
    END DO
  END DO
  meanProt = meanProt/CCHomotopyDatanorm
  u = x(ind+1)
  v = x(ind+2)
  !  Step Two:    Calculate LHS of RHO.
  !$OMP PARALLEL
  !$OMP DO
```

```
  DO i = 1, numprots
    DO j = 1, dims
      outC3((i-1)*dims + j) = (meanProt(j,i) - aInit(j,i))
    END DO
  END DO
  !$OMP END DO
  !$OMP END PARALLEL
  outC3 = outC3 * omt
  outC3(dims*numprots + 1) = phi(x, lambda)
  outC3(dims*numprots + 2) = phi2(lambda)
  !  Step three:    Calculate penalty function derivative (RHS of RHO).
  !$OMP PARALLEL
  !$OMP DO
  DO consind = 1, numcons
    DO i = 1, d - 2
      outt2(consind, i) = d_Constraint(consind, (((i-1)/dims) + 1), &
          mod((i-1), dims) + 1) * scaleFactor * degOfBelief(consind)
    END DO
  END DO
  !$OMP END DO
  !$OMP END PARALLEL
  out2 = 0.0_R8
  out3 = 0.0_R8
  DO j = 1, d - 2
    DO i = 1, numcons
      out2(j) = out2(j) + outt2(i, j)
    END DO
  END DO
  ind = 0
  DO i = 1, numprots
    DO j = 1, dims
      ind = ind + 1
      out2(ind) = out2(ind) + u * d_psi(i, j) + v * d_KS(i, j)
      out3(ind) = out3(ind) + d_kappr(i, j)
    END DO
  END DO
  out2 = out2 * lambda
  out3 = out3 * oml
  out2 = out3 + out2
  out2 = thl * out2
  outp = outC3 + out2
END FUNCTION CCHomotopyRHO

FUNCTION gradlambdaRHO(x, lambda, d) RESULT(outp)
  !  Gradient of rho w.r.t.  lambda.
  REAL(KIND=R8), INTENT(IN), DIMENSION(:)  ::  x
  REAL(KIND=R8), INTENT(IN) ::  lambda
  INTEGER, INTENT(IN) ::  d
  REAL(KIND=R8), DIMENSION(d) ::  outp
  !  Local variables.
  INTEGER ::  i, j, ind, consind
  REAL(KIND=R8) ::  dtanh
  REAL(KIND=R8) ::  omt, omtd
  REAL(KIND=R8) ::  oml, omld
  REAL(KIND=R8) ::  temp
  REAL(KIND=R8) ::  thl, thld
```

```
REAL(KIND=R8), DIMENSION(d) ::  outC3, out2, out3
REAL(KIND=R8), DIMENSION(numcons,dims) ::  outt2
ind = 0
outp = 0.0_R8
out2 = 0.0_R8
outC3 = 0.0_R8
DO i = 1, numprots
   DO j = 1, dims
      ind = ind + 1
      meanProt(j, i) = x(ind)
   END DO
END DO
meanProt = meanProt/CCHomotopyDatanorm
u = x(ind+1)
v = x(ind+2)
ind = 0
thl = TANH(60.0_R8 * lambda)
thld = 60.0_R8 * (1/COSH(60.0_R8 * lambda))**2
omt = 1.0_R8 - thl
omtd = -thld
oml = 1.0_R8 - lambda
omld = -1.0
!$OMP PARALLEL
!$OMP DO
DO i = 1, numprots
   DO j = 1, dims
      outC3((i-1)*dims + j) = (meanProt(j,i) - aInit(j,i))
   END DO
END DO
!$OMP END DO
!$OMP END PARALLEL
outC3 = outC3 * omtd
outC3(dims*numprots + 1) = h0
outC3(dims*numprots + 2) = h1
!$OMP PARALLEL
!$OMP DO
DO consind = 1, numcons
   DO i = 1, d - 2
      outt2(consind, i) = d_Constraint(consind, (((i-1)/dims) + 1), &
          mod((i-1), dims) + 1) * scaleFactor * degOfBelief(consind)
   END DO
END DO
!$OMP END DO
!$OMP END PARALLEL
out2 = 0.0_R8
out3 = 0.0_R8
DO j = 1, d - 2
   DO i = 1, numcons
      out2(j) = out2(j) + outt2(i, j)
   END DO
END DO
ind = 0
DO i = 1, numprots
   DO j = 1, dims
      ind = ind + 1
```

```
      out2(ind) = out2(ind) + u * d_psi(i, j) + v * d_KS(i, j)
      out3(ind) = out3(ind) + d_kappr(i, j)
    END DO
  END DO
  out2 = out2 * thl + out2 * lambda * thld
  out3 = out3 * oml * thld + out3 * omld * thl
  out2 = out3 + out2
  outp = outC3 + out2
END FUNCTION gradlambdaRHO

FUNCTION gradRHO(x, lambda, dM21, dM22, d) RESULT(outp)
  !  Gradient of rho w.r.t.  meanProt(dM22, dM21).
  REAL(KIND=R8), INTENT(IN), DIMENSION(:)  ::  x
  REAL(KIND=R8), INTENT(IN) ::  lambda
  INTEGER, INTENT(IN) ::  dM21, dM22
  INTEGER, INTENT (IN) ::  d !  d = numprots * (dims) + 2.
  REAL(KIND=R8), DIMENSION(d) ::  outp
  !  Local variables.
  REAL(KIND=R8), DIMENSION(d) ::  outC3, out2, out3
  REAL(KIND=R8) ::  temp, oml, omt, thl
  INTEGER ::  consind, i, j, ind
  INTEGER ::  uvswitch
  REAL(KIND=R8), DIMENSION(numcons, dims) ::  outt2

  thl = tanh(60.0_R8 * lambda)
  omt = 1.0_R8 - thl
  oml = 1.0_R8 - lambda
  ind = 0
  outp = 0.0_R8
  out2 = 0.0_R8
  DO i = 1, numprots
    DO j = 1, dims
      ind = ind + 1
      meanProt(j, i) = x(ind)
    END DO
  END DO
  meanProt = meanProt/CCHomotopyDatanorm
  u = x(ind+1)
  v = x(ind+2)
  !  uvswitch:    Calculate whether we are looking for the derivative w.r.t.  mu,
  !    nu.
  uvswitch = 0
  IF (dM21 > numprots) THEN
    uvswitch = dM22
  END IF
  IF (uvswitch == 0) THEN
    !  Step Two:    Calculate LHS of dRho.
    outC3 = 0.0_R8
    DO i = 1, numprots
      DO j = 1, dims
        IF ((i == dM21) .AND. (j == dM22)) THEN
          outC3((i-1)*dims + j) = 1.0_R8
        END IF
      END DO
    END DO
    outC3 = outC3 * omt
    outC3(dims*numprots + 1) = d_phi(dM21, dM22, x)
```

```
  outC3(dims*numprots + 2) = d_phi2(dM21, dM22, x)
  !  Step three:    Calculate penalty function derivative (RHS of RHO).
  !$OMP PARALLEL
  !$OMP DO
  DO consind = 1, numcons
    DO i = 1, d - 2
      outt2(consind, i) = dd_Constraint(consind, (((i-1)/dims) + 1), &
          mod((i-1), dims) + 1, dM21, dM22) * scaleFactor * &
          degOfBelief(consind)
    END DO
  END DO
  !$OMP END DO
  !$OMP END PARALLEL
  out2 = 0.0_R8
  out3 = 0.0_R8
  DO j = 1, d - 2
    DO i = 1, numcons
      out2(j) = out2(j) + outt2(i, j)
    END DO
  END DO
  ind = 0
  DO i = 1, numprots
    DO j = 1, dims
      ind = ind + 1
      out2(ind) = out2(ind) + u * dd_psi(i, j, dM21, dM22) + &
          v * dd_KS(i, j, dM21, dM22)
      out3(ind) = out3(ind) + dd_kappr(i, j, dM21, dM22)
    END DO
  END DO
  out2 = out2 * lambda
  out3 = out3 * oml
  out2 = out3 + out2
  out2 = thl * out2
  outp = outC3 + out2
ELSE IF (uvswitch == 1) THEN ! w.r.t.  mu.
  outp = 0.0_R8
  ind = 0
  DO i = 1, numprots
    DO j = 1, dims
      ind = ind + 1
      out2(ind) = d_psi(i, j)
    END DO
  END DO
  outp = outp * lambda * thl
  outp(ind+1) = d_uPhi(x, lambda)
ELSE ! w.r.t.  nu.
  outp = 0.0_R8
  ind = 0
  DO i = 1, numprots
    DO j = 1, dims
      ind = ind + 1
      out2(ind) = d_KS(i, j)
    END DO
  END DO
  outp = outp * lambda * thl
  outp(ind+2) = d_vPhi2(x, lambda)
```

```
  END IF
 END FUNCTION gradRHO

 FUNCTION CCHomotopyJAC(a, lambda, x, k, d) RESULT(outp)
  !  Homotopy Rho function, to be passed to HOMPACK90 and called
  !    by HOMPACK90.    Not intended to be called directly.
  !
  !    On input:
  !
  !    a is unused.
  !
  !    lambda is the lambda value, stored in Y(1) by HOMPACK.
  !
  !    x is the collection of mean prototypes, stored in Y(2:length(Y)-2).
  !
  !    k is the current column of the jacobian being evaluated.
  !
  !    d = dims * numprots + 2
  !
  !    On output:
  !
  !    outp contains the current value of the current column of the jacobian
  !        of rho.
  REAL(KIND=R8), DIMENSION(:), INTENT (IN) ::  a !  Unused parameter variables.
  REAL(KIND=R8), INTENT(IN) ::  lambda !  Lambda.
  REAL(KIND=R8), DIMENSION(:)  ::  x !  Mean prototypes, mu, and nu.
  INTEGER, INTENT(IN) ::  k !  Column of the jacobian.
  INTEGER, INTENT(IN) ::  d !  dims * numprots + 2.
  REAL(KIND=R8), DIMENSION(d) ::  outp !  Column of Jacobian vector.
  !  Local variables.
  INTEGER ::  i, j, ind !  Indices.
  INTEGER ::  temp, k1, k2 !  Used to calculate information to pass to gradient
  !    functions

  ind = 0
  DO i = 1, numprots
    DO j = 1, dims
      ind = ind + 1
      meanProt(j, i) = x(ind)
    END DO
  END DO
  meanProt = meanProt/CCHomotopyDatanorm
  IF (K == 1) THEN
    !  Lambda Jacobian.
    outp = gradlambdaRHO(x, lambda, d)
  ELSE
    !  Non-lambda Jacobian.
    temp = k - 1
    k1 = ((temp - 1) / dims) + 1
    k2 = MOD(temp - 1, dims) + 1
    outp = gradRHO(x, lambda, k1, k2, d)
  END IF
 END FUNCTION CCHomotopyJAC

END MODULE CC_HOMOTOPY_MOD
```