Machine Learning-Based Parameter Validation

Noah Garcia Badayos

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
Electrical Engineering

Virgilio A. Centeno, Chair
Arun G. Phadke
Jaime De La Ree Lopez
C. Yaman Evrenosoğlu
Sandeep Shukla
Naren Ramakrishnan

March 21, 2014
Blacksburg, Virginia

Keywords: power systems, model validation, machine learning, data mining, random forest, Prony analysis, phasor measurement units

Machine Learning-Based Parameter Validation

Noah Garcia Badayos

Abstract

As power system grids continue to grow in order to support an increasing energy demand, the system's behavior accordingly evolves, continuing to challenge designs for maintaining security. It has become apparent in the past few years that, as much as discovering vulnerabilities in the power network, accurate simulations are very critical. This study explores a classification method for validating simulation models, using disturbance measurements from phasor measurement units (PMU). The technique used employs the Random Forest learning algorithm to find a correlation between specific model parameter changes, and the variations in the dynamic response. Also, the measurements used for building and evaluating the classifiers were characterized using Prony decomposition. The generator model, consisting of an exciter, governor, and its standard parameters have been validated using short circuit faults. Single-error classifiers were first tested, where the accuracies of the classifiers built using positive, negative, and zero sequence measurements were compared. The negative sequence measurements have consistently produced the best classifiers, with majority of the parameter classes attaining F-measure accuracies greater than 90%. A multiple-parameter error technique for validation has also been developed and tested on standard generator parameters. Only a few target parameter classes had good accuracies in the presence of multiple parameter errors, but the results were enough to permit a sequential process of validation, where elimination of a highly detectable error can improve the accuracy of suspect errors dependent on the former's removal, and continuing the procedure until all corrections are covered.

To my family.

Acknowledgements

I would first like to express my sincerest gratitude to my advisor, Dr. Virgilio Centeno, who helped bring this research idea into fruition. Everything from your expert advice, encouragement, to the out-of-topic conversations during what little idle time we had, motivated the completion of this endeavor. It was a pleasure having you as a guide in this journey!

To Dr. Arun Phadke, Dr. Jaime De La Ree, Dr. Yaman Evrenosoğlu, Dr. Sandeep Shukla and Dr. Naren Ramakrishnan, you have all molded and buffed this piece of work into its final shape, and I carry it with much satisfaction knowing it has kept me on track, and given me the hope that it will continue to grow in refinement, or at the least be worthy of anyone's attention. For this I am very much indebted to you all.

My power labmates, there are too many of you to mention, but know that I treasure every profound discussion or silly joke we have shared. You are the company and challenge I needed in graduate school, any less then I will not be able to effectively solve world problems, right?

To all my Filipino friends who once lived, or still live, in this little town, beginning with those who first welcomed me into the Pinoy Hokie community—Joan Zapiter, Ate Lynn Rallos and Kuya Jess Calata—my transition could not have been easier without you all. To Analyn, Roche, RJ, the Vejerano, Doria, Hipkins, Sanciangco, Pritchard, Ozaraga, Buhyoff, Junio, Mendoza, Ballweg families, and countless others who filled tables with sumptuous food and hearty laughter during parties, I am glad that even for brief moments I could forget I was in a foreign land.

Without the love and support from my parents, Rod and Noni, my dreams will have remained impossible, and I will have been ill-prepared for the world beyond home. Kuya Lex, Ate Rp, Dino, Brielle and Chloe, your happiness, even the faintest of smiles, have been a source of comfort and strength. In spite of the distance, you all have kept me afloat at all times, and I will forever be grateful that He blessed me with such a beautiful family.

Maraming salamat (Thank you very much)!

Table of Contents

List of Figures

List of Tables

1. Introduction

Prior to a series of unrestrained power system disturbances in the summer of 1996, the Western Electricity Coordinating Council (WECC)[1] power network was heavily loaded, as is typical of the season. This prompted heavy importation of energy from the general northern regions. At that same moment, the system was beset with forced 500 kV outages, specifically the John Day-Marion Lane and the Big Eddy-Ostrander lines. In addition, the 500/230 kV transformer at the Keeler substation was going through scheduled maintenance, weakening the reactive power support at the station's 500 kV bus. The fragile condition of the system needed only a small push to cause it to breakdown, as it did very soon. The 500 kV forced line outages caused other parts of the network to carry more than their normal load. After the first overloaded line tripped, the power flow had drastic shifts resulting in more lines opening. The state of the grid was even further aggravated by equipment failures in parts of the system. As these events transpired, the low frequency inter-area oscillations magnified, ultimately leading to a system split, causing a significant loss of load and affecting millions [1, 2].

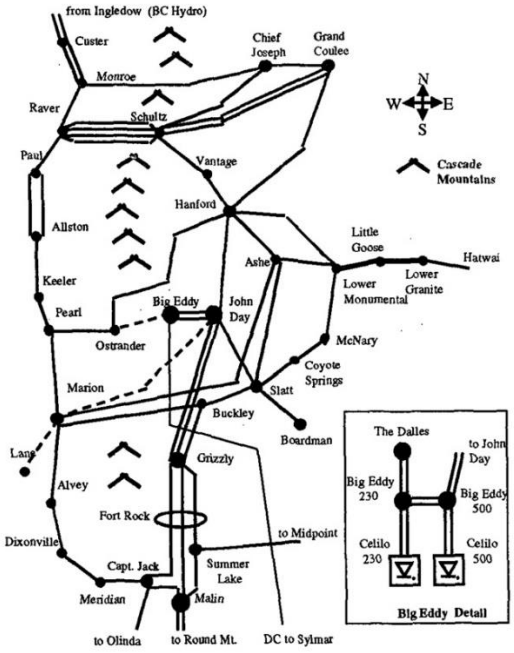

Figure 1-1. WECC System [1]

[1] Formerly known as WSCC, or Western Systems Coordinating Council.

1.1 Troubleshooting the Network

Validation studies were conducted to discover the mechanism of the series of events. Such procedures guide future activities at reinforcing the system and correcting operating practices to prevent a repeat of the disaster [2]. For the WECC event in particular, it was discovered that there was a very poor match between the measurements and the simulations. Using their stock system model, after checking quantities such as change in frequency, damping of oscillations, voltage profiles, and line flows, the extensive analyses have shown that with the same given conditions, the simulation traces a less violent response [1].

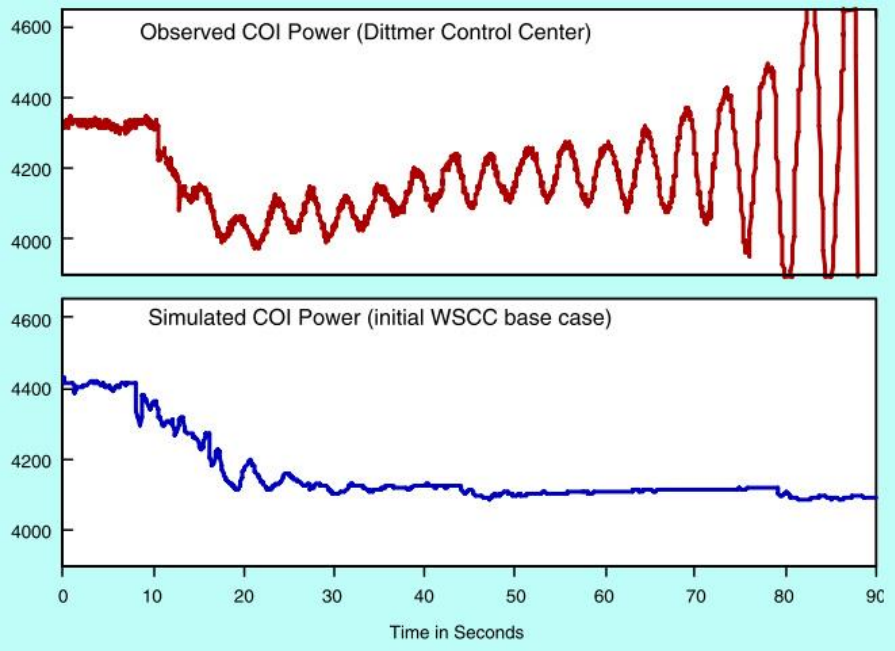

Figure 1-2. Difference between measured (top) and simulated (bottom) traces from the California-Oregon Intertie (COI) [3]

To assess the WECC disturbance, an application called EPRI ETMSP[2] was used for time-domain simulations to compare the recorded power flow and dynamics of the system, before and during the series of events. The dynamic responses were analyzed using two techniques. One was Prony's method, which enables extraction of frequency, damping, magnitude, phase and modes of oscillation. This was applied to both the time-domain simulations and the corresponding recorded signal. The other method applied was small signal stability eigenvalue analysis, whose output can

[2] EPRI ETMSP – EPRI's Extended Transient/ Midterm Stability Program. EPRI stands for Electric Power Research Institute.

be compared against Prony. Small signal stability analysis solves for the modes of oscillation from the response of a linearized power system model. Unlike Prony, small signal analysis can provide the necessary information for identifying the root cause of the system behavior, which will help in the subsequent formulation of remedial measures and model correction [2].

1.2 Power System Dynamic Simulation

The simulation of the dynamic behavior of a power system network, in general, describes the electro-mechanical response of a system subjected to disturbances such as faults, loss of transmission paths, generation and loads. Simulation studies involve transients that influence, or work at the same scale as the rotating inertias of turbine-generator rotors, and are mainly concerned with the ability of the generator to maintain synchronism when subjected to a disturbance [4]. While detailed analysis of the dynamics of rotating inertias were implemented in the early form of simulations, it was not until the significant advances in computing that the electromagnetic transients were factored in. The newly available technology also permitted more sophisticated solutions to old and new problems encountered by power engineers.

Power system simulations serve as the main tool for both planning and operation, with applications such as security evaluation, system reinforcement, control design and others. These simulations, can either be offline or real-time. Offline simulations traditionally solve the non-linear differential equations describing the power network and its connected components. Real-time, on the other hand, are designed to run in synchronism with the actual power network. This enables testing and evaluation of real substation hardware Intelligent Electronic Devices (IED) such as protection and measurement equipment. Simulation with hardware interfacing capability is otherwise known as hardware-in-the-loop testing [5].

The increasing complexity of the modern power network introduces new difficulties with managing, and analyzing the system as a whole. It has made the fundamental problem of running dynamic simulations more challenging, as the contribution of parameters to the dynamics of the network become more compounded. More often than not, the designed simulation components cannot exactly follow the behavior of the counterpart component in the actual system. Instead, models typically assume a simplified, or approximate form.

1.3 Power System Measurements: Synchrophasors

The power system networks continue to grow bigger and more interconnected, to support a continually increasing demand in power. Consequently, an accurate prediction of the dynamics, as well as the development of optimal control to maintain system security, is a constant challenge at present. The complex system has inevitably become a more delicate structure, in that the effect of a disturbance in one part of the network can be felt a significant distance away [6]. The ability to accurately describe the state of the system, which in this case is in terms of the voltage magnitude and angle at each bus, has become extremely valuable to power system planning and operation. This need has been fulfilled by phasor measurement units (PMU), using synchrophasors.

Representation of alternating current (AC) quantities in phasor form was developed by Charles Steinmetz, and has since become a regular feature in electrical analysis [7]. Extraction of the fundamental harmonic component, which is the basis of the positive sequence phasor, was made possible by a digital signal processing technique called the Discrete Fourier Transform (DFT). For a given stream of N samples x_k, the fundamental frequency, or the phasor quantity at nominal frequency, is described by

$$\bar{X}_1 = \frac{\sqrt{2}}{N}\sum_{k=0}^{N-1} x_k e^{-\frac{j2\pi}{N}k} \tag{1.1}$$

The system phasors are all computed with respect to a single time reference, and this synchronization is made possible using global positioning satellite (GPS) clock signals [8], hence the term synchrophasor. The simultaneous accurate measurement of the network phasors, which has become an integral part of Wide Area Measurement Systems (WAMS), gives an excellent snapshot of the system state, limited by the rate at which the phasor data is streamed. It has since been used in several protection, operation and control applications.

1.4 Model Validation

In the event of a disturbance, it is of absolute importance to ensure that any interruption to service is contained and restored in the shortest amount of time possible. Power system

disturbances are very common and very often easy to recover from. However, there are rare events that lead to catastrophic, cascading failures, such as what was described earlier in this chapter. It is therefore vital to make accurate evaluations of the system's health, and subsequently draw good predictions as the network evolves from one state to the next.

Computer simulation of the power system network, comprised of countless component models, are used extensively to analyze the effect of incremental changes in the network. To validate the results of the simulation, comparisons are made, especially during major outage events, between the simulation output and the field measurements. Differences between simulated and recorded data are not an unusual occurrence, and the discrepancies in the projected dynamic response prompts system model correction. Model changes are made until a satisfactory matching response is obtained. A significant number of model adjustments often have to be applied, typically never attaining a perfect match. Moreover, it is possible that the correction applied may only be valid for the event being tested [6]. The inadequacies in models, which can either have an optimistic or conservative outcome for simulations, often do not manifest themselves in steady-state or pre-disturbance conditions, rather they are revealed during a disturbance.

The simulations could only make estimates of the actual condition of the equipment, and nothing can be as conclusive as direct inspections on the equipment concerned. Since the major WECC outage, various efforts to standardize power plant testing and validation have been conceived [9]. For the North American continent, the North American Electric Reliability Corporation (NERC) enforces guidelines to ensure that reliability standards are met all throughout the US and some parts of Canada. All tests are conducted to evaluate power plant performance, correct control problems and verify plant control compliance [3]. Regular validation tests are very necessary, considering the possibility of equipment properties varying with time due to wear and tear.

Traditional onsite testing and validation, which necessitate direct interaction with the machines and other system components, requires units to be offline. The tests are typically conducted during the initial commissioning of the plant, before maintenance, and after major retrofits. This, however, can potentially affect the system security, especially when units are forced to go out of service during high energy demand. Furthermore, during tests, major delays caused

by human error, equipment damage, and other unforeseen circumstances can be very costly [3]. Onsite validations are characterized by thorough evaluations of the models, which may often involve very intrusive testing procedures run until a satisfactory response is obtained. Although needless testing and repetition can be avoided with sound advice from well-seasoned experts, who can provide insight into the events, this is often not an easily accessible resource.

1.5 Proposed Solution for Validation

Various techniques have been developed to validate system models. Most of which have been devised as alternatives to traditional onsite techniques. The methods cover everything from the so called *hybrid simulations* [5, 10], to fitting dynamic model responses, using a very mathematically involved *trajectory sensitivity method* [11, 12]. This study investigates an alternative method for model validation, which uses disturbance information, and a machine learning technique called Random Forest (RF). The decision to use a data mining approach for solving this problem stems from the fact that power systems, due to great advances in computing technology in the past few years, has had the benefit of getting faster, information-rich and more accurate measurements.

Naturally, one would suspect that with the new tools, the wealth of information streamed from the devices may potentially contain patterns useful for identifying specific model parameter errors. This study takes advantage of synchrophasor measurements to validate the errors, as the PMU can provide voltage, current and frequency information, among others, in excellent resolution. Moreover, due to its capability to deliver critical information from the system, it is steadily gaining a larger coverage, which would make validation very accessible to most participating machines and other equipment in the network.

The technique developed validates models by using an RF-generated classifier. The classifier identifies the erroneous parameter, and indicates the required direction of adjustment, e.g. increase or decrease value, all according to the input, which in this case are disturbance synchrophasor measurements. In order to generate the classifier, the simulation is configured to run as the disturbance event happened, with all the known equipment settings set. The classifier is generated by purposefully changing model parameter values randomly, all within set ranges that

define the error classes. Moreover, the synchrophasor sample streams are characterized, prior to training and testing, using Prony's method for decomposing signals into decaying sinusoids. This creates more meaningful entries into the training data as predictor variables, and it also compresses the size of data for evaluation.

1.6 Overview

The next chapter of the dissertation is a survey of relevant validation methods, covering both traditional and novel techniques. Chapters 3 and 4 review the applied concepts, Random Forest and Prony analysis, respectively. A detailed documentation of the method used in the study is found at Chapter 5, followed by the analysis of the results at Chapter 6. Finally, the conclusions, contributions, and proposed future research are highlighted in Chapter 7.

2 Review of Validation Methods

Power system simulation is an essential tool for applications in both planning and operation. The ability to properly dispatch the correct amount of generation days or even years ahead of time, the determination of the operating transfer limits and transmission network reinforcements, assessment of the system's security against events of varying likelihood, and the ability to test the effectiveness of a protection scheme are just a few uses of simulation [5]. The models, such as those used to represent generators, and loads, form the basic components of the simulation. A model is composed of mathematical expressions that attempt to describe its interaction with a selected environment. These expressions can also assume different forms and settings, depending on the required analysis at hand. For power system component models, the parameters that characterize the mathematical expressions are typically based on their respective manufacturer's fresh specifications, or only an estimate made by onsite testing prior to commissioning.

The network model's behavior greatly depend on these equipment configurations, many of which usually have very subtle responses or exhibit no measureable effects for normal operations, and can therefore be very difficult to validate. It is through perturbations, such as network disturbances, that a system component can fully manifest its unique character, that which a model can match with its array of parameters. However, these disturbances are rarely repeated, much less appear in the same location with the same intensity or properties. Moreover, as power systems continue to grow, the interaction of the components continue to evolve. Failure to keep up with the changes, coupled with the equipment behavior drifting from its nominal ratings due to wear and tear, can cause unexpected behavior that may result to catastrophic events in power systems.

The August 10, 1996 WSCC breakdown was one such event [1, 2]. The measurements collected from the incident have shown that the computer simulations register a different response from what was observed during the incident. The poorly modeled components failed to show the severity of the event, which could have been avoided otherwise. The incident prompted an overhaul of the model used for simulations. Among the changes made were replacing the over-simplified Pacific HVDC Intertie (PDCI) model with a more detailed one, modifications on a few

control actions, and corrections on the load model. The load model in particular was based on a predominant use of air conditioning and irrigation of rural areas in that season. The criteria used for validating the models were the frequency and damping of the inter-area oscillation, the change in PDCI behavior, the voltage current and frequency average profiles, and the distribution of the oscillatory model in the system.

2.1 Onsite Validation Methods

The adjustments made to the WECC network successfully corrected the discrepancies. However, this was accomplished without any definitive method or guideline that can be reused by other utilities. Repeated trial and error, plus reliance on an experienced utility personnel's intuition may greatly help with solving the task, but it is by no means efficient nor readily available or practical. With power networks constantly evolving, one can be sure that no single event is useful for an indefinite amount of time for re-validation. This fact makes periodic, onsite testing and validation of equipment the most dependable of all methods, in terms of obtaining good estimates of parameters. The following section gives a survey of a few onsite testing methods used in the industry

2.1.1 Staged Tests [3]

The staged tests are designed to draw the required response from the machine and its controls to extract the required parameters. They are regarded to be the one of the simplest, least time consuming and most direct way of estimating the parameters. The method does not require control systems to be dismantled, although some protection features have to be disabled. Also, these tests are least likely to cause damage or unintentional tripping of the unit.

A. Generator Parameters

The complete synchronous machine equations that describe the electrical characteristics, while accurate, cannot be directly measured using the responses of the machine [13]. As a result, they are expressed in terms of derived parameters that are observable from the terminals under specific test conditions.

To measure the generator parameters, a method called reactive power rejection test is applied to the machine. The generator is setup such that it is not producing real power, and instead absorbing reactive power from the system. This forces the internal flux to completely align with the direct axis (d-axis), which consequently causes the internal voltage to be at the quadrature axis (q-axis) as shown in Figure 2-1.

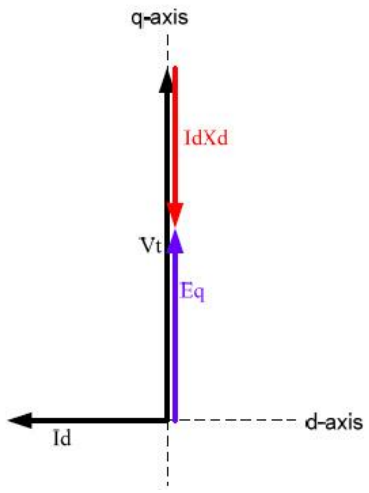

Figure 2-1. Zero Power Factor Phasor Diagram [3]

With the field excitation voltage held constant, and subsequently disconnecting the generator from the system feeding the reactive power, a subtransient and transient voltage decay is expected to ensue. With the internal flux lying at the d-axis, the parameters Xd, X'd, X''d, T'do and T''do can be estimated. The q-axis can be derived from the d-axis parameters using the estimated relationships shown in Table 2-1, and the stator leakage reactance is assumed to be 80% of X''d.

Table 2-1. Relationship of d and q-axis parameters [3]

Parameter	Round Rotor Machine	Salient Pole Machines for Hydro Units
Xq	≈0.9 Xd	≈0.6 to 0.7 Xd
X'q	≈1.5 X'd	0
X''q	≈X''d[9]	≈X''d
T'qo	≈0.3 T'do	0
T''qo	≈T''do	≈T''do

Another method, which combines a real and reactive power rejection test can be used to estimate the q-axis parameters. This process initially requires the generator flux to be aligned along the q-

axis. Upon disconnection, the rotor angle is recorded, and the transient response of the rotor angle will be the basis for the estimates. The method however needs reasonable initial estimates of the q-axis parameters to properly align the generator flux along the q-axis, and this may entail repeated testing. Moreover, the mechanical speed of the shaft needs to be measured to obtain the rotor angle, which may not readily be available in the machine, thus requiring a complete shutdown of the unit. Other studies covering direct testing for synchronous machine parameter estimation can be seen in [14-16].

B. Excitation System Parameters

The staged test for exciters can be done in two ways. One is applying a voltage step response, and the other by using the reactive power rejection test similar to what was used for the generator. The step response can be performed online or offline, and it simply done by applying small incremental voltage steps into the regulator and recording the response. For the reactive power rejection, the same procedure used in the generator test is applied, except that the automatic voltage regulator (AVR) is left active. Upon disconnection of the generator from the system, the test records the AVR's response, which is expected to act to maintain the terminal voltage of the generator.

C. Turbine-Governor and Unit Inertia

The turbine-governor is mainly checked for three features. One is the droop setting, the mode of operation, and the turbine-governor's response time. One of the many ways to estimate the droop, is by recording the unit's input valve position, against its power output. Also, while the unit is offline, the governor's speed-load reference set point can be measured for varying turbine speeds. The turbine-governor droop can then be calculated as

$$R = \frac{\Delta\omega}{\Delta P}, \tag{2.1}$$

where $\Delta\omega$ is the change in speed offline and ΔP is the change in unit power. Both quantities should be measured against the same change in reference speed. A uniform droop characteristic as computed above can be assumed for gas turbines. The hydro and steam turbines however have non-linear characteristics, which will then require a more sophisticated method of calculation.

The mode of operation can be easily identified on-site, while the turbine-governor response can be evaluated using a "megawatt rejection test or a speed-load reference step test".

2.1.2 Frequency Response-Based Techniques [3]

Another group of testing methods for parameter estimation is called Stand Still Frequency Response (SSFR) [17]. The process requires isolating the component that needs to be modeled, and injecting signals generated using precision signal generator and amplifiers. The signals are voltage waveforms, and precisely varied in frequency and amplitude. The output gain and phase difference observed are measured, which will then determine the transfer function describing the component under test. When this is applied to generators, the rotor is locked, ensuring proper alignment along the required axis to derive the parameters.

It is a requirement for this method that the model structure is known prior to the tests. The techniques employed for this parameter estimation method use algorithms such as recursive least-squares estimation and maximum likelihood estimation. To use this method, it has been recommended to use 3rd order models, which presents a difficulty since most available models in North America use 2nd order equations. The technique overall is more accurate than staged tests, but at the same time is more intrusive and more time-consuming.

Onsite testing of equipment can be the most inconvenient of all methods for validation. Most of the techniques employed in this method requires the equipment to go out of service. The practice can be especially impractical on instances when generation in the system barely catches up with the demand, and the network generally being short on available alternative power sources.

2.2 Parameter Estimation Based on Disturbance Monitoring

While frequency response-based testing is not widely adapted, mainly due to its very intrusive nature, staged testing poses a concern, especially for nuclear generating stations. The power industry is considering estimating parameters using an alternative method, and that is through analyzing a power system network's dynamic response to a disturbance. It has been stated that for this method to be effectively implemented, the initial simulation model must already be "reasonably accurate", from power plant equipment to load. This is primarily due to the fact that

present systems contain thousands of component models, which will make the validation process too complicated for each piece of the model in the network checked for every iteration. Disturbance-based model validation is however thought of only as a complementary procedure to traditional power plant testing. Also, it has been speculated that for this to be realized, network devices should have very high resolution samples, and would require more measurements than what is typically available, such as including excitation voltage monitoring for generators [3].

2.2.1 Hybrid Simulation

One of the model validation methods developed to address the difficulty of the validation task after the WECC collapse simplifies the problem by partitioning the system, and working at one small section at a time [18]. At the boundaries of the partition the recorded measurements are used as injections during simulation, instead of replacing boundary connections with equivalent system models. It uses an infinite-bus model (Figure 2-2), which feeds the recorded bus voltage and frequency data, then captures the matching response from the model under test. This is realized by using a large synchronous machine simulation model with fast-responding exciters and governors. The resulting power output from the simulation is compared to the measurements for validation. The target measurement source for this method are the Phasor Measurement Units (PMU).

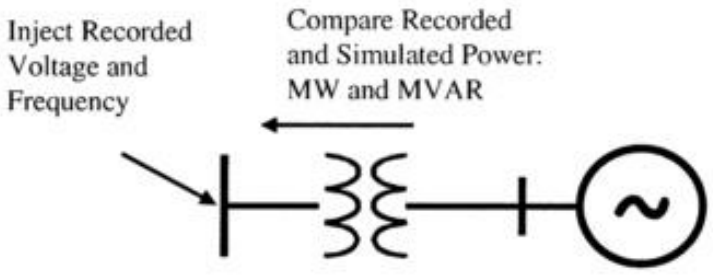

Figure 2-2. Boundary substitute for dynamic simulation [18]

A general procedure for this validation method has been outlined in [18]. It begins with the acquisition of the participating dynamic models from the generating plant. The collected data will be subject to review by the control area under which the models are used. The performance of the plant will then be monitored by collecting bus voltage frequency, powerhouse line currents and

control signals. Finally, with all measurements in place, any recorded response from the plant can be analyzed and compared with the simulation response for discrepancies.

This aforementioned technique became the basis of several other validation methods known as the *Hybrid simulation*. This simulation technique's strength relies on accurate playback of field measurements at the boundaries of networks, which in theory can effectively isolate the trouble areas of a system. The recorded signals injected into the boundary of the subsystem is claimed to eliminate errors that may be introduced by the traditional method of partitioning a network using equivalent models [5, 10, 19, 20].

The following method uses a slightly different implementation than what was described in [18]. In this version of Hybrid simulation, the recorded signals are injected using an artificial generator connected to an ideal transformer at the boundary (Figure 2-3). This generator has the following properties [5]:

1. The generator used is a classical model.
2. The internal reactance of the generator should be zero. To facilitate numerical calculation, a near-zero value is used instead.
3. Inertia constant is a large number.
4. The transformer it is connected to has a near-zero impedance.

The initial conditions set are:

1. Generator voltage is 1 *pu*.
2. The initial power output is as measured at the boundary prior to the event.
3. The initial ratio of the transformer is according to the voltage at the boundary.
4. The initial phase shift is as recorded at the boundary.

For the simulation, the generator is set to run with a constant voltage E behind the internal impedance, and the ideal transformer will playback the recorded changes in the voltage, with changes occurring at every time step. By employing the hybrid simulation to a large system, validation can be facilitated through reduction of the network into manageable subsections. These partitions can be more conveniently inspected for inconsistencies between measured and simulated measurements.

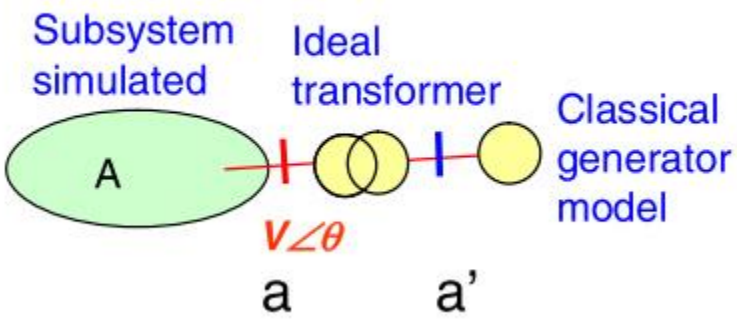

Figure 2-3. Boundary substitution using a classical generator model and an ideal phase shifter [5].

It has been shown that the dynamic simulation with the aid of external measurements can very accurately replicate an event, regardless of the isolation of the subsection. The key to this technique is that the injected measurements will be able to transmit any external perturbations into the subsystem under test. This is a feature that cannot be easily implemented in traditional system reduction methods.

The recorded signals injected into the subsystem at the boundary will induce a response, which then will provide an indication of the presence of erroneous parameters. The power system measurements applied at the boundary are voltage magnitude, angle/frequency which can implicitly contain the disturbance information. The resulting real and reactive power caused by the external input will be compared to the recorded real and reactive quantities. All the measurements are obtained using phasor measurement units.

The investigation cited in [19] has tested systems undergoing frequency excursion and system oscillations, and has demonstrated its ability to identify areas that need correction. The system under evaluation required extensive trial and error to achieve a satisfactory match between the readings from the simulation output and the measurements. Validation of the generators inside a subsection of the power grid using the hybrid technique still required careful tests, to identify variables in the model that needed adjustment.

An increased availability of PMUs will allow a decreasing system partition size to effectively pinpoint the source of the error, but a subsection may still require meticulous tests, depending on the complexity of the models contained in it. Furthermore, it has been emphasized in hybrid simulation studies that the measurements should exactly match the simulation time step.

This can be ensured by data pre-processing, which fixes missing data points, loss of synchronism, and uneven time intervals.

The hybrid method has been extended to validating load models in [10]. This validation compares aggregated transmission-level loads against detailed distribution-level loads. The detailed model consists of a motor and ZIP (constant impedance, current and power) load for each distribution load branch in a radial system. The process helps determine the adequacy of loads used for simulations, and can provide insights into what kind of improvements can be made. Difficulty is encountered when the load contains multiple motor loads, which may not be satisfactorily reduced in aggregate form. An exact aggregate equivalent is almost impossible to achieve, and instead, parameter sensitivity is determined to identify the critical load parameters to improve the aggregate models.

2.2.2 Trajectory Sensitivity

Another technique used to aid in model validation is called the trajectory sensitivity analysis covered in [11, 12, 21-23]. This method can be used in conjunction with the hybrid simulation technique [20, 24, 25], where the reduced subsystem allows the problematic parameter to be easily targeted and subsequently corrected. Particularly, in [20], trajectory sensitivity is used to search for parameters that have the largest influence on the dynamics of a disturbance, then an extended Kalman filter (EKF) is employed to tune the selected variables by reducing the mismatch between the actual and predicted measurements. While excellent accuracies can be achieved using the trajectory sensitivity method, in general, the procedure requires a very thorough description of the model and the event. As such, performing validation on a complex series of events involving a sizable number of component models can be quite challenging, as no small information can be spared in the process.

Like the previous methods, trajectory sensitivity takes advantage of disturbance data, and can specifically work with systems that generate signals which is a mix of continuous time dynamics, discrete-time and discrete event dynamics, all of which can seemingly fit any randomly mixed power system event. Such systems, otherwise known as hybrid systems can be modeled using switched differential-algebraic equations

$$\dot{\underline{x}} = \underline{f}(\underline{x}, y)$$

$$0 = \begin{cases} g^{-}(\underline{x}, y) & s(\underline{x}, y) < 0 \\ g^{+}(\underline{x}, y) & s(\underline{x}, y) > 0 \end{cases} \tag{2.2}$$

$$\underline{x}^{+} = \underline{h}(\underline{x}^{-}, y^{-}) \qquad s(\underline{x}, y) = 0$$

where $\underline{x}^{+} = [x^t \quad z^t \quad \lambda^t]^t$, x are the continuous dynamic states, z the discrete dynamic states, y the algebraic states, and λ the parameters. The dynamics of a system away from a discontinuity is (add number to equations mark 1)

$$\begin{aligned} \dot{\underline{x}} &= \underline{f}(\underline{x}, y) \\ 0 &= g(\underline{x}, y) \end{aligned} \tag{2.3}$$

And the flows defined as

$$\begin{aligned} \underline{x}(t) &= \phi_{\underline{x}}(\underline{x}_0, t) \\ y(t) &= \phi_y(\underline{x}_0, t) \end{aligned} \tag{2.4}$$

With initial conditions

$$\begin{aligned} \phi_{\underline{x}}(\underline{x}_0, t_0) &= \underline{x}_0 \\ g(\underline{x}_0, \phi_y(\underline{x}_0, t_0) &= 0 \end{aligned} \tag{2.5}$$

The variation of the quantities based on changes in selected parameters, or the trajectory sensitivities, are derived from the set of hybrid equations. The sensitivity of the flows can be obtained by first solving for the Taylor series expansion. Neglecting the higher order terms yields

$$\Delta\underline{x}(t) = \frac{\delta\underline{x}(t)}{\delta\underline{x}_0}\Delta\underline{x}_0 \equiv \underline{x}_{\underline{x}_0}(t)\Delta\underline{x}_0 \tag{2.6}$$

$$\Delta y(t) = \frac{\delta y(t)}{\delta \underline{x}_0} \Delta \underline{x}_0 \equiv y_{\underline{x}_0}(t) \Delta \underline{x}_0$$

Obtaining the sensitivities $\underline{x}_{\underline{x}_0}$ and $y_{\underline{x}_0}$ by differentiating (above equation mark 1) with respect to $\underline{x}_0$ gives

$$\begin{aligned} \dot{\underline{x}}_{\underline{x}_0} &= \underline{f}_{\underline{x}}(t) \underline{x}_{\underline{x}_0} + \underline{f}_{\underline{y}}(t) y_{\underline{x}_0} \\ 0 &= g_{\underline{x}}(t) \underline{x}_{\underline{x}_0} + g_y(t) y_{\underline{x}_0} \end{aligned} \tag{2.7}$$

where $\underline{f}_{\underline{x}} \equiv \frac{\delta \underline{f}}{\delta \underline{x}}$, with the other time-varying Jacobian matrices following the same form.

The model parameters are estimated from the disturbance measurement by finding the best fit between the model output and the measurements with sequences of samples z, which is assumed to have sufficiently large number of samples to estimate the unknowns. The proper fitting of the model flow to the measurement will determine the appropriate values for $\underline{x}_0$, which contains parameters and initial conditions. Given the measurement samples and the corresponding algebraic state $\breve{y}(t)$, the parameter can be estimated using the Gauss-Newton approach. For estimating the parameter from a single measurement, the mismatch at each sample time is

$$e(\underline{x}_0) = \breve{y}(\underline{x}_0) - z \tag{2.8}$$

Where

$$e(\underline{x}_0) = \begin{bmatrix} e_0(\underline{x}_0) \\ e_1(\underline{x}_0) \\ \vdots \\ e_q(\underline{x}_0) \end{bmatrix}, \quad \breve{y}(\underline{x}_0) = \begin{bmatrix} \breve{y}_0(\underline{x}_0) \\ \breve{y}_1(\underline{x}_0) \\ \vdots \\ \breve{y}_q(\underline{x}_0) \end{bmatrix}, \quad z = \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_q \end{bmatrix} \tag{2.9}$$

Finding $\underline{x}_0$ that minimizes the least squares cost

$$J(\underline{x}_0) = \frac{1}{2} \sum_{k=0}^{q} \left| e_k(\underline{x}_0) \right|^2 = \frac{1}{2} e(\underline{x}_0)^t e(\underline{x}_0) = \frac{1}{2} \left\| e(\underline{x}_0) \right\|_2^2 \tag{2.10}$$

Which then reduces it to a nonlinear least squares formulation solvable using Gauss-Newton, which is based on linearizing $e(\underline{x}_0)$ around $\underline{x}_0^j$

$$\overline{e}(\underline{x}_0, \underline{x}_0^j) = e(\underline{x}_0^j) + \frac{\delta e(\underline{x}_0^j)}{\delta \underline{x}_0}(\underline{x}_0 - \underline{x}_0^j) \tag{2.11}$$

Where

$$\frac{\boldsymbol{\delta e(\underline{x}_0^j)}}{\boldsymbol{\delta \underline{x}_0}} = \frac{\boldsymbol{\delta \breve{y}(\underline{x}_0^j)}}{\boldsymbol{\delta \underline{x}_0}} = \begin{bmatrix} \breve{\boldsymbol{y}}_{\underline{\boldsymbol{x}}_0}(\underline{\boldsymbol{x}}_0^{\boldsymbol{j}}, \boldsymbol{t_0}) \\ \breve{\boldsymbol{y}}_{\underline{\boldsymbol{x}}_0}(\underline{\boldsymbol{x}}_0^{\boldsymbol{j}}, \boldsymbol{t_1}) \\ \vdots \\ \breve{\boldsymbol{y}}_{\underline{\boldsymbol{x}}_0}(\underline{\boldsymbol{x}}_0^{\boldsymbol{j}}, \boldsymbol{t_q}) \end{bmatrix} \tag{2.12}$$

Which is basically a matrix of trajectory sensitivities at different time steps. At each iteration, the value of $\underline{x}_0$ at which $\frac{1}{2}\left\|e(\underline{x}_0, \underline{x}_0^j)\right\|_2^2$ is minimum is solved.

2.2.3 Other Notable Disturbance-Based Validation Methods

Other studies have used more sophisticated methods for identifying load parameters, such as that in [26], which is a measurement-based identification method using genetic algorithm (GA) and the Levenberg-Marquardt (L-M) algorithm. The combination of the algorithms takes advantage of GA's global optimum search ability and L-M's local optimum search ability. A notable advantage of the merged algorithm for optimization is the significant reduction in computation time, compared to a simple GA. The accuracy obtained by the combined method is comparable to that of L-M, with some resulting estimates closer to the actual value than L-M, but the overall RMS error shows the combined GA and L-M algorithm to have lower errors for all tests. A similar technique, which employs sensitivity analysis and Particle Swarm Optimization (PSO), was studied in [27]. Other novel approaches, such as estimation of parameters using stochastic approximation [28], recursive non-linear least square optimization [29, 30], Synthesized Information Factor (SIF) [31, 32], and derivation using induced small disturbances [33] have been explored in previous studies.

3 Random Forest

The validation method evaluated in this study seeks to draw patterns from the dynamic response of a network based on changes made in the model parameters involved. The characterization of the dynamic measurements, which normally consists of hundreds of samples, takes the most relevant features from the recording, but hardly reduces the readings to what can be practically analyzed by simple inspection. Furthermore, since the model error classifiers that the study aims to develop varies from one power network or event to another, the tools for analysis should be very adaptable, and at the same time capable of delivering the results in the least amount of time possible. It is for this reason that a machine learning technique was sought.

The Random Forest (RF), which is a tree ensemble-based classification and regression algorithm, was selected as it is highly regarded for its accuracy [34, 35]. Moreover, it is known to be capable of working on high-dimensional problems directly, "fast to train and predict", and can be implemented using parallel computation [36]. These properties make it ideal for the development of the validation method, which will be described in Chapter 5.

In power systems, RF has been used for applications such as predicting system conditions [34, 37, 38], identifying faults in wind turbines [39], and in FACTS-based transmission lines [40]. In all the studies cited above, the RF algorithm has excelled in accuracy. It has also been noted for its speed, making it more attractive for protection applications, compared to other machine learning algorithms.

As with other tree-based methods, it can perform both classification and regression, although the following discussion will be restricted to classification as it is the only relevant method to the study.

The process of classification essentially attempts to uncover patterns relating a set of observations to a target set of classes, which may describe a condition, state or an outcome. In general, given a random vector described by $X = \left(X_1 \ldots . X_p\right)^T$, representing the set predictor variables, and the random variable Y, as the response, the objective is to find a prediction function

$f(X)$ which approximates Y. This is accomplished by minimizing an applicable loss function $(Y, f(X))$ [36]. The loss function for classification can be described by

$$L\big(Y, f(X)\big) = I\big(Y \neq f(X)\big) = \begin{cases} 0 \; if \; Y = f(X) \\ 1 \; otherwise \end{cases} \tag{3.1}$$

Minimizing the expectation of the loss function, given by $E_{XY}\left(L\big(Y, f(X)\big)\right)$ gives

$$f(x) = arg \max_{y \in \mathcal{Y}} P(Y = y | X = x). \tag{3.2}$$

Ensemble machine learning methods, on the other hand, consists of a collection of base learners, represented by $h_1(x) \ldots . h_j(x)$, which for RF are the individual trees. The group of decision trees forms $f(x)$, which then transforms the classification function into

$$f(x) = arg \max_{y \in \mathcal{Y}} \sum_{j=1}^{J} I(y = h_j(x)) \, , \tag{3.3}$$

which indicates that ensemble learners classify according to a majority vote. The j^{th} base learner in RF is specifically denoted by $h_j(X, \Theta_j)$, where Θ_j represents the independent random variables for each tree. The randomness introduced to the ensemble represents the key features of the RF algorithm, which will be discussed in Section 3.2. Before continuing into the details of the RF algorithm, it is important to understand how individual tree learners are formed, and how Random Forest builds on the decision trees with a few modifications.

3.1 Background: Decision Trees [41]

The process of classification in Random Forests (RF) is a result of combining the classification outputs from multiple decision trees. A single unit of decision tree is generated from a data set, which contains multiple instances of random observations with selected attributes, and the corresponding class. A tree is grown by splitting on the attributes recursively until the predictor spaces have been optimally partitioned, where each branch of the tree leads to a single class at the terminal node.

The generated tree ideally represents the compressed form of the training data, where the discovered patterns are summarized in terms of a series of simple conditional statements. Each conditional statement forms a node in the tree, the result of the statement form the branches, that cause the splitting of the predictor space into two (Figure 3-1). A branch may continue into another partitioning statement, or stop once the desired purity of the class is reached. Aside from data compression, decision trees are more frequently constructed to serve as a prediction function for data with an undetermined class.

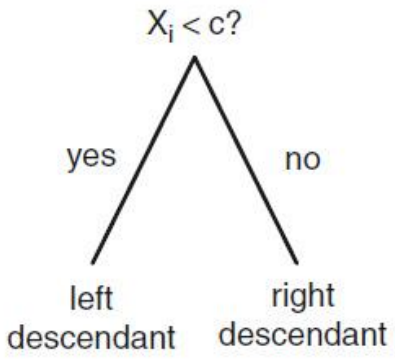

Figure 3-1. Node splitting into branches [36]

Decision trees are fundamentally based on what is called the Top-Down Induction of Decision Trees (TDIDT). The decision rules, in the form of conditional statements, originate from a data set containing several observations with a group of attributes, where each attribute is a potential candidate for splitting. The TDIDT algorithm can split an attribute no more than once. Therefore, at maximum, a branch can only grow up to length M, where M is the number of attributes in an instance. To ensure consistency of the decision tree, it is necessary for the training set to satisfy the *adequacy condition*, which states that "no two instances with the same values of all the attributes may belong to different classes" [41].

A. Splitting the Nodes

The TDIDT algorithm itself however, does not specify any guidelines for how the splitting of attributes should proceed. As it is, the decision trees generated would not be of any value. To create a more meaningful classifier, methods for attribute splitting have been developed. One such method is called *entropy*, where the best split is based on maximizing what is called the *information gain*. The measure of entropy is given by the equation

$$E = -\sum_{i=1}^{K} p_i \log_2 p_i \,, \tag{3.4}$$

where K is the number of classes, and p_i is the proportion of class i out of all instances in the set. The value of entropy from node to node from all prospective splits is compared, and the attribute split resulting in the biggest average drop in entropy, or increase in information gain, will be selected as the best split. The information gain computed from this method tends to have a bias on categorical attributes with numerous possible values, especially those that have uniformly distributed instances for an attribute. To solve this, a measure called *gain ratio* is derived from the attribute split information. The split information is taken from the sum of instances s_j for each attribute value for a given attribute split

$$Split\ Information = -\sum_{j=1}^{V} \frac{s_j}{N_j} \log_2 \frac{s_j}{N_j}, \tag{3.5}$$

where N_j is the total number of attributes in the attribute subset. The gain ratio is computed as

$$Gain\ Ratio = \frac{Information\ Gain}{Split\ Information}. \tag{3.6}$$

Another possible measure for splitting is called the *Gini Index of Diversity* given by

$$Gini = 1 - \sum_{i=1}^{K} p_i^2. \tag{3.7}$$

In terms of frequency f_{ij}, which indicates the number of instances belonging to class i of the j^{th} attribute, the Gini index for the subset's split on the selected attribute is given as

$$Gini_{new} = 1 - \left(\frac{1}{N}\right) \sum_{j=1}^{V} \left(\frac{1}{N_j}\right) \sum_{i=1}^{K} f_{ij}^2 \tag{3.8}$$

Similar to entropy, the best split is determined by the attribute whose split results to the greatest reduction of the average Gini index.

B. Continuous Value Discretization

Attributes used for generating classifiers can either be categorical, which have a finite set of possible values, or continuous, as most numerical properties assume. For decision trees to process continuous attributes, the values are discretized or broken down into a finite number of groups. The process of discretization may be applied locally or globally. For local discretization, the continuous attributes are converted at every step of the node splitting. Global discretization on the other hand, converts all continuous attributes in a training set once, independent of what data mining algorithm it will be used for.

Local discretization arranges the given values of a continuous attribute, together with the corresponding class frequencies, in ascending order and partitions the range of values into two parts, one at a time, with the available values as cut points, creating several pseudo-attributes. Each pseudo-attribute's information gain or gain ratio is computed, which in turn can be used for comparison against categorical attributes.

For global discretization, like the previous method, continuous attribute values with classification frequencies are arranged in an ascending order. After which, a statistical test called the ChiMerge algorithm is applied on two adjacent rows one at a time. The algorithm tests the hypothesis that the class is independent of the intervals, or the adjacent rows, an instance is found under. If the hypothesis is satisfied, the rows are merged, otherwise, they are left separate.

C. Pruning

For any given training data, the TDIDT algorithm by itself can generate decision rules that maximizes the growth of the tree based on the number of attributes available. This may work very well for compressing information of the training set, but may do poorly for predicting classes for all unseen data. Overfitting results from overspecialized rules, which is characterized by inclusion of attribute criteria that may be caused by noise or irrelevant information in the training set. This is remedied by reducing the decision rules down to the most significant terms for generalization, which is otherwise known as *pruning*. The pruning process can be done while the tree is being grown, by setting a threshold on the number of instances in a subset or the depth of a branch. Pruning can also be applied after the tree is grown, which involves measuring the significance of the branches with respect to its immediate parent or root node.

D. Limitations

The decision tree algorithm, while it has been used successfully in a variety of applications, has some inherent limitations. One such problem is the possibility of a node containing two or more correlated attributes. These attributes, otherwise known as surrogates, can result in identical node splitting. Interpretability, which is one of the decision tree's favorable properties, can be difficult in such cases, especially in high-dimensional data sets. Furthermore, it has been noted that decision trees are not robust against small shifts in the training data. Slight perturbations in the data can potentially cause drastic changes in the structure of the tree, which result in variability in the predictions [36, 37]. More importantly, several algorithms based on the decision tree have been developed and proven to be more accurate, such as the Random Forest [34].

3.2 Tree Ensemble: The Random Forest Algorithm [35, 36, 42]

For a given training data denoted by $\mathcal{D} = \{(x_1, y_1) \ldots . (x_N, y_N)\}$, where x_i is a realization of predictors, and y_i is the corresponding response, the base learner tree in RF can be described by $\hat{h}_j(x, \theta_j, \mathcal{D})$, where θ_j is a realization of the random variable. The randomness applied to the base learner trees, represented in θ_j, is done in two ways.

1. By fitting each tree to an independent bootstrap sample from the given training data, which basically takes resamples from the training set to build each base learner in the forest. So for training set consisting of N instances, N instances are sampled at random, with replacement, to grow the individual trees.
2. Taking a random subset m from the collection of M predictors to split on, independently for each node, when generating the tree. Note that $m \ll M$, and m is held constant for the whole tree-growing process.

The randomization added to the generation of the ensemble minimizes correlation between any two trees, and at the same time increases the strength, or classification accuracy, of each tree. Normally, a simple reduction in m results in reduction of both correlation and strength, and an increase causes both to increase. On top of the error rate reduction due to the randomization features of the algorithm, as a result, RF is also known to be robust against outliers and noise.

3.2.1 Estimation of generalization error

In the process of bootstrapping the training data to build the base learners for RF, some instances will be unused. These records, called out-of-bag data, can be used for estimating the generalization error and variable importance. The predictions for error estimation are obtained by using out-of-bag observations with respect to a base learner, or the instances not picked in the resampling process to generate the tree. For classification, the out-of-bag error rate is given by

$$E_{OOB} = \frac{1}{N}\sum_{i=1}^{N} I\left(y_i \neq \hat{f}_{oob}(x_i)\right), \tag{3.9}$$

where $\hat{f}_{oob}(x_i)$ is the out-of-bag prediction for the i^{th} observation. The out-of-bag prediction errors are then set by class by cross-tabulating y_i and $\hat{f}_{oob}(x_i)$.

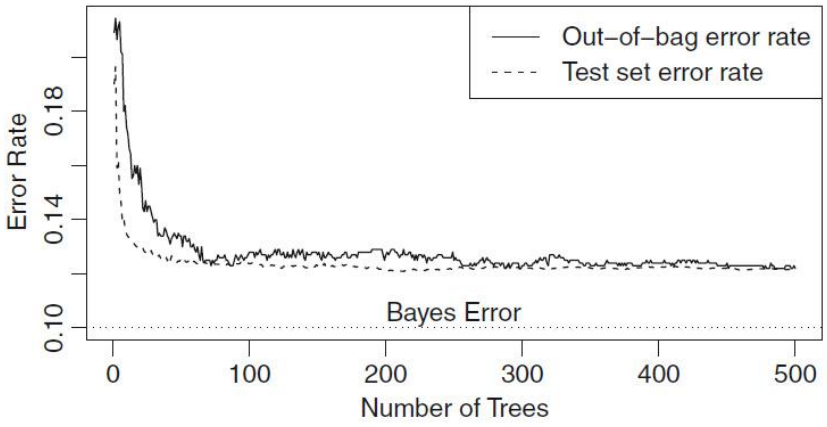

Figure 3-2. Out-of-bag and test set error rate comparison [36]

About a third of the instances in the bootstrap training set are left out. And while the test error rate decreases and converges to a certain rate as the number of trees grown increases (Figure 3-2), the out-of-bag error usually overestimates this error rate. Unbiased error estimates are said to be obtained past the point of test set error rate convergence.

3.2.2 Tuning Random Forest

There are three tunable parameters in RF that can be used to possibly improve the accuracy of the prediction. These are

1. The number of predictor variables m used for splitting at each node.
2. The number of trees in the forest J.
3. The size of the trees in terms of the smallest node size or the maximum number of terminal nodes.

For a given number of predictors M in a data set, the standard default size for splitting a node in RF is $m = \sqrt{M}$. It can be adjusted based on the out-of-bag error rate, which consequently would give a biased estimated of the generalization error. Random forests however are typically not sensitive to m, and will thus only slightly affect the prediction.

For the selection of the number of trees J, it has been established that the greater number of trees, the smaller the generalization error. Unlike many ensemble methods, there is no risk of J getting too big and suffer overfitting. As the number of trees increase, it has been shown that RF appears to converge to a certain generalization error limit. It seems, that it is favorable to grow as many trees as it is practically and computationally possible. Consequently, it can be seen in Figure 3-2 that the out-of-bag error, which closely follows the generalization error, can be used as basis for sizing the number of trees. The visualization can map the section where the out-of-bag error begins to stabilize, which will enable the selection of the least number of trees at the best accuracy.

It was initially thought that RF will not have overfitting problems with growing very large trees, though studies have indicated otherwise. In cases where large trees overfit, the number of nodes can be tweaked to possibly improve accuracy. The out-of-bag error information can be used as reference for tuning the parameter, but can again cause bias in the estimated generalization error.

3.2.3 Variable Selection

Selection of the most important predictor variables in Random Forest to reduce dimensionality and improve interpretability of the classifier can be done using principal component analysis (PCA). However it is also possible to miss important information for prediction in the process. As an alternative, the variable importance can be measured from the Random Forest itself using out-of-bag information. This is done by selecting a single predictor variable k from the out-of-bag data set, and randomly permuting its values with the other variables fixed before passing the set into the trees. The error rate of the predictions in the original data will

then be compared to that of the permuted data set. An increase in the error rate can then be interpreted as an indication of the variable's importance in the prediction process. The process of permutation allows identification of all important predictor variables, including those that are highly-correlated.

3.2.4 Proximities and Imputing Missing Values

The measure of proximity is the number of times that any two observations fall in the same terminal node over the total number of trees in the forest. A proximity of 1 indicates the highest proximity and 0 the lowest. This is used as an indication of the closeness of observations in the predictor space, which may also translate into the Euclidean space. However, the strength of predictors also factor into the measure of proximity. For example, two observations may appear close in the Euclidean space, but if they greatly differ in the most important predictors, they proximities will turn out smaller, and vice versa.

Using proximity, missing values in observations can be estimated, or imputed, through an iterative process. A forest is initially grown with median estimates, where the proximities are subsequently checked, and new imputations are made using a "proximity-weighted average for continuous predictors or a proximity-weighted vote for a categorical predictor". A new forest is grown, proximities are checked, and estimates are made, repeatedly, until the process arrives at stable imputations.

3.2.5 Random Forest Example

A sample dataset from Weka, called *Image Segmentation Data* under the filename *segment-challenge*, has been used for the following experiment. This data was used for an image recognition test, where the images were randomly taken from 7 outdoor images and "handsegmented to create a classification for every pixel" [43], and each segment is a 3x3 region. The dataset contains 19 continuous attributes and 1500 instances. The classes defined were: **brickface**, **sky**, **foliage**, **cement**, **window**, **path**, **grass**.

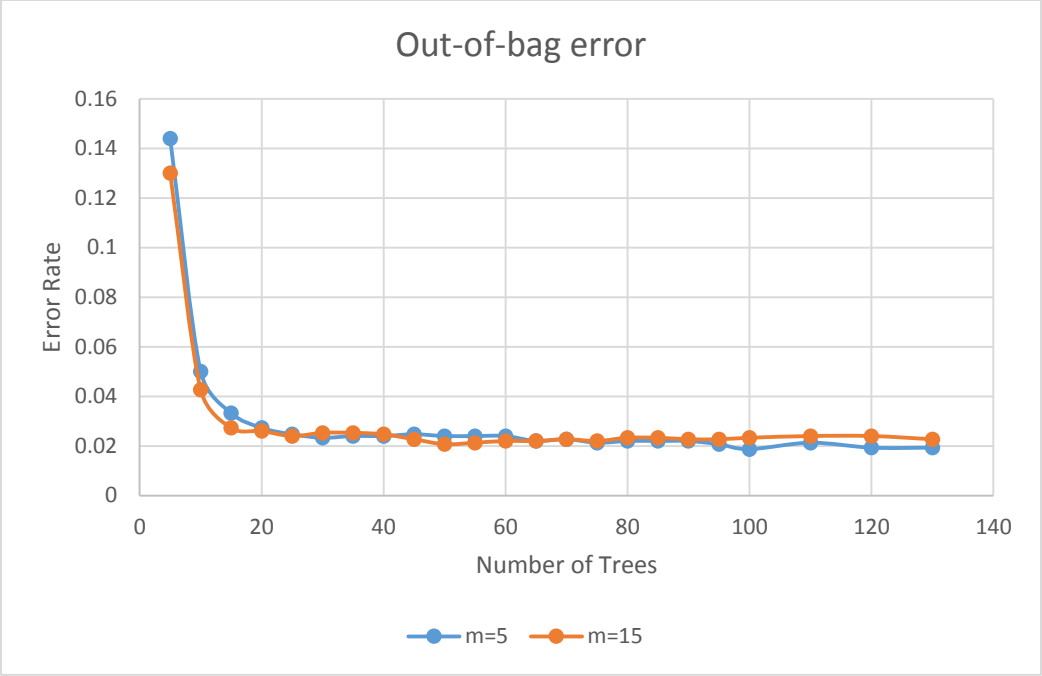

Figure 3-3. Comparison of RF with two different number of attributes split at each node

The variation of the out-of-bag error with increasing number of trees was evaluated for the dataset. For this test, the performance of RF for two ensembles with different number of attributes split at each tree node were compared. One was using the default number of attributes which is $\log_2 M + 1$, where M is the total number of attributes in the dataset. In this case, it would have $m = 4.25 + 1 = 5.25 \approx 5$. The other ensemble was set to have 15, which is just 4 attributes shy of the total number of attributes.

The two curves, in general, had approximately equivalent rates throughout, with the error rate settling at around 2%. Initially, $m = 15$ performed stronger, with lower rates at the first few points compared to the default, but at the end, the ensemble with the default setting registered slightly lower error rates. Overall, the graph hardly shows any difference in execution for the two ensembles. However, the glaring difference lies in the amount of time it needed to grow the forests. At 130 trees, the default ensemble took 2.13 seconds to build the model, while for $m = 15$ it was 5.38 seconds. A 10-fold cross-validation method was used, so each duration is multiplied 10 times, making the disparity more apparent. The result shows that an ensemble with a reduced selection of attributes for splitting not only is capable of matching that with a bigger subset in terms of error rate, but it can surpass its performance in speed.

Table 3-1. Dataset Confusion Matrix

=== Confusion Matrix ===							
a	b	c	d	e	f	g	<-- classified as
202	0	0	1	2	0	0	a = brickface
0	220	0	0	0	0	0	b = sky
0	1	200	3	4	0	0	c = foliage
0	0	2	209	8	1	0	d = cement
1	0	9	6	188	0	0	e = window
0	0	0	1	0	235	0	f = path
0	0	0	0	0	0	207	g = grass

The result of the classification is summarized in what is called a confusion matrix. The table indicates the classes at the rightmost column, and how the observations were classified as, designated by the top row. More details about measures of accuracy can be seen in Chapter 6. The matrix shown is from the random forest with 100 trees with the default number of attributes split. It can be seen that most of the observations have been properly classified. The classes **cement** and **window** had the most misclassification with respect to the number of classes in each class. The class **window**, for example has a sizable number of instances misclassified as **foliage** and **cement**.

4 Prony analysis

In 1795, Gaspard Riche, Baron de Prony developed a method of representing the gas expansion behavior of gases in terms of sums of damped exponentials. The method he proposed interpolates data between equally spaced measured points by first fitting damped exponentials on the recorded samples, then computing for the intermediate points by evaluating the exponential model. This method of modeling data as linear combination of exponentials has evolved from exactly fitting damped.

For a given set of N data samples $x[1], \dots, x[N]$, the Prony method estimates $x[n]$ using a p-term exponential model given by

$$\hat{x}[n] = \sum_{k=1}^{p} A_k \exp[(\alpha_k + j2\pi f_k)(n-1)T + j\theta_k] \qquad 1 \le n \le N, \tag{4.1}$$

where T is the time between samples in seconds, A_k the amplitude, α_k the damping factor in $seconds^{-1}$, f_k the frequency in Hz, and θ_k the sinusoidal phase in radians. For real data samples however, the exponentials should exist in complex conjugate pairs, which will then transform the expression into

$$\hat{x}[n] = \sum_{k=1}^{p/2} 2A_k \exp[\alpha_k(n-1)T] \cos[2\pi f_k(n-1)T + j\theta_k] \qquad 1 \le n \le N. \tag{4.2}$$

In power systems, Prony analysis has been widely used for analyzing oscillations [44-47], and characterizing systems based on their responses, from which models can be derived [48-53]. In addition, Prony has also been used for protection applications [54, 55], and tracking dynamics of measurements [56-58]. Its exceptional performance in the preceding applications makes it a favorable candidate for model validation.

4.1 Original Method [59]

If the number of exponential parameters are as many as the data samples, an exact fitting can be made. Equation (4.1) can be written as

$$x[n] = \sum_{k=1}^{p} h_k z_k^{n-1}, \tag{4.3}$$

where

$$h_k = A_k \exp(j\theta_k)$$
$$z_k = \exp[(\alpha_k + j2\pi f_k)T)]. \tag{4.4}$$

With the number of samples equal to the number of complex parameters in the exponentials, the expression for estimate $\hat{x}[n]$ can be replaced with $x[n]$. Equation (4.3) can be rewritten in matrix form

$$\begin{bmatrix} z_1^0 & z_2^0 & \cdots & z_p^0 \\ z_1^1 & z_2^1 & \cdots & z_p^1 \\ \vdots & \vdots & & \vdots \\ z_1^{p-1} & z_2^{p-1} & \cdots & z_p^{p-1} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_p \end{bmatrix} = \begin{bmatrix} x[1] \\ x[2] \\ \vdots \\ x[p] \end{bmatrix} \tag{4.5}$$

In order to solve this, it is important to note that equation (4.3) is a solution to a homogeneous linear constant-coefficient difference equation. Defining a polynomial $\phi(z)$ containing z_k as roots

$$\phi(z) = \prod_{k=1}^{p} (z - z_k), \tag{4.6}$$

and expressing (4.6) in summation form by expansion into a power series

$$\phi(z) = \sum_{m=0}^{p} a[m] z^{p-m}, \tag{4.7}$$

which now contains complex coefficients $a[m]$ such that $a[0]$. Applying (4.7) to the expression at (4.1) will ultimately yield the $p \times p$ matrix that satisfies the generated relation, given by

$$\begin{bmatrix} x[p] & x[p-1] & \cdots & x[1] \\ x[p+1] & x[p] & \cdots & x[2] \\ \vdots & \vdots & & \vdots \\ x[2p-1] & x[2p-2] & \cdots & x[p] \end{bmatrix} \begin{bmatrix} a[1] \\ a[2] \\ \vdots \\ a[p] \end{bmatrix} = - \begin{bmatrix} x[p+1] \\ x[p+2] \\ \vdots \\ x[2p] \end{bmatrix} \tag{4.8}$$

The matrix of data samples above can now be solved to obtain the complex polynomial coefficients, given which, can solve for the roots z_i in (4.5). From this the damping and the sinusoidal frequency can be solved

$$\alpha_i = \frac{\ln|z_i|}{T} \quad second^{-1} \tag{4.9}$$

$$f_i = \tan^{-1}\frac{\left[\frac{\Im\{z_i\}}{\Re\{z_i\}}\right]}{2\pi T} \quad Hz$$

With z_i known, the h_i parameters can be solved, which will give the amplitude and phase components

$$\begin{aligned} A_i &= |h_i| \\ \theta_i &= \tan^{-1}\frac{\Im\{z_i\}}{\Re\{z_i\}} \quad radians \end{aligned} \tag{4.10}$$

Typically, the number of data samples exceed the required number of exponentials to be fitted. Such cases cannot be solved by the original method. Consequently, several procedures have been developed to handle overdetermined cases, among which involve the least squares technique.

4.2 Prony by Parametric Modeling [60]

In this study, the Prony decomposition is performed by using a parametric modeling method, used for designing infinite impulse response (IIR) filters. This process involves estimating the poles and zeroes of a transfer function of an IIR filter, whose impulse response matches that of the sequence of data samples. The IIR filter transfer function is given as

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}}, \tag{4.11}$$

where the impulse response $h[n]$ in the z-domain is given by

$$H(z) = \sum_{n=0}^{\infty} h[n] z^{-n} \tag{4.12}$$

Using the $K + 1$ terms of the impulse response, the IIR filter transfer function can be convolved in the z-domain as $B(z) = H(z)A(z)$, giving the matrix

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_M \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} h_0 & 0 & 0 & \cdots & 0 \\ h_1 & h_0 & 0 & & \\ h_2 & h_1 & h_0 & & \\ \vdots & & & & \vdots \\ h_M & & & & \\ & & & & \\ \vdots & & & & \\ h_K & & \cdots & & h_{K-N} \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix} \tag{4.13}$$

To solve for the coefficients, the matrix can be partitioned as

$$\begin{bmatrix} \boldsymbol{b} \\ - - \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} H_1 \\ - - - - - \\ \boldsymbol{h_1} \quad | \quad H_2 \end{bmatrix} \begin{bmatrix} 1 \\ - - \\ \boldsymbol{a}^* \end{bmatrix} \tag{4.14}$$

where

- $\boldsymbol{b}$ is the vector of M+1 numerator coefficients
- $\boldsymbol{a}^*$ is the vector of N denominator coefficients
- $\boldsymbol{h_1}$ is a the vector of the last $K - M$ terms of the impulse response
- H_1 is the $(M + 1) \times (N + 1)$ partition and
- H_2 is the remaining $(K - M) \times N$ partition of the impulse response

The vector $\boldsymbol{a}^*$ coefficients can then be solved from

$$\mathbf{0} = \boldsymbol{h_1} + H_2 \boldsymbol{a}^* \tag{4.15}$$

After which, the numerator coefficients given by $\boldsymbol{b}$ can be solved using

$$\boldsymbol{b} = H_1 \boldsymbol{a} \tag{4.16}$$

For $K = M + N$, H_2 will be square, from which $\boldsymbol{a}$ and $\boldsymbol{b}$ can be solved if H_2 is not singular. In this case the number of unknowns can be matched with the number of impulse response terms. If H_2 is singular, (4.15) will have many solutions, and $h[n]$ can be generated by a lower order system.

Once all the coefficients are solved, the poles p_i and residues r_i of the transfer function are solved by partial fraction expansion. The decaying sinusoids' components can be solved as

$$\begin{aligned} \alpha_i &= \Re(p_i) \\ f_i &= \frac{\Im(p_i)}{2\pi} \\ A_i &= |r_i| \\ \phi_i &= \angle r_i \end{aligned} \tag{4.17}$$

4.3 Prony Example

The following data was taken from the application Prony Toolbox, which can be found in the Matlab Central File Exchange database. The data is filed under the database table name '*pure_demo*'. This example demonstrates the decomposition of signals into decaying sinusoids, and how the Prony order number affects the estimation of the signal. The code used for the example can be found in Appendix C.3.

Given the signal (Figure 4-1)

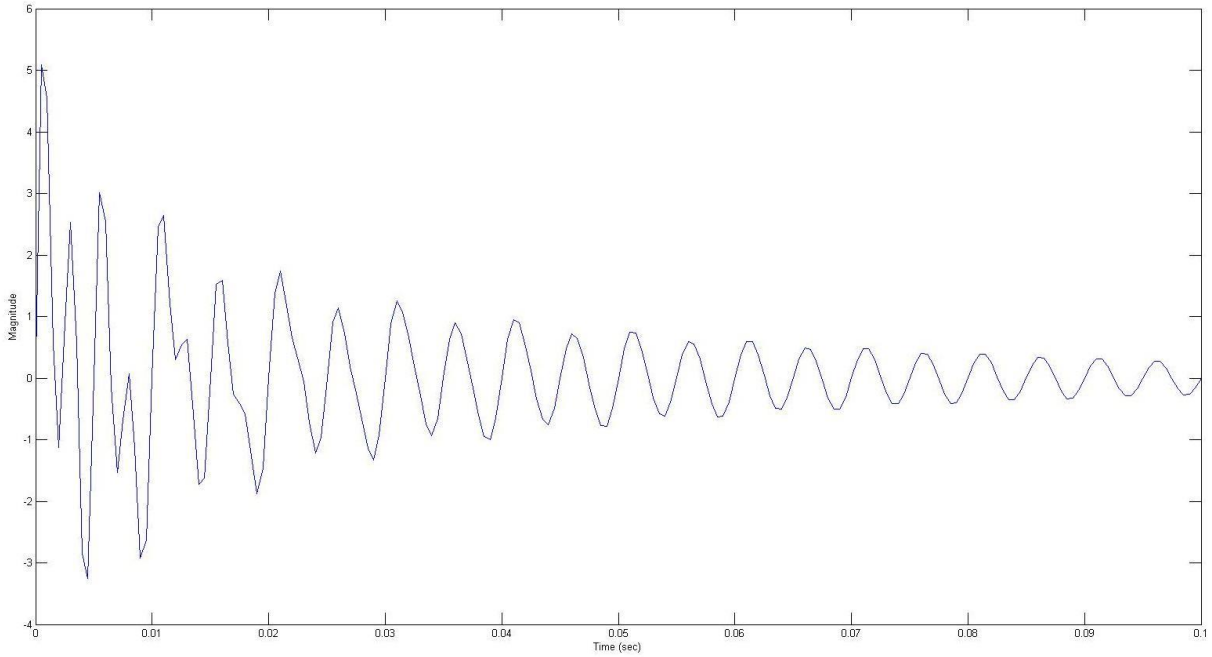

Figure 4-1. Input Signal

Prony can synthesize the signal by resolving the samples into decaying sinusoids. For an order 6 decomposition, we will get (Figure 4-2)

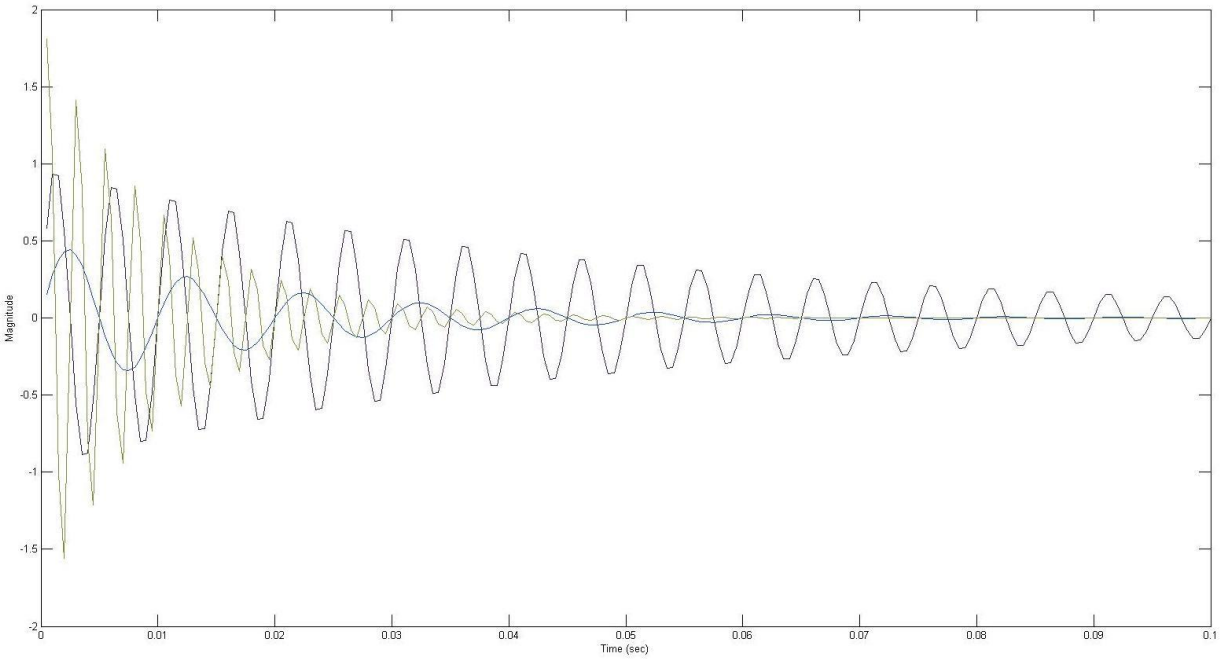

Figure 4-2. Signal decomposition using an order 6 estimate

Recomposing the decaying sinusoids will demonstrate Prony's estimate of the signal (Figure 4-3)

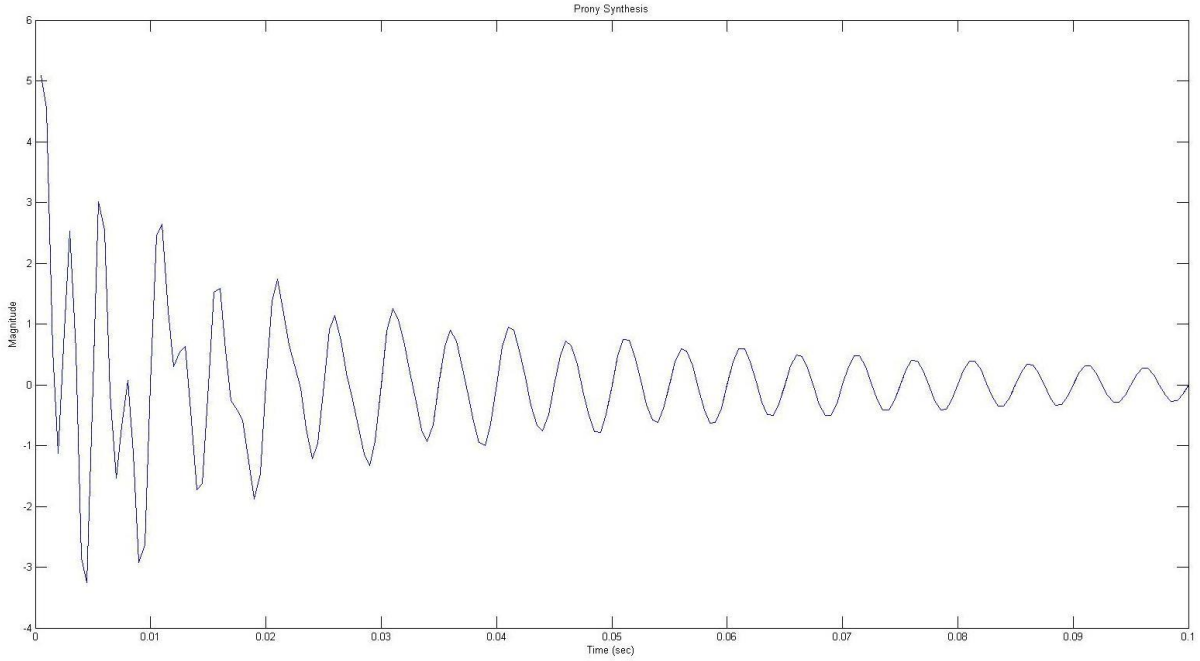

Figure 4-3. Input signal compared with the recomposed decaying sinusoids at order 6

The estimate and the input signal display a good fit for an order 6 decomposition. To illustrate an imperfect decomposition, an order 4 Prony synthesis was run, yielding (Figure 4-4)

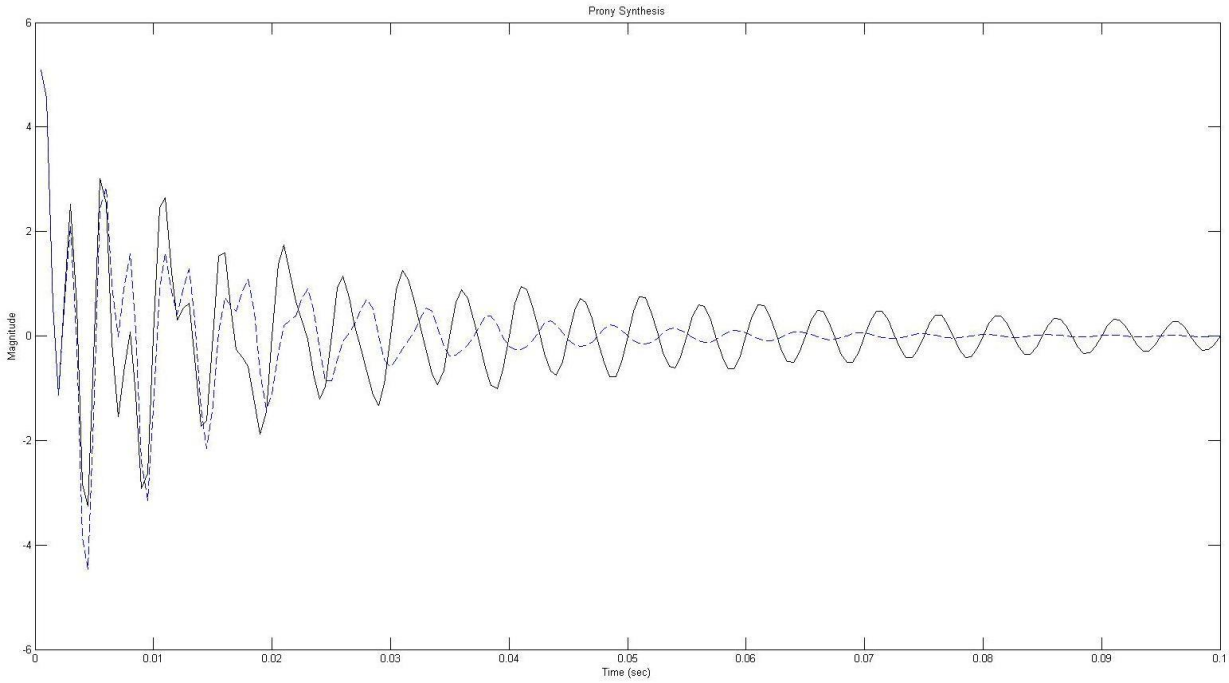

Figure 4-4. Input signal (solid line) compared with the recomposed decaying sinusoids (dashed line) at order 4

Where a clear mismatch can be seen between the Prony estimate and the original signal. The order 4 decomposition can be seen in (Figure 4-5).

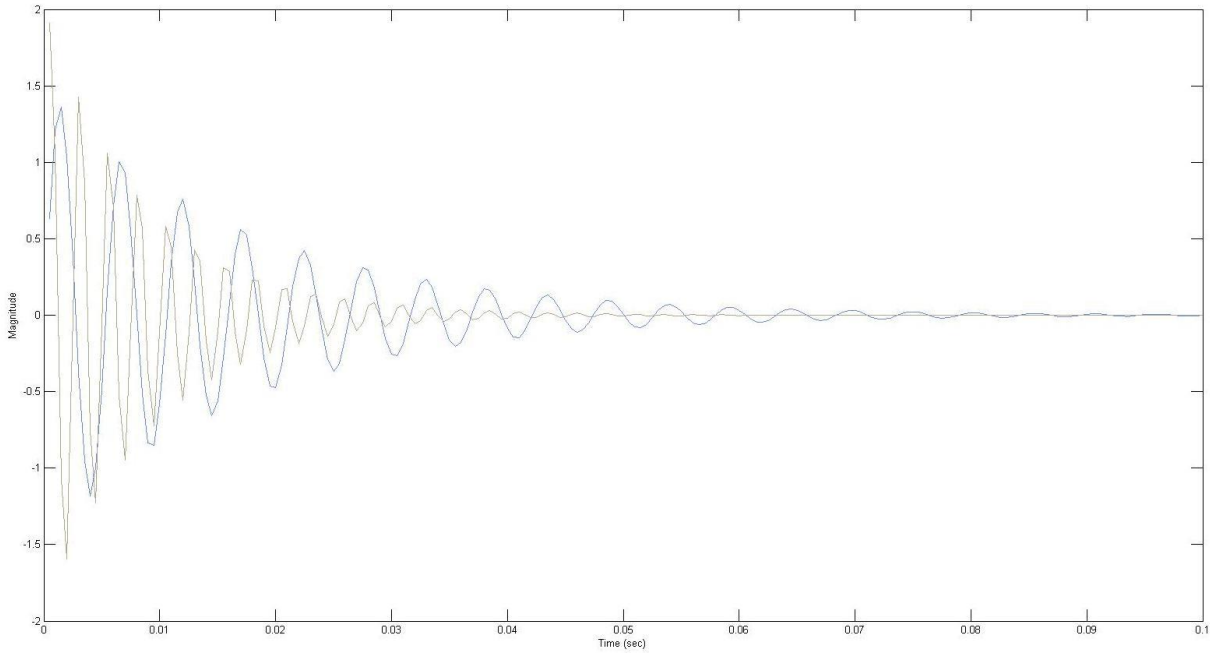

Figure 4-5. Signal decomposition using an order 4 estimate

5 Methodology

Identification of power system simulation model errors can be an arduous task based on the typical scale of networks, and the variety of components connected and interacting in response to various dynamic events. Model parameters are set usually according to a manufacturer's specification, but offline tests on the actual components can also be conducted prior to commissioning, to verify the settings [3], or to make modifications based on the system's requirements. However, even with rigorous evaluation, the devices' known settings on the network could slowly drift, perhaps due to the long-term exposure to variations in temperature, humidity, and other factors that introduce stress and strain to the materials. Without regular scheduled equipment testing, it is highly likely to have outdated model settings that will poorly capture a system's response to present events.

Onsite validation of equipment can very accurately determine parameter values, but pulling power system components out of service can be an impractical and costly option. Non-intrusive, computer model-based validation methods normally estimate simulation errors using techniques involving waveform matching, by taking simulated outputs against field measurements. The optimal match for the measurement dynamics will yield the supposed correct model settings. This approach is typically very mathematically involved, and will require the full set of dynamic equations for validation. The data mining-based validation method explored in this research uses a more simplified approach, and will allow selective testing of parameters that need to be corrected. Moreover, it can theoretically work without knowing the intricacies of the models under test. However, since this is only a classification process, it will not suggest the amount of adjustment needed for eliminating the error. It will instead single out the parameters requiring correction, and indicate whether the tuning required is an increase or decrease in quantity.

5.1 Model Validation Using Classification

The idea behind validation through classification is to discretize all the possible corrections into a more manageable group of classes. This means a specific error will be assigned to a certain

class, along with other errors classified as such. For this study, the partitioning of the error is intuitively arranged according to its relative position in a range of values for a parameter.

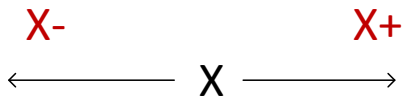

Figure 5-1. Parameter classification range.

For a given model default or base parameter **X**, there are at most two possible error ranges. One is the range greater than the base, denoted by **X+**, and the other less than the base, **X-** (Figure 5-1). It has to be ensured that the ranges will not exceed the allowable minimum or maximum value for the variable. The extent to which a parameter can be modified is projected to correspond to a dynamic change in the model output, and anything in excess of the allowable range can perhaps reduce classification accuracy. Also, overshooting the range may generate impossible model responses, which is a waste of simulation and training time.

The classifier is designed to work in a similar manner as a stroboscopic tuner. A strobe tuner is a musical instrument tuner that zeroes in on the closest vibrating note, and indicates whether an increase or decrease in pitch is required to get the precise tuning, shown by a rotating display. When the note is held in tune, the display is static. For the classifier designed to validate a single parameter error, the classifier simply picks the error type out of all the selection of variables, then, at the same time, tells the direction of the tuning required.

Using classification for the model validation process simplifies the location of error, as it reduces errors into targets consisting of a small group of classes. Once the suspected errors in the network are identified, the precise correction can be easily revealed, as the working range will be known.

The classifier generated will be specific to a single power system event. The process of validation for one disturbance is not expected to perfectly tune all the models under test. It is understood that the variety of parameters in models have distinct responses to different types of perturbations, and that they have varying orders of magnitude. Thus one event may fine tune a certain parameter **X**, and another event may properly correct another parameter **Y**. A perfectly tuned simulation model will perhaps only be achieved after several validation iterations. Although

the frequency of disturbance needed to attain a complete validation cannot be expected, the incremental corrections on models is still reasonable as opposed to no remedial actions taken at all.

5.2 Building the Classifier

To build the classifier, the simulation model must first accurately emulate the network configuration and conditions prior to the subject event. The models in question should be prepared according to the known running settings. In addition, the sequence of events should be initiated and spaced accordingly. For each complete simulation of the event, a deliberate random error is applied to the dynamic parameter under test. The error will be well within the recommended class range of the parameter. The expectation for this process is that the multiple iterations under one assigned class will have distinct and detectable properties in the dynamic response that can be used for classification. For this type of data generation, it is thus recommended to use a simulation application that is capable of automatically randomizing dynamic parameters on every run.

The basic requirements for a power system analysis software to generate the necessary data set for a data mining-based parameter validation are the following:

1. The application should be capable of running an unbalanced three-phase dynamic simulation, which makes all the sequence components available.
2. The model parameters should be accessible and mutable through a user-defined script or program to allow automated dynamic parameter variation.
3. The simulation output should be in an externally readable standard data format.

Typically, power system software suites only provide positive sequence data for dynamic simulations. Positive sequence data by itself will satisfactorily accomplish the needed task, but the study will be at a disadvantage as PMUs, which are here simulated, can provide other sequence information that can potentially be more useful. The second requirement is relatively difficult to satisfy. However, dynamic files such as those in PSS/E or TSAT[3] are made available in a simple text-based format, which can be changed using external custom-made programs. The last

[3] PSS/E and TSAT are dynamic simulation applications. PSS/E is Power System Simulator for Engineering and TSAT is Transient Security Assessment Tool.

requirement is rarely a problem, as most power systems software suites ensure simulation data can be read and analyzed by a wide variety of software applications.

For this research, the software used was PowerFactory, version 15, developed by DIgSILENT GmbH. This power system analysis software satisfies all the three aforementioned requirements. Moreover, it is capable of integrating detailed transmission, distribution and industrial grid models which makes it attractive for model validation studies. The exhaustive amount of detail in components that can be assembled together will ensure that most parameters will be accounted for when analyzing the system's response to an event [61].

Furthermore, the software has its own programming language, simply called DIgSILENT Programming Language (DPL), which was designed to have a syntax very similar to the object-oriented language C++. The whole simulation suite, in fact, follows an object-oriented paradigm as all application modules, network components, simulation commands, virtual instrument panels et al, are programmatically accessible as unique objects with attributes and methods. This feature makes it very easy to accomplish the second requirement.

5.2.1 Dynamic Simulation

The most essential component of this study is generating the transients caused by the selected disturbances on the system. Since the short circuit events applied to the network are asymmetric, the simulation software must be capable of executing an unbalanced three-phase transient simulation. DIgSILENT PowerFactory was specifically selected for this task, not only for its capacity to run simulations with extensive network component detail, but also for the wide variety of options and flexibility of its dynamic simulation functions. The dynamic simulation was run using the unbalanced three-phase electromechanical transients. Electromechanical transient studies run the simulations using the passive, steady-state network model but with balanced or unbalanced network conditions.

Initialization of the dynamic simulations begins with calculation of the initial conditions of all network elements using load flow information. The software then utilizes an iterative method for solving AC and DC load flows simultaneously with the dynamic state variable integrals [62].

All the simulations needed in the study are executed automatically using DPL, with all the required settings and network events already in place.

The test network used in the simulation is a DIgSILENT tutorial four bus system with a double bus configuration set to run with a 60 Hz system frequency. All buses are connected to synchronous machines and general loads. One of the buses, marked S3 (Figure 5-2), is connected to a detailed load in the form of a radial distribution network containing standard asynchronous machine loads and a variety of general loads. Bus S1 is connected to a detailed model of a generating station consisting of two salient pole synchronous generators, an asynchronous machine load and a general load, all at 33 kV. The validation experiment will be applied to this station, particularly the synchronous generator PP1_G2. For this generating station, it will be assumed that a reference PMU, from which all measurements in the study are taken, is installed at S1, and the disturbance or fault is applied to one of the overhead lines linking the power plant to bus S1.

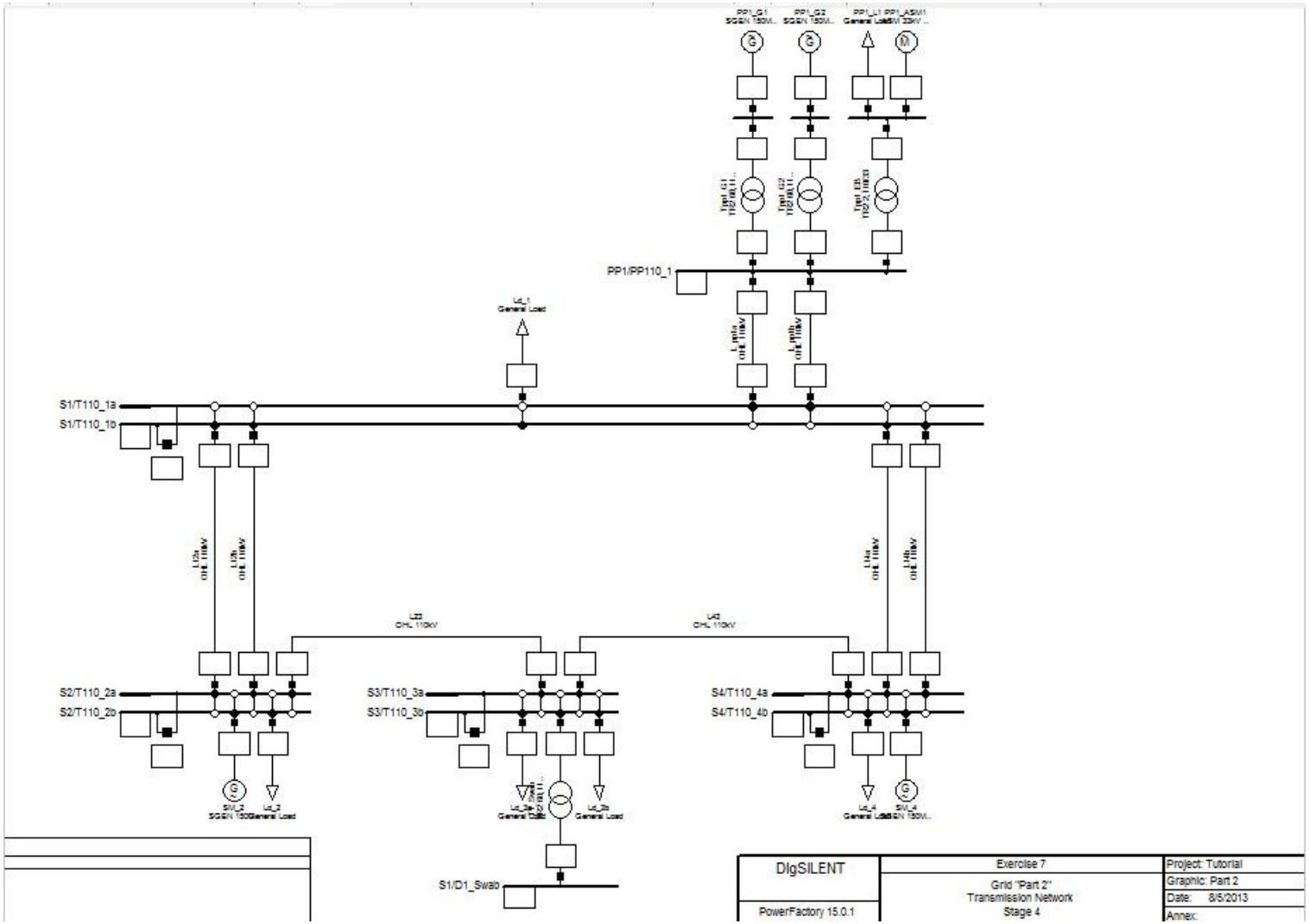

Figure 5-2. Simulation Model

The model being validated is a 150 MVA, 33 kV salient pole synchronous generator found within the power plant (Figure 5-3), marked as PP1_G2, which also includes an exciter and a governor in addition to the standard generator parameters. The validation takes advantage of a fault occurring close to the plant. It is applied to one of the 110 kV lines, labelled as L_pp1a, connecting the generating station to the bus S1, The disturbance is expected to draw a good dynamic response out of the machine due to its proximity. The big question however is whether most, if not all of the parameters will have detectable influence on the dynamics after the fault is applied. As of this study, the effect varying fault distances to detectability is not investigated, but it is assumed that the most inclusive response, in terms of model parameters, can be captured when it is closest to the disturbance.

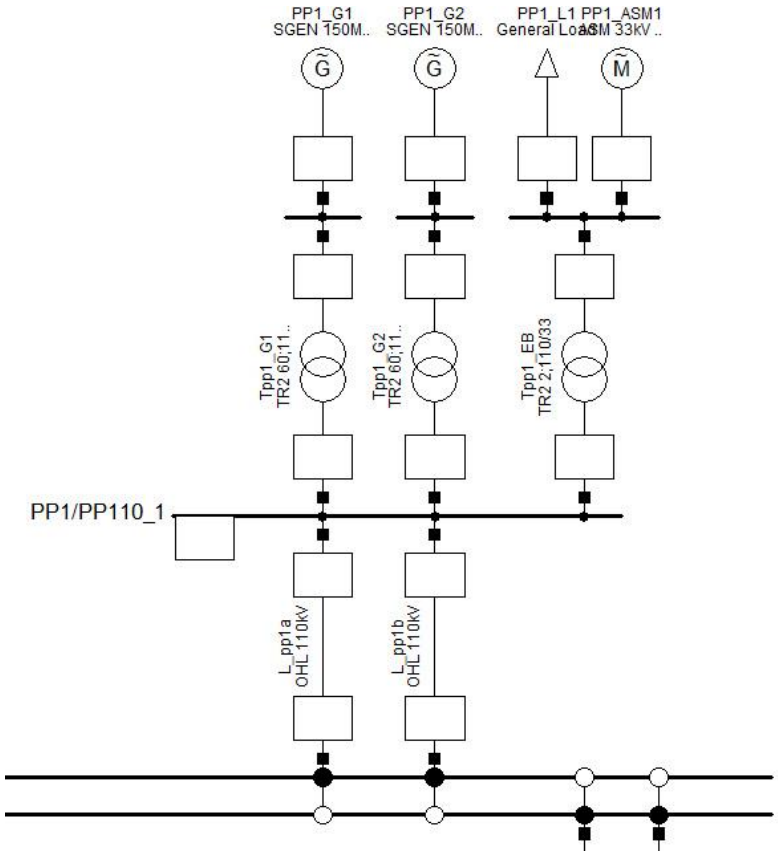

Figure 5-3. Close up of generating station

The disturbance events applied to the 110 kV overhead line L_pp1a include two of the most common short circuit events, namely the single line-to-ground fault, and the double line-to-ground fault. The single line-to-ground is the most common type of fault, which is about 70 to 80% of all faults in terms of frequency of occurrence, and will therefore be the best type of power

system event to validate against. This is followed by double line-to-ground fault by a huge margin, accounting for 10 to 17% of all faults [63]. Incidentally, due to the unbalanced nature of the selected short circuit events, it will allow for comparison of classification accuracy between different sequence components. Both faults were be applied halfway across the length of the line, but for the testing to account for any inaccuracies in fault location due to imprecise measurement, an allowance of $\pm 10\%$ was randomly padded to the distance for each run. Three-phase short circuit tests were also conducted. Due to the symmetrical nature of three-phase faults, only the positive sequence data can be evaluated for classification. For all short circuit types, the faulted phases are cleared 0.2 seconds after the inception of the fault. To simulate the Synchrophasor measurements, the sequence outputs were sampled at a rate of 60 frames per second, which is one of the possible settings in a PMU in a 60 Hz network.

As mentioned previously, the parameters are subdivided into error classes. Each class covers a range of possible values that a parameter can assume. It is important to note that the ranges are separated from the base value by a certain size of margin. The margin should provide enough space for the base value to assume a unique response from the adjacent class ranges. The margins, indicated by the dashed dimension lines in Figure 5-4, separates the base from both the upper and lower ranges. For this study, the margin lengths were arbitrarily selected, and could vary anywhere from 0.30% to 50%, depending on the size of the difference between the base, and the upper or lower limit of the parameter.

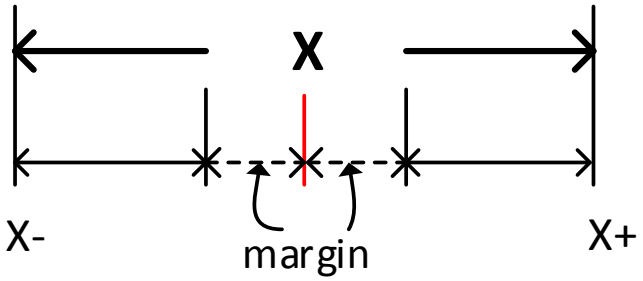

Figure 5-4. Parameter range margin

If the margin for a certain range is too close to the base parameter value, it is likely that the range is omitted. The proper sizing of the margin for validation is another matter that has to be further investigated, as a parameter's practical working range and weight on the dynamics are not typically made available in pertinent documents. The ranges used in the study (Table 5-1 to Table

5-3) were based on tables and examples from [13]. It can be seen however, that there could be inconsistencies in the documented range and the default settings used (Table 5-1) where the base value is lower than the suggested lower limit (highlighted in red), in which case the range is just removed.

Table 5-1. Standard Parameter default settings and available range

Standard Parameters	Base	LL	UL	Margin	%LL Margin	%UL Margin
Xd (synchronous reactance, pu)	1.68	0.60	2.30	0.1000	9.26	16.13
Xq (synchronous reactance, pu)	1.61	0.40	2.30	0.1000	8.26	14.49
Td' (transient time constant, s)	1.00	1.50	10.00	0.5000		5.56
Xd' (transient reactance, pu)	0.30	0.20	0.40	0.0250	25.00	25.00
Td'' (subtransient time constant, s)	0.05	0.01	0.05	0.0100	25.00	
Tq'' (subtransient time constant, s)	0.05	0.01	0.05	0.0100	25.00	
Xd'' (subtransient reactance, pu)	0.20	0.12	0.25	0.0250	31.25	
Xq'' (subtransient reactance, pu)	0.20	0.12	0.25	0.0250	31.25	
Xl (stator leakage reactance, pu)	0.10	0.10	0.20	0.0250		25.00

Table 5-2. Exciter parameter default settings and available range

Exciter	Base	LL	UL	Margin	%LL Margin	%UL Margin
Tr (measurement delay, s)	0.01	0.00	0.05	0.0025	25.00	6.25
Ka (controller gain, pu)	150.00	100.00	200.00	5.0000	10.00	10.00
Ta (controller time constant, s)	0.03	0.02	0.89	0.0025	50.00	0.29
Ke (excitor constant, pu)	1.50	0.00	2.00	0.1000	6.67	20.00
Te (excitor time constant, s)	0.40	0.40	1.20	0.1000		12.50
Kf (stabilization path gain, pu)	0.00	0.00	0.06	0.0005		0.88
Tf (stabilization path time constant, s)	0.10	0.10	1.00	0.1000		11.11

Table 5-3. Governor parameter default settings and available range

Governor	Base	LL	UL	Margin	%LL Margin	%UL Margin
R (permanent droop, pu)	0.04	0.01	0.10	0.0025	8.33	4.17
r (temporary droop, pu)	0.50	0.10	1.00	0.0025	0.63	0.50
Tr (governor time constant, s)	8.41	5.00	12.00	0.5000	14.67	13.92
Tf (filter time constant, s)	0.05	0.01	0.10	0.0025	6.25	5.00
Tw (water starting time, s)	0.10	0.10	2.00	0.0250		1.32
At (turbine gain, pu)	1.15	1.00	1.50	0.0250	16.67	7.14
Tg (servo time constant, s)	0.50	0.10	1.00	0.0250	6.25	5.00

The parameter randomization subroutine algorithm is described in the following flowchart.

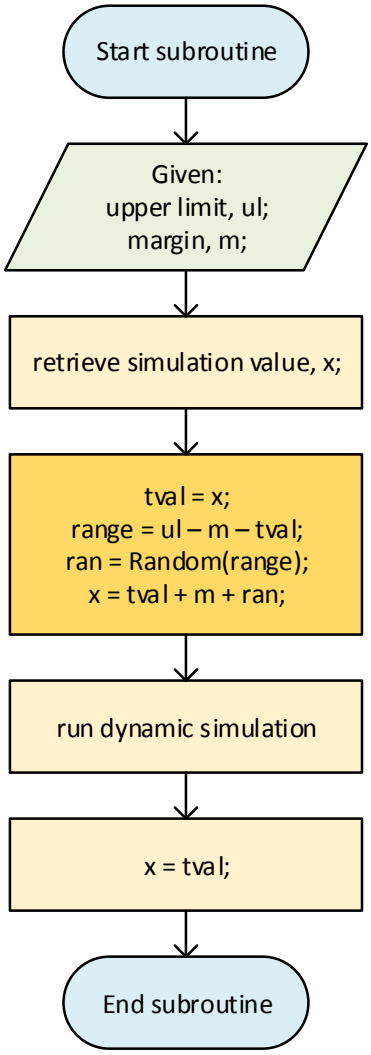

Figure 5-5. Parameter Randomization algorithm

The subroutine shown in Figure 5-5 randomizes a parameter x based on the allowable range, determined by the upper or lower limit, and the set margin, then restores the original value after the dynamic simulation is run. Moreover, the fault location randomization, not included in the flowchart above, is performed in conjunction with this procedure. The subroutine was written using DPL in DIgSILENT, where each parameter has its own randomization code, and accessed as needed. Each parameter subroutine is run a set number of times and assigned a nominal class describing the type of error induced. The full code can be seen at Appendix A.2.

5.2.2 Data Processing

The dynamic simulation output from DIgSILENT is configured to be written in a .txt format. Each file is one simulation run, containing all the measurements used in the study, and each measurement is an array of time-domain samples covering the required length of window. These raw measurements need to be further processed before being added as an instance to the data set, which will be then used for generating the classifier. The number of simulations required to make a satisfactory classifier necessitates a complete automation of the whole process. Also, the automation makes this method of model validation a more viable option compared to other existing practices. The size of data required, and the speed at which the whole task needs to be completed is clearly becoming less of a concern in the information age. To accomplish all this, a custom application was designed and written using the C# programming language. The complete architecture showing the flow of data is seen Figure 5-6.

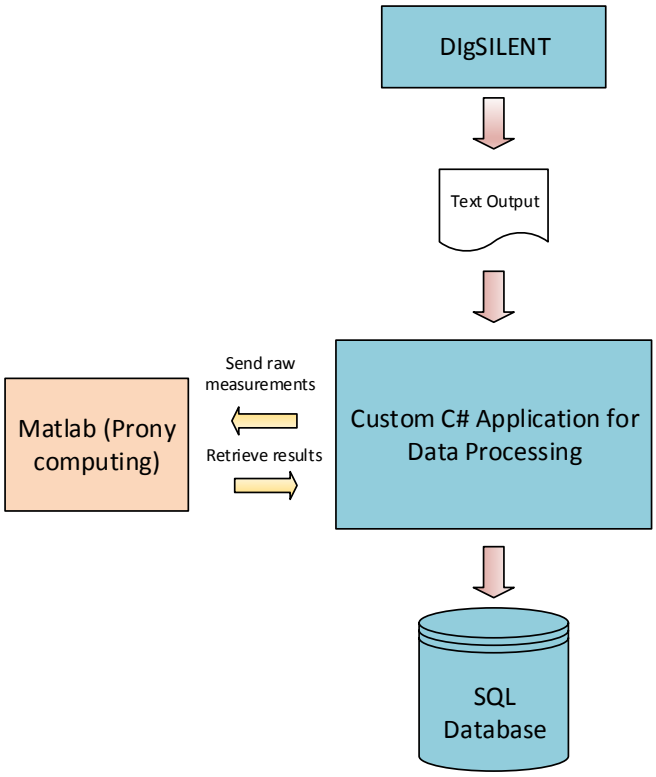

Figure 5-6. Data Processing Architecture

The custom application, which was named DIgSILENT Text Output Processor, handles all the necessary tasks between the simulation output and database storage (see Appendix B.1). The program runs according to the settings provided in the interface such as the measurements needed,

number of case repetitions, the list of parameters to be processed and others (Figure 5-7). The database used for building a classifier has to be configured prior to running the program, and for this purpose, a separate program was written for this, and is described in Section 5.2.4.

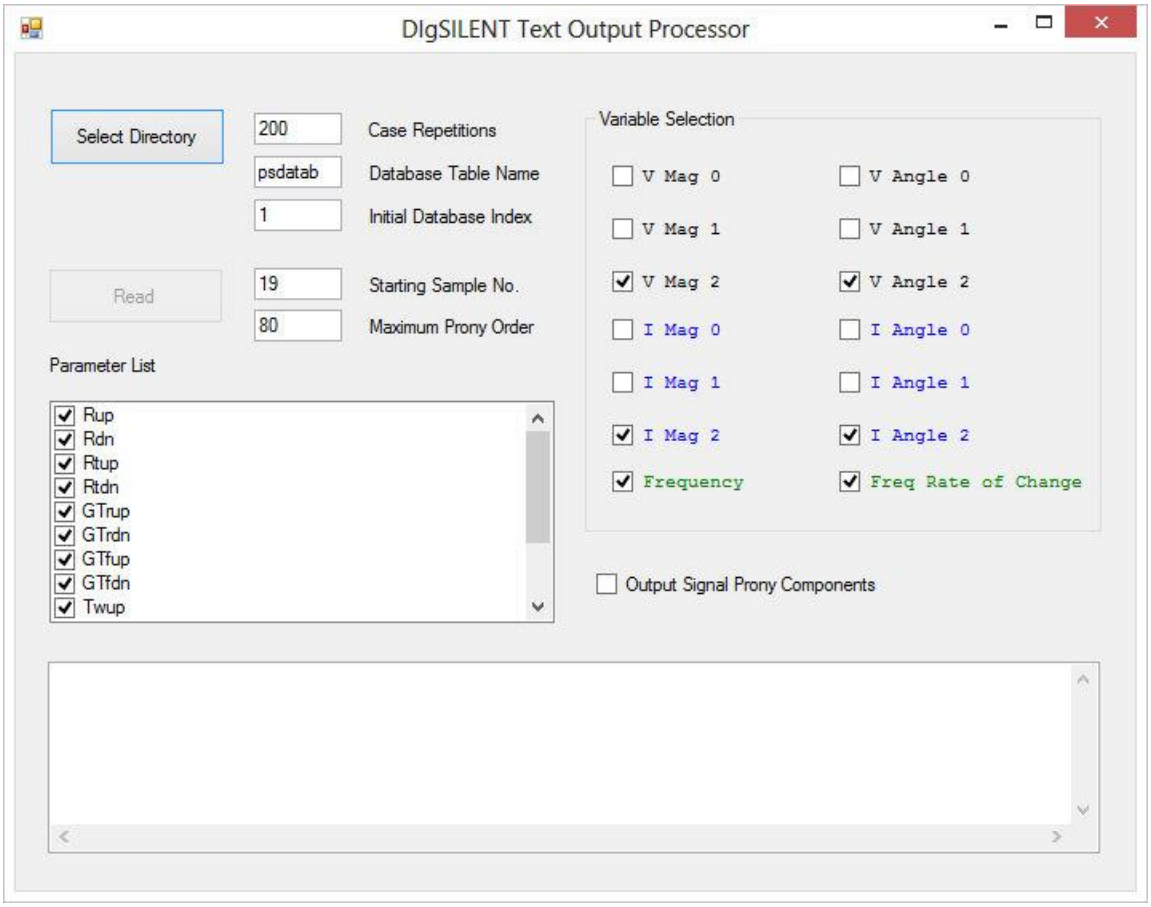

Figure 5-7. Text Output Processor Screenshot

A flowchart of the tasks performed inside the Text Output Processor is shown in Figure 5-8. The application first looks for the text files containing the parameter name specified by the user in the program interface list box. The names are accordingly used as a suffix in the filenames, and it also corresponds to the nominal class all the measurements in the file belong to. The text file output from DIgSILENT PowerFactory contains the measurement stream from zero, positive and negative sequence voltage and current, the frequency and rate of change of frequency. All have been recorded at 60 samples per second to emulate a typical sampling rate for a phasor measurement unit. Moreover, the file contains the timestamp for each sample value. The C# software program, allows the user to select the measurements to be used for the training set, provides the interface for setting the starting sample, the database table to be used with the index,

and also set the maximum Prony order to be used for decomposing the signal. Furthermore, the list of nominal classes used in the interface for the training and classification process is in the form of a checked list box, where an item can be deselected according to what is needed in the study.

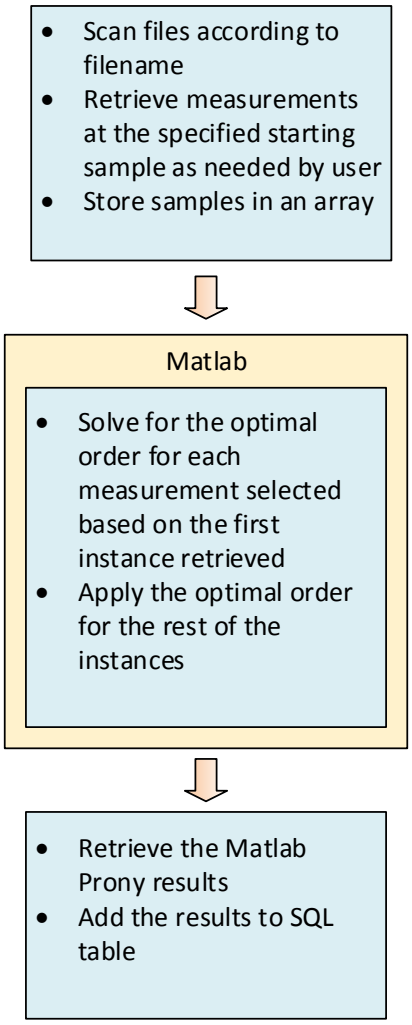

Figure 5-8. Text Output Processor application flowchart

The application has been designed to interface with the command line Matlab, which applies Prony decomposition on the measurements, as described in Chapter 4. For this study, the sample stream input to Prony is configured in the user interface, such that it only contains the transient information after the last discontinuity, which in this case is the opening of the breaker. Since all PMU measurements are synchronized, the starting sample will be applied to all measurements in the file. Variables input by the user, such as the settings and measurement data can be sent to the command line module of Matlab then retrieved as needed. The Component Object Model (COM) called *MLapp* makes it possible to interface C# with Matlab, which will then

handle all computation-intensive processes. The MLapp interface will allow control over the Prony analysis script, which is written for the specific set of data fed by the Text Output Processor application.

Once the Matlab Prony output has been retrieved, the SQL database table assigned to the dataset will be queried to add the results to the designated columns. Each row in the SQL table contains the corresponding nominal class that will be later used by the data mining algorithm as basis for building the classifier. One should note that the SQL database can only fit 1024 columns, thus the number of measurements and the Prony order used should be limited. For an order n Prony decomposition, the number of elements output is $2n + 1$. The size of the order used in the study is 80, thus each measurement has 161 entries into the database table, which is still a reasonable amount of data compression from a total window length of 614 samples, after truncation. With frequency, rate of change of frequency, and the voltage and current measurements having a magnitude and angle component, the number of column entries into the table will be $6 \times 161 = 966$, plus the nominal variable column. This size will thus be enough to fit one set of sequence component information into the database table, as intended.

5.2.3 Prony

The stock Prony function in Matlab was designed mainly for parametric estimation to model filters. It gives the numerator and denominator coefficients for a filter whose impulse response is similar to the input signal, using a given number of poles and zeroes [64]. Instead of solving for the properties of the decaying sinusoid from the resulting coefficients, the coefficients are directly used as the predictor variables, effectively reducing the variable entries, which will then speed up the training and classification process. For the purpose of the study, slight modifications had to be made to properly extract the required features from the measurements. The modified code was based off of the *HPRONY* script (see Appendix C.1) from the Higher Order Spectral Analysis (HOSA) toolbox from the Matlab file exchange site [65]. The code mainly adds preventive measures to avoid computation errors. One such safeguard truncates leading and trailing zeroes from the input data sequence, which can cause problems with Prony calculations, and another measure handles Toeplitz incompatibility. In addition, the modified code sets the size of zeroes and poles to $n - 1$ and n respectively, for a given order size n.

Before Prony is computed for the whole training set, the optimal order for each measurement is computed, and this is all based on the first record in the data set. Normally, the bigger the order number, the better the fit. However, due to the limited column size of the SQL database, the maximum order size should accordingly be limited. For this reason, prior to the tests, the default order size had to be determined. This was done by sampling measurements and inspecting the fit for different orders. It was determined that a good fit was achieved for most measurements at order 80 (Figure 5-9 to Figure 5-10). This is however not directly applied to measurements, rather, it is set as the maximum order. Prior to applying the decomposition, the optimal Prony order for each measurement at the first record of the training set is computed. It therefore, for example, is possible to have an order value less than 60, but nothing greater than 80. (See Appendix C.2.2)

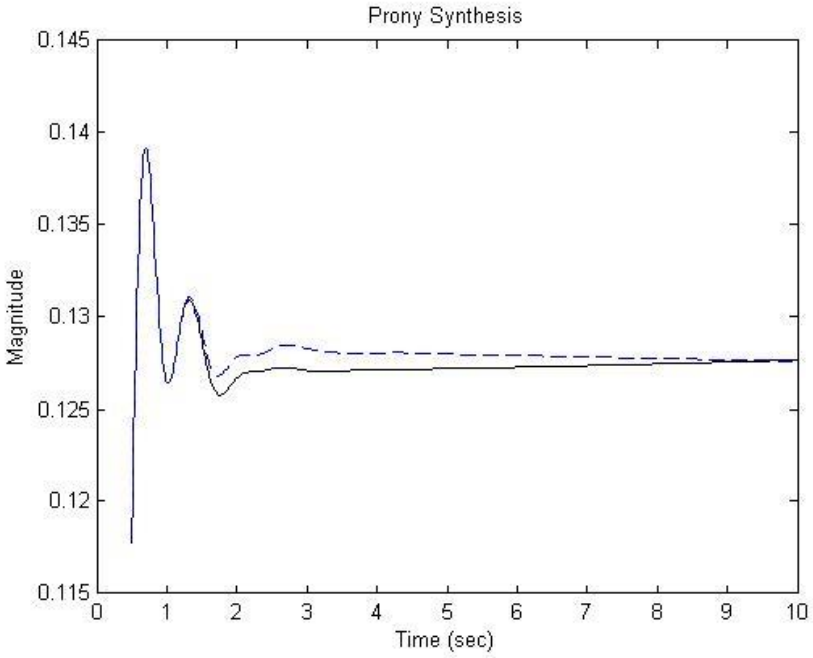

Figure 5-9. Prony fitting at order 30

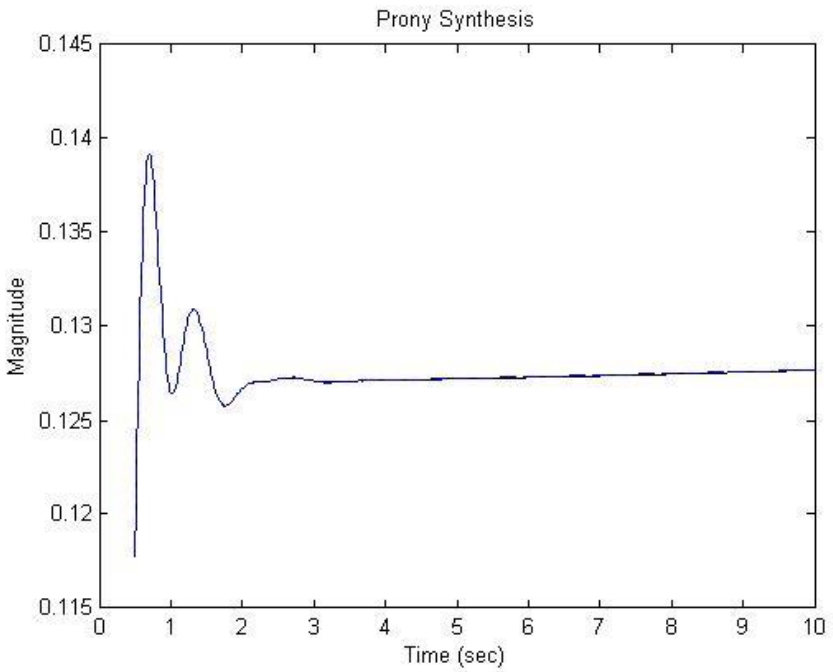

Figure 5-10. Prony fitting at order 80

The optimal order is determined by computing for the mean squared error between the input measurement, and the decomposed and subsequently recombined measurement according the selected number of poles. The order conversion with the smallest mean squared error is picked as the optimal value for a measurement, and will then be applied to the whole batch of measurements in the training data. The result of the decomposition will then be retrieved by the custom application and written into an SQL database table. If any of the measurements have order values less than the set maximum, the Prony script simply pads zeroes to accommodate it in the table.

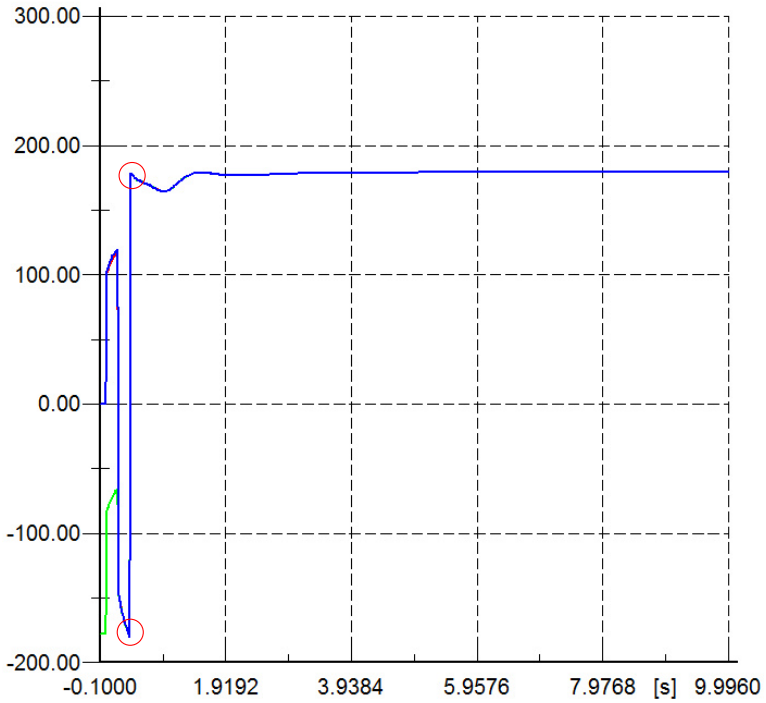

Figure 5-11. Angle jump, marked with the red circles

There are instances in the simulation when the software outputs angle measurements that have drastic jumps when the values are bordering ± 180 (Figure 5-11). This can also cause problems with the Prony calculations. To remedy this, the angle measurements are all subject to correction using a Matlab code found in Appendix C.2.5.

5.2.4 Schema Generator

Prior to running the simulation, the required database table should already exist to receive the variables for the dataset. A database schema generator application (Figure 5-12) was built specifically to create a table structure based on the measurements needed by the user for classification.

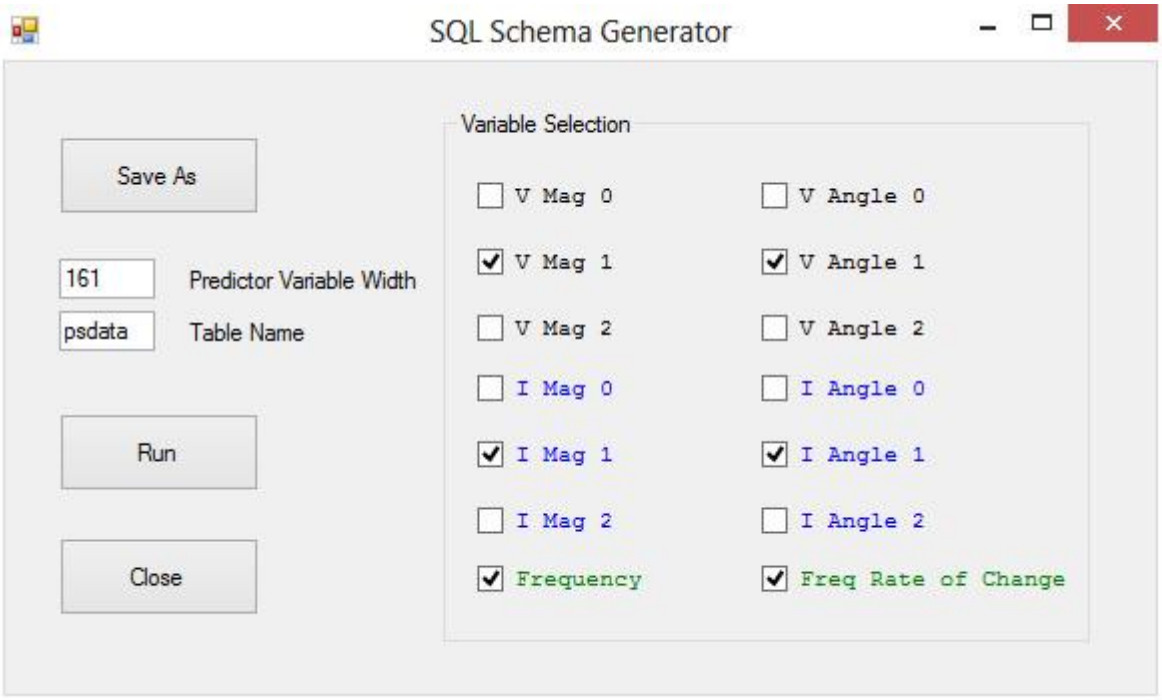

Figure 5-12. Schema Generator Interface

The user simply has to match the set of variables and the table name from the Text Output Processor settings. Also, the number of columns taken up by each measurement should be explicitly indicated in the application. This size of the measurement columns is simply based on the maximum order used in the Prony decomposition, as mentioned in Section 5.2.3. Once the schema file is generated, it is then executed and SQL automatically creates the table. The query generated by the application was designed such that, if another table exists with the same name as the most recent one created, it will drop the old table. Thus the generated schema can also be used to refresh or replace the previous tables to make way for new training data (see Appendix B.2).

5.2.5 Data Mining

The machine learning software used for all tests is a suite called WEKA or Waikato Environment for Knowledge Analysis and includes Random Forests among other data mining algorithms [43]. Moreover, the software conveniently can read a number of database formats, including SQL which was used for the study. The raw SQL table collected from the simulation measurements contains an ‘id’ column for indexing, which is manually removed prior to feeding the dataset to the Random Forest function. All Random Forest algorithm settings, except for the number of trees used for the ensemble are kept at the default value. For all tests, a 10-fold cross-

validation is used. The program outputs the results of the classification in a text format, which includes the out-of-bag error rate, several measures of accuracy described in Chapter 6, and the confusion matrix.

5.3 Tests

5.3.1 Single Parameter Error Test

The first series of tests for the Random Forest algorithm was designed to determine if single parameter value errors will be manifested in the dynamic measurements, and whether these variations, if any at all, are distinct from all the other induced errors under observation. As a corollary, the experiments will be used evaluate the effectiveness of single parameter error classification by signal pattern recognition.

The first half of the experiment makes a comparative study of classifiers generated using different sequence components. The data, in the form of positive, negative, and zero sequence information are all available measurements coming from phasor measurement units. A single line-to-ground fault was applied on a line close to the generator model under test, as described in Section 5.2.1. The parameters in the discrete units forming the generator, specifically the governor, exciter, and the machine itself, were separately evaluated for the test. The sequence measurements come from the bus voltage and line current indicated in Section 5.2.1. Moreover, the frequency and rate of change of frequency measurements were all added to the sequence measurements tested.

The second half of the test combines all generator parameters to determine whether the classification accuracy is maintained, with a larger selection of classes. Also, the test only uses what was eventually determined to be the most accurate sequence information for classification from the first half of the single-error experiment. For this half of the experiment, the validation is tested for two unbalanced events: a single line-to-ground and a double line-to-ground fault, where both faults are applied in the same line and at the same target location.

For the final part, two three-phase short circuit fault tests were conducted. Since the fault completely cuts off the connections of line L_pp1a when the breaker opens, the line currents are

absent, post fault, which reduces the predictor variables for the dataset. Also, since the fault is symmetrical, only the positive sequence data is available for analysis. The preceding conditions were used for the first test. For the second test, an alternative current measurement source was used. The current was simply taken from the 110 kV line parallel to the faulted line, marked L_pp1b, which is expected to pick up all of the current immediately upon L_pp1a's disconnection.

All of the experiments for single error parameter validation run the DIgSILENT DPL script in Appendix A.1. The script simply runs a list of subroutines, where each subroutine was designed to specifically work with a single parameter error, as described in Section 5.2.2 . The subroutine in itself executes the dynamic simulation, with all the settings such as simulation time step, output rate, and fault event predefined and applied for all active cases in the classification study.

The subroutines, when selected to run, assigns its own tag or suffix to the file created on execution. This distinguishes the different nominal classes associated with each parameter error made for the data set. The naming convention used for each parameter error typically begins with its known shorthand notation, followed by an *'up'* or *'dn'* suffix to indicate whether the error was higher or lower than the default model value. For example the class name for the d-axis synchronous reactance ***Xd*** errors at the upper range is **Xdup**, and that of the q-axis synchronous reactance ***Xq*** at the lower range is **Xqdn**. For the transient and subtransient parameters that contain apostrophe marks such as ***X'd*** or ***X''q***, the names are appended with the character *'s'*, followed by the range suffix, with the number of added characters depending on the number of apostrophes. The d-axis transient reactance at the lower range, for example, will be **Xdsdn**. There are parameter shorthand notations used by multiple models such as ***Tr*** and ***Tf***, which are found in both the exciter and governor. In this case, the class name is prefixed with an additional character. For example, instead of using **Trup** again for the governor, **GTrup** is used. Finally, the default simulation parameter settings is assigned its own class, simply called **Base**. All the nominal classes used for the test are given in detail in Chapter 6.

5.3.2 Multiple Parameter Error Test

In general, the main difficulty in model validation, especially those involving several combinations of various components that contribute to the system dynamics, is simultaneously tracking all sources of error with the limited measurements available. The case may likely be that,

model errors can have very subtle, faintly detectable contributions to one type of response, and have a substantial influence on another. This set of experiments endeavors to develop a classifier that can check the presence of a parameter error that is simultaneously active with other model errors.

The multiple error parameter validation study approaches this problem by simply creating separate classifiers for each error of interest. Each classifier has a target 'subject' parameter error, and the objective is to be able to detect the presence, or absence of the subject error. The hypothesis for this experiment is that errors have unique signatures in the recorded measurements that can be tracked and used to determine its presence. Each classifier in this test will then require a simple binary classification of 'ON' or 'OFF', to designate an error's existence or non-existence, respectively.

The simulation script designed for the test contains all random errors that can be selectively switched on or off (see Appendix A.3). The subject parameter error will assume the on and off state, while the other random parameter errors will all be run, initially, at the same time as the subject. One by one, the background parameter errors will be switched off to observe the effect of the errors on the subject classifier's accuracy (Figure 5-13). For each random background error eliminated, a classifier is generated and the error rate is subsequently determined.

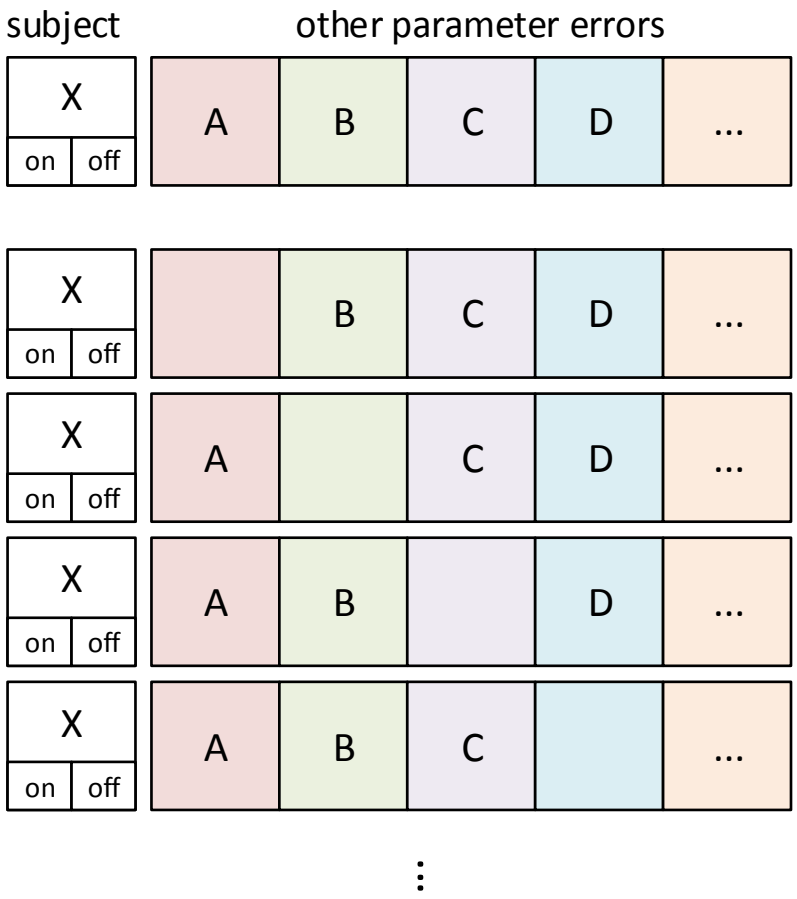

Figure 5-13. Multiparameter validation illustrating the two states of a subject parameter, and the switchable random background errors. Each row in the figure is used to train a classifier.

For the series of tests performed, only the generator standard parameter set was evaluated, due to the large number of simulations required. Also, the best sequence information from a single line-to-ground disturbance was used for generating the classifier. For each classifier, the weight, or the impact of the selected random parameter errors on the subject parameter was determined based on the changes in accuracy, which is estimated using the out-of-bag (OOB) error computed by Weka.

For each error classifier, the test starts with all parameter errors active. Ideally, a classifier should be able to pick up the subject error's distinguishing features while mixed with all random errors. Considering the non-linear nature of the dynamic models, however, this can hardly be realized. Thus, at each test, a selected parameter error is turned off, and the resulting average OOB error is recorded. This process aims to identify the biggest parameter(s) that obscure the classification of the subject parameter. The multi-parameter error study is designed to group tests

into sets, where each set contains the same number of actively running random background errors. For each set, after the largest error causing parameter is determined, it is permanently turned off. A new set of tests is subsequently run to search for the next largest error-causing parameter, with respect to the subject parameter. This process is repeated until the last parameter is removed or if an OOB error of 0 % is reached. The nominal classes used for this series of tests uses the same classes for single parameter errors, but appended with either ‘On’ or ‘Off’.

6 Results and Discussion

6.1 Weka Output Report Interpretation [66]

There are several measures used for interpreting the accuracy of the generated classifier in Weka. The quantities are mainly based on what is called the confusion matrix or contingency table.

```
=== Confusion Matrix ===

a b   <-- classified as
7 2 | a = yes
3 2 | b = no
```

Figure 6-1. Confusion Matrix [66]

The confusion matrix breaks down a classifier's correct and incorrect classifications by creating class labels at the row and column sections. The row labels mark the actual class, while the column labels are the classification results. If for instance, if we have only two classes 'a' and 'b' as shown in Figure 6-1, the element at ('a','a') designates the number of instances belonging to 'a' that were correctly classified as 'a'. Similarly, the element at ('b','b') are the number of correctly classified 'b' instances. The elements crossing classes ('a','b') and ('b','a') are the misclassified instances. Assuming a certain class x, the following measures of accuracy are then derived from this matrix

1. True Positive (TP) rate or Recall – "the proportion of examples which were classified as x, among all examples which truly have class x". Based on Figure 6-1, this would be $7/(7+2) = 0.778$ for class 'a' and $2/(3+2) = 0.4$ for class 'b'.
2. False Positive (FP) rate – "the proportion of examples which were classified as class x, but belong to a different class, among all examples which are not of class x." For class 'a', that would be all instances misclassified as 'a', which is in the figure is only element ('a','b') = 3, out of all instances not belonging to class 'a', which is the sum of all rows except those in row 'a'. Thus FP for 'a' is $3/(3+2) = 0.6$. For class 'b' it is $2/(2+7) = 0.222$.

3. Precision – "the proportion of the examples which truly have class x among all those which were classified as class x." In this case for class 'a' it is element ('a','a') over the sum of the whole class 'a' column, which is $7/(7+3) = 0.7$. For class 'b' it is $2/(2+2) = 0.5$.
4. F-Measure – is $2 * Precision * Recall/(Precision + Recall)$.
5. ROC (Receiver Operating Characteristic) Area – is a measure which can vary between "0.5 (no discrimination power) to 1.0 (perfect accuracy)." The ROC curves (Figure 6-2) take the True Positive and False Positive quantities to measure a classifier's ability to distinguish classes correctly [67, 68]. Computing the ROC Area is more involved than the previous measures and will not be covered in this discussion.

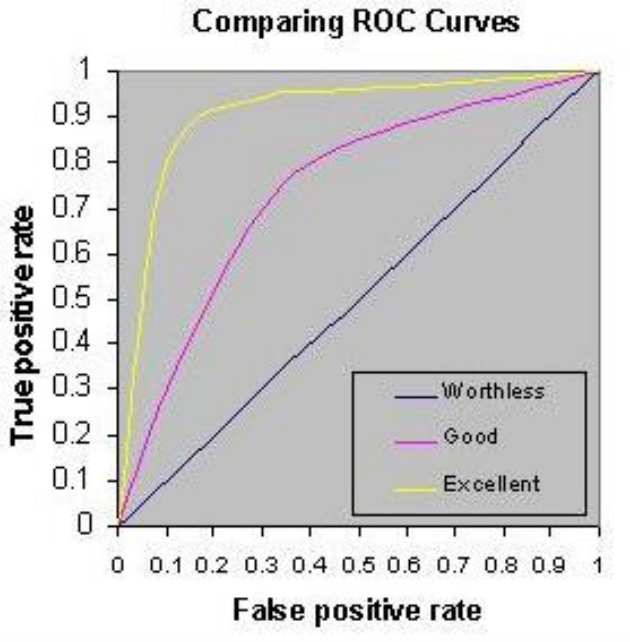

Figure 6-2. ROC Curve Comparison [68]

6.2 Single Parameter Error Experiment

The single parameter error test investigates a parameter error's detectability from the selected simulation disturbance measurements. It is hypothesized that each parameter of the model has a unique influence on the dynamics of the event which can be captured and later used for identifying the cause of discrepancy between a simulated and measured data. For this series of tests, it is assumed that only one error exists in the entire model at a given time, therefore the generated classifier will only work for such condition.

The first part of the test in section 6.2.1 compares the accuracy of the different classifiers based on the positive, negative, and zero sequence component data. The classifiers are tested on the standard generator, exciter, and governor models individually, for a single line-to-ground short circuit. For section 6.2.2, using the best sequence measurement, the models are combined to evaluate the classification algorithm's capability to sort distinguishing characteristics from each parameter error with a large assortment of classes. The latter section tests the classifier's performance for two disturbances, which are the single line-to-ground and double line-to-ground faults. Finally, Section 6.2.3 tests the classification accuracy for combined models with a three-phase short circuit, where one test has no current measurement, and the other uses current information from another line.

6.2.1 Sequence Data Comparison

An investigation into the classification accuracy of different sequence components was conducted. The test covered the generator, exciter and governor parameters subjected to a single line-to-ground fault. The simulation data output was set at 60 samples per second, and each nominal class has 100 randomly generated instances. The comparison was made with the number of RF trees fixed at 100.

A. Standard Parameters

For the standard parameter tests, the nominal classes used are shown in Table 6-1.

Table 6-1. Standard Parameter Nominal Classes

Base	Default Simulation Setting
Xdup	(Xd) synchronous reactance, upper range
Xddn	(Xd) synchronous reactance, lower range
Xqup	(Xq) synchronous reactance, upper range
Xqdn	(Xq) synchronous reactance, lower range
Tdsup	(Td') transient time constant, upper range
Xdsup	(Xd') transient reactance, upper range
Xdsdn	(Xd') transient reactance, lower range
Tdssdn	(Td'') subtransient time constant, lower range
Tqssdn	(Tq'') subtransient time constant, lower range
Xdssdn	(Xd'') subtransient reactance, lower range
Xqssd	(Xq'') subtransient reactance, lower range
Xlup	(Xl) stator leakage reactance, upper range

A.1 Positive Sequence

Table 6-2. Positive Sequence Classification Accuracy for Standard Parameters

=== Detailed Accuracy By Class ===						
TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.99	0.018	0.825	0.99	0.9	0.996	Base
0.95	0.007	0.922	0.95	0.936	0.998	Xdup
0.99	0.001	0.99	0.99	0.99	1	Xddn
0.94	0	1	0.94	0.969	1	Xqup
0.98	0	1	0.98	0.99	1	Xqdn
0.93	0	1	0.93	0.964	1	Tdsup
1	0.003	0.971	1	0.985	1	Xdsup
0.99	0	1	0.99	0.995	1	Xdsdn
0.98	0	1	0.98	0.99	1	Tdssdn
1	0	1	1	1	1	Tqssdn
1	0	1	1	1	1	Xdssdn
0.99	0	1	0.99	0.995	1	Xqssdn
0.83	0.008	0.892	0.83	0.86	0.993	Xlup
Weighted Avg. 0.967	0.003	0.969	0.967	0.967	0.999	

Table 6-3. Positive Sequence Confusion Matrix for Standard Parameters

a	b	c	d	e	f	g	h	i	j	k	l	m	<-- classified as
99	0	0	0	0	0	0	0	0	0	0	0	1	a = Base
2	95	0	0	0	0	0	0	0	0	0	0	3	b = Xdup
1	0	99	0	0	0	0	0	0	0	0	0	0	c = Xddn
0	0	0	94	0	0	0	0	0	0	0	0	6	d = Xqup
0	0	1	0	98	0	1	0	0	0	0	0	0	e = Xqdn
0	7	0	0	0	93	0	0	0	0	0	0	0	f = Tdsup
0	0	0	0	0	0	100	0	0	0	0	0	0	g = Xdsup
1	0	0	0	0	0	0	99	0	0	0	0	0	h = Xdsdn
0	0	0	0	0	0	2	0	98	0	0	0	0	i = Tdssdn
0	0	0	0	0	0	0	0	0	100	0	0	0	j = Tqssdn
0	0	0	0	0	0	0	0	0	0	100	0	0	k = Xdssdn
1	0	0	0	0	0	0	0	0	0	0	99	0	l = Xqssdn
16	1	0	0	0	0	0	0	0	0	0	0	83	m = Xlup

The overall classification of standard parameter errors using positive sequence data Table 6-2 is fairly good with majority of classes having the F-Measure and ROC Areas greater than 90%. Two parameters **Tqssdn** and **Xdssdn** have perfect classification accuracy across all measures. Most classes registered perfect precision, resulting to good ROC area rates. Only one parameter, **Xlup** tallied a TP rate lower than 90%. The confusion matrix (Table 6-3) revealed that a sizable number of instances have been misclassified as **Base**, and can perhaps be an indication that **Xlup** has relatively weak distinguishing characteristics.

A.2 Negative Sequence

Table 6-4. Negative Sequence Classification Accuracy for Standard Parameters

=== Detailed Accuracy By Class ===						
TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
1	0	1	1	1	1	Base
1	0.003	0.971	1	0.985	1	Xdup
1	0	1	1	1	1	Xddn
1	0	1	1	1	1	Xqup
1	0	1	1	1	1	Xqdn
0.98	0	1	0.98	0.99	1	Tdsup
1	0	1	1	1	1	Xdsup
1	0	1	1	1	1	Xdsdn
1	0	1	1	1	1	Tdssdn
1	0	1	1	1	1	Tqssdn
1	0	1	1	1	1	Xdssdn
1	0	1	1	1	1	Xqssdn
0.99	0	1	0.99	0.995	1	Xlup
Weighted Avg. 0.998	0	0.998	0.998	0.998	1	

Table 6-5. Negative Sequence Confusion Matrix for Standard Parameters

=== Confusion Matrix ===													
a	b	c	d	e	f	g	h	i	j	k	l	m	<-- classified as
100	0	0	0	0	0	0	0	0	0	0	0	0	a = Base
0	100	0	0	0	0	0	0	0	0	0	0	0	b = Xdup
0	0	100	0	0	0	0	0	0	0	0	0	0	c = Xddn
0	0	0	100	0	0	0	0	0	0	0	0	0	d = Xqup
0	0	0	0	100	0	0	0	0	0	0	0	0	e = Xqdn
0	2	0	0	0	98	0	0	0	0	0	0	0	f = Tdsup
0	0	0	0	0	0	100	0	0	0	0	0	0	g = Xdsup
0	0	0	0	0	0	0	100	0	0	0	0	0	h = Xdsdn
0	0	0	0	0	0	0	0	100	0	0	0	0	i = Tdssdn
0	0	0	0	0	0	0	0	0	100	0	0	0	j = Tqssdn
0	0	0	0	0	0	0	0	0	0	100	0	0	k = Xdssdn
0	0	0	0	0	0	0	0	0	0	0	100	0	l = Xqssdn
0	1	0	0	0	0	0	0	0	0	0	0	99	m = Xlup

Classification of standard parameter errors using negative sequence data has outstanding rates across the board (Table 6-4). Majority of the F-Measure and ROC Area values are perfect. Those classes with less than 100% rates only suffered very minor flaws. The parameter class **Xdup** accounts its imperfect F-measure to small precision problems. Minor misclassification errors from both **Tdsup** and **Xlup** resulted to imperfect TP rates, and, as can be seen in the confusion matrix (Table 6-5), contributed to **Xdup**'s decrease in precision.

A.3 Zero sequence Standard Parameters

Table 6-6. Zero Sequence Classification Accuracy for Standard Parameters

=== Detailed Accuracy By Class ===						
TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
1	0.003	0.971	1	0.985	1	Base
0.99	0.003	0.961	0.99	0.975	1	Xdup
1	0	1	1	1	1	Xddn
0.98	0.001	0.99	0.98	0.985	1	Xqup
0.99	0	1	0.99	0.995	1	Xqdn
0.99	0.001	0.99	0.99	0.99	1	Tdsup
1	0	1	1	1	1	Xdsup
0.99	0.001	0.99	0.99	0.99	1	Xdsdn
1	0	1	1	1	1	Tdssdn
0.98	0	1	0.98	0.99	1	Tqssdn
1	0	1	1	1	1	Xdssdn
1	0.002	0.98	1	0.99	1	Xqssdn
0.95	0.001	0.99	0.95	0.969	0.999	Xlup
Weighted Avg. 0.99	0.001	0.99	0.99	0.99	1	

Table 6-7. Zero Sequence Confusion Matrix for Standard Parameters

=== Confusion Matrix ===													
a	b	c	d	e	f	g	h	i	j	k	l	m	<-- classified as
100	0	0	0	0	0	0	0	0	0	0	0	0	a = Base
0	99	0	1	0	0	0	0	0	0	0	0	0	b = Xdup
0	0	100	0	0	0	0	0	0	0	0	0	0	c = Xddn
0	0	0	98	0	0	0	0	0	0	0	1	1	d = Xqup
1	0	0	0	99	0	0	0	0	0	0	0	0	e = Xqdn
0	0	0	0	0	99	0	1	0	0	0	0	0	f = Tdsup
0	0	0	0	0	0	100	0	0	0	0	0	0	g = Xdsup
0	0	0	0	0	1	0	99	0	0	0	0	0	h = Xdsdn
0	0	0	0	0	0	0	0	100	0	0	0	0	i = Tdssdn
2	0	0	0	0	0	0	0	0	98	0	0	0	j = Tqssdn
0	0	0	0	0	0	0	0	0	0	100	0	0	k = Xdssdn
0	0	0	0	0	0	0	0	0	0	0	100	0	l = Xqssdn
0	4	0	0	0	0	0	0	0	0	0	1	95	m = Xlup

The zero sequence parameters shown in Table 6-6 have better TP and FP rates than the positive sequence set, giving an almost perfect ROC Area column overall. The precision in this group also has better rates, with no class below 90%. The parameter **Xlup** had the worst misclassification rate (Table 6-7), although the partition was not substantial unlike that in the positive sequence group.

It is very clear from the three sets that classification using the negative sequence information had the best classification rate for standard parameters. It is also interesting to note that all three sets had imperfect **Xlup** and **Tdsup** TP classification rates, with the positive sequence registering the worst rate for both parameters out of all groups.

B. Exciter Parameters

For the exciter parameter tests, the nominal classes used are shown in Table 6-8.

Table 6-8. Exciter Nominal Classes

Base	Default Simulation Setting
Trup	(Tr) measurement delay, upper range
Trdn	(Tr) measurement delay, lower range
Kaup	(Ka) controller gain, upper range
Kadn	(Ka) controller gain, lower range
Taup	(Ta) controller time constant, upper range
Tadn	(Ta) controller time constant, lower range
Keup	(Ke) excitor constant, upper range
Kedn	(Ke) excitor constant, lower range
Teup	(Te) excitor time constant, upper range
Kfup	(Kf) stabilization path gain, upper range
Tfup	(Tf) stabilization path time constant, upper range

B.1 Positive Sequence

Table 6-9. Positive Sequence Classification Accuracy for Exciter Parameters

=== Detailed Accuracy By Class ===						
TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.59	0.058	0.48	0.59	0.529	0.928	Base
0.85	0.007	0.914	0.85	0.881	0.976	Trup
0.7	0.04	0.614	0.7	0.654	0.941	Trdn
0.77	0.009	0.885	0.77	0.824	0.972	Kaup
0.77	0.009	0.885	0.77	0.824	0.977	Kadn
0.91	0	1	0.91	0.953	0.969	Taup
0.57	0.055	0.483	0.57	0.523	0.92	Tadn
0.91	0.003	0.968	0.91	0.938	0.99	Keup
0.95	0	1	0.95	0.974	0.994	Kedn
0.98	0.003	0.97	0.98	0.975	0.997	Teup
0.92	0	1	0.92	0.958	0.988	Kfup
0.99	0.005	0.943	0.99	0.966	1	Tfup
Weighted Avg. 0.826	0.016	0.845	0.826	0.833	0.971	

Table 6-10. Positive Sequence Confusion Matrix for Exciter Parameters

=== Confusion Matrix ===												
a	b	c	d	e	f	g	h	i	j	k	l	<-- classified as
59	0	9	1	0	0	31	0	0	0	0	0	a = Base
5	85	1	5	0	0	4	0	0	0	0	0	b = Trup
13	0	70	1	1	0	15	0	0	0	0	0	c = Trdn
10	4	2	77	1	0	5	0	0	0	0	1	d = Kaup
3	0	15	0	77	0	5	0	0	0	0	0	e = Kadn
2	0	1	0	1	91	0	1	0	1	0	3	f = Taup
29	1	9	1	1	0	57	0	0	0	0	2	g = Tadn
2	3	0	1	0	0	1	91	0	2	0	0	h = Keup
0	0	5	0	0	0	0	0	95	0	0	0	i = Kedn
0	0	0	0	0	0	0	2	0	98	0	0	j = Teup
0	0	2	0	6	0	0	0	0	0	92	0	k = Kfup
0	0	0	1	0	0	0	0	0	0	0	99	l = Tfup

For the positive sequence set (Table 6-9), a few parameters had very poor classification rates. The nominal classes **Base** and **Tadn** both had F-Measures below 60% due to low TP and Precision rates. The ROC Area for both classes are below 93%, which is the lowest value in the lot. The class **Trdn** also had a low F-measure rate at 65.4%. All of the other parameter classes had F-measures greater than 80% with **Taup**, **Kedn** and **Kfup** tallying perfect precisions. It can be observed from the confusion matrix (Table 6-10) that **Base** and **Tadn** had split large partitions, with one getting misclassified as the other. Other parameter classes with low classification rates, **Trdn** and **Kaup**, have sizable shares of instances going to **Base**. The spread of substantial misclassification appear to be limited to a few classes, which resulted to interacting drop in accuracy measures.

B.2 Negative Sequence

Table 6-11. Negative Sequence Classification Accuracy for Exciter Parameters

=== Detailed Accuracy By Class ===						
	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	1	0	1	1	1	1	Base
	0.94	0.012	0.879	0.94	0.908	0.995	Trup
	0.98	0.005	0.942	0.98	0.961	1	Trdn
	0.99	0.009	0.908	0.99	0.947	1	Kaup
	0.95	0.008	0.913	0.95	0.931	0.997	Kadn
	0.84	0	1	0.84	0.913	0.974	Taup
	0.99	0.002	0.98	0.99	0.985	1	Tadn
	0.98	0.001	0.99	0.98	0.985	1	Keup
	0.98	0	1	0.98	0.99	0.997	Kedn
	1	0.002	0.98	1	0.99	1	Teup
	0.93	0.002	0.979	0.93	0.954	0.998	Kfup
	0.96	0.001	0.99	0.96	0.975	0.998	Tfup
Weighted Avg.	0.962	0.003	0.963	0.962	0.962	0.997	

Table 6-12. Negative Sequence Confusion Matrix for Exciter Parameters

=== Confusion Matrix ===												
a	b	c	d	e	f	g	h	i	j	k	l	<-- classified as
100	0	0	0	0	0	0	0	0	0	0	0	a = Base
0	94	1	4	1	0	0	0	0	0	0	0	b = Trup
0	0	98	0	0	0	2	0	0	0	0	0	c = Trdn
0	1	0	99	0	0	0	0	0	0	0	0	d = Kaup
0	0	4	1	95	0	0	0	0	0	0	0	e = Kadn
0	8	0	1	1	84	0	1	0	2	2	1	f = Taup
0	0	1	0	0	0	99	0	0	0	0	0	g = Tadn
0	2	0	0	0	0	0	98	0	0	0	0	h = Keup
0	1	0	1	0	0	0	0	98	0	0	0	i = Kedn
0	0	0	0	0	0	0	0	0	100	0	0	j = Teup
0	1	0	0	6	0	0	0	0	0	93	0	k = Kfup
0	0	0	3	1	0	0	0	0	0	0	96	l = Tfup

Unlike the positive sequence set, the negative sequence classification shown in Table 6-11 had no trouble with the **Base** and **Tadn** classes. One class, **Teup**, had a perfect TP with a close to flawless FP giving a high F-measure value. The parameter class **Taup** had the lowest TP rate at 84%, but interestingly had a perfect precision. Most of the other parameter classes have very good F-measures, with **Trup** having the lowest precision at 87.9%. The confusion matrix (Table 6-12) reveals that a good portion of the instances belonging to **Taup** are spread to other classes, with

majority going to **Trup**. This particular misclassification reduced the **Taup** TP rate and **Trup**'s precision.

B.3 Zero Sequence

Table 6-13. Zero Sequence Classification Accuracy for Exciter Parameters

=== Detailed Accuracy By Class ===						
	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.61	0.062	0.473	0.61	0.533	0.94	Base
	0.94	0.006	0.931	0.94	0.935	0.976	Trup
	0.76	0.032	0.685	0.76	0.72	0.981	Trdn
	0.8	0.002	0.976	0.8	0.879	0.994	Kaup
	0.97	0.003	0.97	0.97	0.97	0.985	Kadn
	0.91	0.001	0.989	0.91	0.948	0.995	Taup
	0.4	0.058	0.385	0.4	0.392	0.917	Tadn
	1	0	1	1	1	1	Keup
	0.94	0.001	0.989	0.94	0.964	0.997	Kedn
	1	0	1	1	1	1	Teup
	0.97	0.001	0.99	0.97	0.98	0.989	Kfup
	0.88	0	1	0.88	0.936	0.995	Tfup
Weighted Avg.	0.848	0.014	0.866	0.848	0.855	0.981	

Table 6-14. Zero Sequence Confusion Matrix for Exciter Parameters

=== Confusion Matrix ===												
a	b	c	d	e	f	g	h	i	j	k	l	<-- classified as
61	0	13	0	0	0	26	0	0	0	0	0	a = Base
4	94	0	0	0	0	2	0	0	0	0	0	b = Trup
6	0	76	0	0	0	18	0	0	0	0	0	c = Trdn
9	0	1	80	0	0	10	0	0	0	0	0	d = Kaup
1	1	0	0	97	0	1	0	0	0	0	0	e = Kadn
0	6	0	0	1	91	0	0	1	0	1	0	f = Taup
46	0	14	0	0	0	40	0	0	0	0	0	g = Tadn
0	0	0	0	0	0	0	100	0	0	0	0	h = Keup
0	0	6	0	0	0	0	0	94	0	0	0	i = Kedn
0	0	0	0	0	0	0	0	0	100	0	0	j = Teup
0	0	0	0	2	1	0	0	0	0	97	0	k = Kfup
2	0	1	2	0	0	7	0	0	0	0	88	l = Tfup

The zero sequence classification set (Table 6-13) is similar to the positive sequence set, for having very low F-measures for parameter classes **Base** and **Tadn**. Both have incurred poor TP rates coupled with low precision. This mode of classification however, has perfectly classified **Keup** and **Teup**, which then surpasses the classifiers from the negative sequence group, but not by a considerable amount. The parameter class **Trdn** has the lowest precision after **Base** and **Tadn**.

As with the positive sequence matrix, the zero sequence set shows a substantial partitioning of instances as shown in Table 6-14, mainly between Base and **Tadn**. The parameter class **Trdn** also has big chunks of misclassification, mainly with **Tadn**.

The exciter parameter error classification tests overall were not as accurate as the standard parameter classification, but each of the sequence data sets has several parameter classes with very good accuracy rates. The negative sequence set has the most number of very accurate classes, and was the only one not to have problems classifying **Base** and **Tadn**, which were the two worst classified groups in the positive and zero sequence tests.

C. Governor Parameters

For the governor parameter tests, the nominal classes used are shown in Table 6-15.

Table 6-15. Governor Nominal Classes

Base	Default Simulation Setting
Rup	(R) permanent droop, upper range
Rdn	(R) permanent droop, lower range
Rtup	(r) temporary droop, upper range
Rtdn	(r) temporary droop, lower range
GTrup	(Tr) governor time constant, upper range
GTrdn	(Tr) governor time constant, lower range
GTfup	(Tf) filter time constant, upper range
GTfdn	(Tf) filter time constant, lower range
Twup	(Tw) water starting time, upper range
Atup	(At) turbine gain, upper range
Atdn	(At) turbine gain, lower range
Tgup	(Tg) servo time constant, upper range
Tgdn	(Tg) servo time constant, lower range

C.1 Positive Sequence

Table 6-16. Positive Sequence Classification Accuracy for Governor Parameters

=== Detailed Accuracy By Class ===						
TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.44	0.074	0.314	0.44	0.367	0.841	Base
0.35	0.047	0.365	0.35	0.357	0.804	Rup
0.41	0.063	0.333	0.41	0.368	0.858	Rdn
0.45	0.035	0.495	0.45	0.471	0.823	Rtup
0.66	0.021	0.71	0.66	0.684	0.878	Rtdn
0.51	0.046	0.459	0.51	0.483	0.874	GTrup
0.51	0.056	0.411	0.51	0.455	0.887	GTrdn
0.26	0.059	0.252	0.26	0.256	0.818	GTfup
0.31	0.052	0.313	0.31	0.312	0.866	GTfdn
0.23	0.035	0.338	0.23	0.274	0.761	Twup
0.33	0.028	0.478	0.33	0.391	0.8	Atup
0.62	0.042	0.534	0.62	0.574	0.896	Atdn
0.33	0.042	0.375	0.33	0.351	0.831	Tgup
0.37	0.032	0.468	0.37	0.413	0.812	Tgdn
Weighted Avg. 0.413	0.045	0.418	0.413	0.411	0.839	

Table 6-17. Positive Sequence Confusion Matrix for Governor Parameters

=== Confusion Matrix ===														
a	b	c	d	e	f	g	h	i	j	k	l	m	n	<-- classified as
44	1	12	4	0	0	1	15	17	2	0	1	1	2	a = Base
6	35	4	7	2	21	0	4	9	3	2	3	0	4	b = Rup
14	2	41	2	1	0	1	22	8	0	0	3	3	3	c = Rdn
3	4	3	45	0	7	2	3	4	5	3	13	5	3	d = Rtup
1	2	3	0	66	4	6	2	5	0	6	3	0	2	e = Rtdn
1	7	1	7	5	51	3	3	2	1	3	5	0	11	f = GTrup
2	0	6	1	0	2	51	0	0	12	2	8	16	0	g = GTrdn
21	3	29	1	1	1	0	26	9	1	1	3	2	2	h = GTfup
28	8	6	1	1	1	0	12	31	2	3	2	0	5	i = GTfdn
5	5	4	8	0	3	22	1	2	23	8	1	15	3	j = Twup
1	10	4	4	11	7	6	5	2	5	33	7	4	1	k = Atup
2	7	2	6	0	4	3	2	1	0	4	62	6	1	l = Atdn
3	2	6	1	0	1	27	5	3	10	1	3	33	5	m = Tgup
9	10	2	4	6	9	2	3	6	4	3	2	3	37	n = Tgdn

Governor parameter validation using positive sequence data shown in Table 6-16 is apparently a very poor choice for classification, as less than a handful of the parameter classes even have F-measures greater than 50% and not one ROC Area rating is higher than 90%. Only

one parameter, **Rtdn**, has an F-measure greater than 60%. This is followed by **Atdn** at 57.4%. The rest of the classes are well below 50%.

C.2 Negative Sequence

Table 6-18. Negative Sequence Classification Accuracy for Governor Parameters

=== Detailed Accuracy By Class ===						
TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.97	0.015	0.836	0.97	0.898	0.995	Base
0.84	0.016	0.8	0.84	0.82	0.984	Rup
0.98	0.02	0.79	0.98	0.875	0.994	Rdn
0.64	0.012	0.81	0.64	0.715	0.914	Rtup
0.62	0.008	0.861	0.62	0.721	0.935	Rtdn
0.95	0.008	0.905	0.95	0.927	0.992	GTrup
0.98	0.004	0.951	0.98	0.966	1	GTrdn
0.87	0.012	0.853	0.87	0.861	0.989	GTfup
0.97	0.004	0.951	0.97	0.96	1	GTfdn
0.74	0.005	0.925	0.74	0.822	0.957	Twup
0.81	0.025	0.711	0.81	0.757	0.97	Atup
0.78	0.019	0.757	0.78	0.768	0.983	Atdn
0.82	0.005	0.921	0.82	0.868	0.984	Tgup
0.97	0.007	0.915	0.97	0.942	1	Tgdn
Weighted Avg. 0.853	0.011	0.856	0.853	0.85	0.978	

Table 6-19. Negative Sequence Confusion Matrix for Governor Parameters

=== Confusion Matrix ===														
a	b	c	d	e	f	g	h	i	j	k	l	m	n	<-- classified as
97	0	0	0	0	0	0	0	3	0	0	0	0	0	| a = Base
0	84	0	1	0	5	0	1	0	0	0	9	0	0	| b = Rup
0	0	98	0	0	0	1	0	1	0	0	0	0	0	| c = Rdn
0	10	0	64	1	0	2	0	0	3	7	13	0	0	| d = Rtup
0	0	9	2	62	0	0	0	0	2	21	0	0	4	| e = Rtdn
3	1	0	0	0	95	0	1	0	0	0	0	0	0	| f = GTrup
0	0	2	0	0	0	98	0	0	0	0	0	0	0	| g = GTrdn
13	0	0	0	0	0	0	87	0	0	0	0	0	0	| h = GTfup
3	0	0	0	0	0	0	0	97	0	0	0	0	0	| i = GTfdn
0	2	0	3	6	0	0	4	0	74	1	0	6	4	| j = Twup
0	0	11	2	3	0	0	0	0	1	81	1	1	0	| k = Atup
0	8	0	5	0	5	0	0	0	0	3	78	0	1	| l = Atdn
0	0	1	2	0	0	2	9	1	0	1	2	82	0	| m = Tgup
0	0	3	0	0	0	0	0	0	0	0	0	0	97	| n = Tgdn

In the negative sequence classification set for governors (Table 6-18), although it has generally lower accuracy ratings than those in the standard and exciter parameters, still managed

to have classes with F-measures greater than 90%. Three parameter classes, **GTRdn**, **GTFdn** and **Tgdn**, have close to perfect ROC Area ratings. The lowest F-measure registered is that of **Rtup** at 71.5%, followed by **Rtdn** at 72.1%, even though the two parameter classes' TP ratings are of the reverse order. The confusion matrix (Table 6-19) shows that **Rtup** and **Rtdn** split a good amount of its misclassification into **Atdn** and **Atup** respectively. Parameter class **Gtfup** has a significant portion classified solely into Base, even though the reverse is not true.

C.3 Zero Sequence

Table 6-20. Zero Sequence Classification Accuracy for Governor Parameters

=== Detailed Accuracy By Class ===						
TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.34	0.091	0.224	0.34	0.27	0.796	Base
0.22	0.045	0.275	0.22	0.244	0.73	Rup
0.32	0.068	0.267	0.32	0.291	0.83	Rdn
0.35	0.062	0.302	0.35	0.324	0.853	Rtup
0.24	0.042	0.304	0.24	0.268	0.787	Rtdn
0.58	0.044	0.504	0.58	0.54	0.927	GTrup
0.31	0.071	0.252	0.31	0.278	0.8	GTrdn
0.24	0.071	0.207	0.24	0.222	0.759	GTfup
0.22	0.068	0.198	0.22	0.209	0.761	GTfdn
0.11	0.032	0.212	0.11	0.145	0.636	Twup
0.34	0.051	0.34	0.34	0.34	0.819	Atup
0.31	0.049	0.326	0.31	0.318	0.825	Atdn
0.13	0.033	0.232	0.13	0.167	0.637	Tgup
0.19	0.051	0.224	0.19	0.205	0.713	Tgdn
Weighted Avg. 0.279	0.055	0.276	0.279	0.273	0.777	

Table 6-21. Zero Sequence Confusion Matrix for Governor Parameters

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h  i  j  k  l  m  n   <-- classified as
 34  3 14  0  0  4  0 13 18  2  0  1  6  5 |  a = Base
  8 22  7  5  4  3 10  6  8 11  2 12  1  1 |  b = Rup
 14  1 32  1  4  3  1 16 11  1  1  2  1 12 |  c = Rdn
  1  8  1 35  6  1 20  2  0  2 14  9  1  0 |  d = Rtup
  4  6  5  3 24  7  9  3  2  3 19  8  4  3 |  e = Rtdn
  5  1  1  2  6 58  0  4  3  1  0  3  2 14 |  f = GTrup
  3  7  0 24  1  1 31  0  2  4 11 10  4  2 |  g = GTrdn
 15  6 15  2  2  3  2 24 22  2  2  0  0  5 |  h = GTfup
 27  2 16  0  2  3  0 14 22  1  0  0  5  8 |  i = GTfdn
  9  8  5  5  6  6 13 10  5 11  5  5  7  5 |  j = Twup
  1  1  5 11 15  1 13  5  2  0 34  9  0  3 |  k = Atup
  1  5  0 19  7  7 14  0  0  2  9 31  5  0 |  l = Atdn
 11  9  7  6  1  5  9  7  7 10  2  5 13  8 |  m = Tgup
 19  1 12  3  1 13  1 12  9  2  1  0  7 19 |  n = Tgdn
```

The zero sequence set (Table 6-20) for classification was worse than the positive sequence set. Only **Gtrup** had an F-measure greater than 50%. The average TP and Precision rates were around 27%.

It is very obvious from the governor classification set that the negative sequence set is the only reliable data source for classifying governor parameter errors. The positive and zero sequence sets have absolutely no redeeming features that will be more useful than that of the negative sequence set, which, although not completely accurate across the board, still has a few F-measure values that are well above 90%.

D. Conclusion

In summary, out of all the sequence data used for classification, the negative sequence set has consistently produced the best parameter error classifier. It has almost perfectly identified every parameter class in the standard generator group. The result of which was only barely accomplished using the zero sequence classification set. In the exciter parameters group, the negative sequence classification set's greatest advantage was its very accurate classification of the classes **Base** and **Tadn**, on top of its excellent F-measure ratings all throughout. There are however a few select classes from either the positive or zero sequence groups which were distinctly better

than the negative set. The positive sequence classification has the best accuracy ratings for **Tfup** and **Taup**, and the zero sequence performed well classifying **Kfup**, **Kadn** and **Trup**. In such cases, the model parameters can be tuned according to the sequence data set which yields the most accurate classification. For the governor parameters group, it is very clear that only the negative sequence data produced useable results.

The high accuracy of the negative sequence-derived classifiers may be attributed to the unbalanced nature of the fault being imposed on the generator, which normally produces a balanced output. The specific configuration used in the system (Figure 5-3) allows for a continuous delivery of three-phase power even after the faulted phase is cleared, due to the presence of a second line L_pp1b, parallel to the faulted line. However, the unbalance will remain, as the effective impedance will have changed upon removal of a phase. This irregularity in impedance will be highly prominent immediately after the fault, and can thus have an immediate effect on the machine. The nature of the short circuit event is deemed to make the parameter changes more pronounced in negative sequence data. Nevertheless, as of this study, a thorough analysis of the individual predictor variables used in the classifiers has yet to be made to verify the theory.

6.2.2 Classification for Combined Classes

A. Single Line-to-Ground Case

The accuracy rates of the all the parameter classes for the single line-to-ground (SLG) case shown in Table 6-22 are very similar to the individual tests for the standard, exciter and governor models. It can be inferred from the results that, in general, the classifier is still able to pick unique patterns for each of the nominal classes in spite of the additional selection of parameters, thus keeping the accuracy values virtually unchanged. The subset of standard parameters contain the most number of very accurate ratings. The classes **Tdssdn** and **Xqssdn** have attained perfect classification. The F-measure ratings are all greater than 90% due to good TP and precision rates. Not counting the **Base**, which for this test is shared by all parameter classes, the only class with a precision below 90% is **Xlup**. The subgroup of exciter parameters did not have a flawlessly classified parameter class, but all except **Taup** have greater than 90% F-measure. Unlike the standard parameter set, none of the precision ratings are less than 90%. The governor subgroup did not do as well as the standard and exciter parameter set, but it still managed to fairly good F-measure ratings, the top two being **Gtrdn** and **Tgdn**, with F-measures of 97.5% and 95% respectively. The two worst performers were temporary droop pair **Rtup** and **Rtdn**, obtaining F-measures below 70%.

The confusion matrix in Table 6-23 shows that the standard and exciter class parameters hardly misclassify into other groups. Instances leaking into other groups typically appear sporadically in ones and twos. The exciter parameter **Taup**, however, misclassified a good number of instances into standard parameter's **Xlup**, contributing to its reduction in precision, and into its own **Trup**. The governor had the most number of misclassifications, most notably from **Rtup** and **Rtdn**, with a sizable partition going into **Atdn** and **Atup** respectively. There were other significant misclassifications in the governor group of parameter classes, but it is very interesting to note that these are only local to the group, with the exception of **Gtfup** giving a good number of instances to Base, which impacted its precision. Overall, the classification of parameter errors in a group containing a large number of classes has hardly had an effect on its accuracy ratings, and in this respect can possibly qualify it as a robust method of error identification.

Table 6-22. Combined-Parameters Single Line-to-Ground Case Accuracy Table

=== Detailed Accuracy By Class ===						
TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.97	0.005	0.851	0.97	0.907	0.998	Base
0.98	0.001	0.951	0.98	0.966	0.998	Xdup
0.97	0	1	0.97	0.985	0.997	Xddn
1	0	0.99	1	0.995	1	Xqup
0.98	0	0.99	0.98	0.985	1	Xqdn
1	0	0.99	1	0.995	1	Tdsup
1	0.001	0.971	1	0.985	1	Xdsup
1	0.001	0.971	1	0.985	1	Xdsdn
1	0	1	1	1	1	Tdssdn
0.99	0	1	0.99	0.995	1	Tqssdn
1	0	0.99	1	0.995	1	Xdssdn
1	0	1	1	1	1	Xqssdn
0.97	0.003	0.898	0.97	0.933	0.998	Xlup
0.94	0.002	0.922	0.94	0.931	0.997	Trup
0.98	0.002	0.942	0.98	0.961	1	Trdn
0.98	0.001	0.951	0.98	0.966	1	Kaup
0.91	0	0.989	0.91	0.948	0.996	Kadn
0.84	0.001	0.966	0.84	0.898	0.979	Taup
1	0.001	0.98	1	0.99	1	Tadn
0.95	0	0.99	0.95	0.969	0.999	Keup
0.96	0.001	0.98	0.96	0.97	0.996	Kedn
0.98	0.001	0.97	0.98	0.975	0.999	Teup
0.96	0.001	0.98	0.96	0.97	0.998	Kfup
0.94	0	0.989	0.94	0.964	0.997	Tfup
0.79	0.004	0.849	0.79	0.819	0.994	Rup
0.99	0.007	0.805	0.99	0.888	0.992	Rdn
0.6	0.004	0.811	0.6	0.69	0.971	Rtup
0.58	0.003	0.829	0.58	0.682	0.956	Rtdn
0.96	0.004	0.857	0.96	0.906	0.999	GTrup
0.98	0.001	0.97	0.98	0.975	1	GTrdn
0.88	0.004	0.863	0.88	0.871	0.996	GTfup
0.98	0.003	0.916	0.98	0.947	1	GTfdn
0.69	0.001	0.958	0.69	0.802	0.979	Twup
0.81	0.013	0.633	0.81	0.711	0.989	Atup
0.83	0.009	0.709	0.83	0.765	0.994	Atdn
0.86	0.002	0.915	0.86	0.887	0.994	Tgup
0.95	0.001	0.95	0.95	0.95	1	Tgdn
Weighted Avg. 0.924	0.002	0.928	0.924	0.923	0.995	

Table 6-23. Combined-Parameters Single Line-to-Ground Case Confusion Matrix

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	aa	ab	ac	ad	ae	af	ag	ah	ai	aj	ak	<-- classifie
97	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	a = Base
0	98	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	b = Xdup
0	0	97	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	c = Xddn
0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	d = Xqup
0	0	0	0	98	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	e = Xqdn
0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	f = Tdsup
0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	g = Xdsup
0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	h = Xdsdn
0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	i = Tdssdn
0	0	0	0	0	0	1	0	0	99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	j = Tqssdn
0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	k = Xdssdn
0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	l = Xqssdn
0	0	0	1	0	0	0	0	0	0	1	0	97	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	m = Xlup
0	0	0	0	0	0	0	0	0	0	0	0	3	94	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	n = Trup
0	0	0	0	0	0	0	0	0	0	0	0	0	0	98	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	o = Trdn
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	98	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	p = Kaup
0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	91	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	0	2	0	0	0	q = Kadn
0	1	0	0	0	1	0	0	0	0	0	0	6	4	0	0	0	84	0	0	0	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	r = Taup
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	s = Tadn
0	2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	95	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	t = Keup
0	0	0	0	0	0	1	0	0	0	0	0	0	2	0	0	0	0	0	0	96	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	u = Kedn
0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	98	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	v = Teup
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	96	0	0	0	0	0	0	0	0	0	0	1	0	0	0	w = Kfup
0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	94	0	0	0	0	0	0	0	0	0	2	0	1	0	x = Tfup
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	79	0	0	0	8	0	1	0	0	0	12	0	0	y = Rup
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	99	0	0	0	0	0	1	0	0	0	0	0	z = Rdn
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	60	0	1	0	0	0	2	14	20	0	0	aa = Rtup
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	3	0	0	0	0	1	0	1	5	2	58	0	2	0	0	1	22	0	1	3	ab = Rtdn
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	96	0	0	0	0	0	0	0	0	ac = GTrup
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	98	0	0	0	0	0	0	0	ad = GTrdn
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	88	0	0	0	0	0	0	ae = GTfup
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	98	0	0	0	0	0	af = GTfdn
0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	4	0	0	0	0	0	0	0	0	1	0	5	4	0	0	3	0	69	4	2	5	1	ag = Twup
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	1	6	0	0	0	0	0	81	0	0	0	ah = Atup
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	4	0	5	0	0	0	0	1	83	0	0	ai = Atdn
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	1	9	0	0	1	0	86	0	aj = Tgup
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	95	ak = Tgdn

B. Double Line-to-Ground Case

The double line-to-ground (DLG) parameter error classification case (Table 6-24) was able to produce a similar accuracy performance as the single line-to-ground case, where the standard parameter set has the best accuracy ratings, followed by the exciter, then the governor. For this event, the standard parameter has more parameter classes with perfect accuracy ratings. It has F-measure ratings all greater than 90%, but as with the SLG set, barring the Base parameter class, all the precision ratings are above 90% except for **Xlup**. The exciter group, like that of the SLG case, has mostly over 90% F-measure ratings, but this time two parameters, **Trdn** and **Tadn**, did not make this cut. The parameter class **Trdn** has the lowest TP rate in the exciter group, although it has a very good precision. **Tadn**, on the other hand, had a perfect TP rate, but with a relatively poor precision at 78.1%. The governor group in the DLG set had more weakly classified parameter errors than in the SLG set. Aside from the temporary droop pair **Rtup** and **Rtdn**, the turbine gain pair **Atup** and **Atdn**, and class **Twup** all had less than 70% F-measure rates.

The standard parameter group in the confusion matrix (Table 6-25) hardly misclassifies into other groups, but in one parameter class **Xdup**, a small part of its instances crossed into exciter's **Keup**. The exciter parameter group had a few notable misclassifications. The biggest one is **Trdn** parting with a substantial number of instances to **Tadn**. There were also three parameter classes, **Trup**, **Taup** and **Teup** contributing to the decrease in the precision of the standard parameter group's **Xlup**. The governor parameter group once again shows that **Rtup** and **Rtdn** has sizable partitions going into **Atdn** and **Atup** respectively. The class parameter **Atup** has shown to be a very weak class as it has instances corrupting several parameter classes, largely **Rdn**, **Rtdn** and **Tgup**. **Atdn** also has a big portion of its classifications going into **Rup**. As with the SLG case, most of the misclassifications are just local to the governor group, thus maintaining the integrity of the accuracy rates of the classes in the standard and exciter group.

Table 6-24. Combined-Parameters Double Line-to-Ground Case Accuracy Table

=== Detailed Accuracy By Class ===						
TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
1	0.003	0.893	1	0.943	0.999	Base
0.95	0.001	0.979	0.95	0.964	0.998	Xdup
0.97	0.001	0.98	0.97	0.975	0.999	Xddn
1	0	1	1	1	1	Xqup
1	0	1	1	1	1	Xqdn
0.98	0	1	0.98	0.99	1	Tdsup
1	0	0.99	1	0.995	1	Xdsup
1	0	1	1	1	1	Xdsdn
1	0	1	1	1	1	Tdssdn
1	0.001	0.98	1	0.99	1	Tqssdn
1	0	1	1	1	1	Xdssdn
1	0	1	1	1	1	Xqssdn
0.99	0.004	0.868	0.99	0.925	0.999	Xlup
0.92	0.001	0.948	0.92	0.934	0.997	Trup
0.72	0.002	0.923	0.72	0.809	0.996	Trdn
0.98	0.003	0.899	0.98	0.938	1	Kaup
0.92	0	0.989	0.92	0.953	0.994	Kadn
0.93	0	0.989	0.93	0.959	0.997	Taup
1	0.008	0.781	1	0.877	0.997	Tadn
0.98	0.002	0.925	0.98	0.951	0.998	Keup
0.95	0	0.99	0.95	0.969	0.997	Kedn
0.93	0.001	0.969	0.93	0.949	0.997	Teup
0.93	0	1	0.93	0.964	0.985	Kfup
1	0.001	0.952	1	0.976	1	Tfup
0.78	0.01	0.684	0.78	0.729	0.986	Rup
0.92	0.009	0.73	0.92	0.814	0.996	Rdn
0.53	0.007	0.679	0.53	0.596	0.962	Rtup
0.47	0.008	0.635	0.47	0.54	0.939	Rtdn
0.98	0.005	0.845	0.98	0.907	1	GTrup
0.93	0.004	0.853	0.93	0.89	0.996	GTrdn
0.95	0.003	0.896	0.95	0.922	0.999	GTfup
0.92	0	0.989	0.92	0.953	0.998	GTfdn
0.5	0.003	0.806	0.5	0.617	0.961	Twup
0.45	0.009	0.592	0.45	0.511	0.945	Atup
0.7	0.011	0.642	0.7	0.67	0.984	Atdn
0.79	0.009	0.699	0.79	0.742	0.99	Tgup
0.93	0.004	0.877	0.93	0.903	0.99	Tgdn
Weighted Avg. 0.892	0.003	0.892	0.892	0.888	0.992	

Table 6-25. Combined-Parameters Double Line-to-Ground Case Confusion Matrix

```
=== Confusion Matrix ===

   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o   p   q   r   s   t   u   v   w   x   y   z  aa  ab  ac  ad  ae  af  ag  ah  ai  aj  ak   <-- classifie
 100   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  a = Base
   0  95   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  b = Xdup
   0   0  97   0   0   0   0   0   0   1   0   0   0   0   0   0   1   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  c = Xddn
   0   0   0 100   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  d = Xqup
   0   0   0   0 100   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  e = Xqdn
   0   1   0   0   0  98   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  f = Tdsup
   0   0   0   0   0   0 100   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  g = Xdsup
   0   0   0   0   0   0   0 100   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  h = Xdsdn
   0   0   0   0   0   0   0   0 100   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  i = Tdssdn
   0   0   0   0   0   0   0   0   0 100   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  j = Tqssdn
   0   0   0   0   0   0   0   0   0   0 100   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  k = Xdssdn
   0   0   0   0   0   0   0   0   0   0   0 100   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  l = Xqssdn
   0   0   0   0   0   0   0   0   0   0   0   0  99   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  m = Xlup
   0   0   0   0   0   0   0   0   0   0   0   0   4  92   0   1   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   1 |  n = Trup
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  72   0   0   0  28   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  o = Trdn
   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0  98   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  p = Kaup
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2   0  92   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1   0   0   0   0   4 |  q = Kadn
   0   0   0   0   0   0   0   0   0   0   0   0   4   0   0   0   0  93   0   0   0   3   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  r = Taup
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 100   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  s = Tadn
   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0  98   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  t = Keup
   0   0   2   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0  95   0   0   0   0   0   0   0   0   0   0   0   1   0   0   1   0 |  u = Kedn
   0   0   0   0   0   0   0   0   0   0   0   0   7   0   0   0   0   0   0   0   0  93   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |  v = Teup
   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0  93   1   0   0   1   1   0   1   0   0   0   1   0   0   0 |  w = Kfup
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 100   0   0   0   0   0   0   0   0   0   0   0   0   0 |  x = Tfup
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  78   0   3   1   5   1   1   0   0   1  10   0   0 |  y = Rup
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  92   0   0   0   6   1   0   0   1   0   0   0 |  z = Rdn
   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   2   0   0   0   0   0   0   0   0   7   1  53   3   3   2   2   1   3   4  13   2   3 |  aa = Rtup
   0   0   0   0   0   0   0   0   0   0   0   0   0   2   0   1   0   0   0   0   0   0   0   0   4   1  10   6  47   2   1   0   0   3  14   5   4   0 |  ab = Rtdn
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  98   0   2   0   0   0   0   0   0 |  ac = GTrup
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   4   0   0   0  93   0   0   0   0   0   2   0 |  ad = GTrdn
   5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  95   0   0   0   0   0   0 |  ae = GTfup
   7   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1  92   0   0   0   0   0 |  af = GTfdn
   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1   2   0   0   0   0   0   0   0   0   3   4   6   7   1   1   1   0  50   5   3  15   0 |  ag = Twup
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   2   0   0   0   0   0   0   0   0   5  13   2  10   4   1   0   0   3  45   4   9   1 |  ah = Atup
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0  17   0   5   1   2   1   0   0   0   1  70   1   1 |  ai = Atdn
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   2   1   1   2   1   0   2   0   1   4   3  79   3 |  aj = Tgup
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1   1   2   0   1   0   0   0   0   1   0  93 |  ak = Tgdn
```

6.2.3 Three-Phase Short Circuit Tests

As can be seen in Table 6-26, in spite of the removal of the current from the set of predictor variables, majority of the standard parameter classes have decent accuracy rates. Two parameter classes, **Xdsup** and **Tqssdn**, were able to register perfect TP rates, with only minor flaws in precision. The rest of the classes have at least an 80% F-measure rate. The error rates for standard parameters in this test are comparable to the positive sequence results from the isolated SLG standard parameter test, with some minor reduction in accuracies. For the exciter parameter subset, the two parameters **Kaup** and **Kadn** have notably higher TP rates, than the isolated SLG case. This can possibly be attributed to the difference in fault types, in which the two concerned parameter classes may be slightly more detectable for a three-phase fault. The class **Tadn**, however, has consistently performed poorly for the two tests. It is no surprise that for the governor parameter subset, all the classes are hardly useful for validation. The highest F-measure rate comes from **Rtdn** at 74.7%, which, on the down side only has a 62% TP rate. The **Base** class registered a very low F-measure rate, which can be attributed to major misclassifications from the governor subset, as can be seen from the confusion matrix found in Appendix D.1.

The second case (Table 6-27) notably has better accuracy rates than the first. For the standard parameter set, all but two classes have F-measure rates above 90%. The parameter **Xqup** registered a perfect accuracy rate, followed by **Tdssdn** and **Tqssdn**, both of which only suffered minor reduction in precision. For the exciter set, majority of the parameter classes have better than 90% F-measure rates. The worst performer, quite surprisingly, is **Trdn**, with an F-measure of 77.1%. The class **Tadn** this time had a more decent accuracy, but is not far from **Trdn**'s accuracy rates. The governor set once again has the worst accuracies. However, one parameter, **Twup**, registered an 87.1% F-measure rate. This may again be attributable to the fault type, which can make some otherwise poor-performing classes stand out. In spite of the general overall improvement in performance for the second case, the **Base** class had a worse accuracy rate than the first test, owing to larger misclassifications from the governor set (see Appendix D.2).

Table 6-26. Combined-Parameters Three-Phase Short Circuit Case (No Current) Accuracy Table

=== Detailed Accuracy By Class ===						
TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.49	0.027	0.338	0.49	0.4	0.946	Base
0.83	0.001	0.943	0.83	0.883	0.997	Xdup
0.87	0.006	0.806	0.87	0.837	0.994	Xddn
0.98	0.001	0.98	0.98	0.98	1	Xqup
0.88	0	0.989	0.88	0.931	0.998	Xqdn
0.98	0	1	0.98	0.99	1	Tdsup
1	0.003	0.917	1	0.957	1	Xdsup
0.97	0	1	0.97	0.985	1	Xdsdn
0.97	0.001	0.97	0.97	0.97	0.999	Tdssdn
1	0.001	0.971	1	0.985	1	Tqssdn
0.99	0	1	0.99	0.995	1	Xdssdn
0.99	0.001	0.952	0.99	0.971	1	Xqssdn
0.89	0.007	0.774	0.89	0.828	0.992	Xlup
0.85	0.006	0.787	0.85	0.817	0.973	Trup
0.7	0.011	0.636	0.7	0.667	0.966	Trdn
0.83	0.005	0.822	0.83	0.826	0.988	Kaup
0.83	0.003	0.892	0.83	0.86	0.976	Kadn
0.91	0	1	0.91	0.953	0.985	Taup
0.49	0.014	0.49	0.49	0.49	0.939	Tadn
0.91	0.004	0.867	0.91	0.888	0.997	Keup
0.81	0.003	0.9	0.81	0.853	0.991	Kedn
0.92	0	0.989	0.92	0.953	0.996	Teup
0.91	0	0.989	0.91	0.948	0.989	Kfup
0.88	0.006	0.8	0.88	0.838	0.994	Tfup
0.34	0.019	0.337	0.34	0.338	0.924	Rup
0.23	0.013	0.338	0.23	0.274	0.93	Rdn
0.26	0.021	0.252	0.26	0.256	0.937	Rtup
0.62	0.001	0.939	0.62	0.747	0.937	Rtdn
0.35	0.014	0.417	0.35	0.38	0.954	GTrup
0.4	0.022	0.336	0.4	0.365	0.952	GTrdn
0.33	0.018	0.337	0.33	0.333	0.932	GTfup
0.36	0.025	0.283	0.36	0.317	0.939	GTfdn
0.7	0.01	0.667	0.7	0.683	0.962	Twup
0.61	0.012	0.587	0.61	0.598	0.963	Atup
0.26	0.02	0.268	0.26	0.264	0.939	Atdn
0.15	0.014	0.234	0.15	0.183	0.934	Tgup
0.64	0.014	0.552	0.64	0.593	0.964	Tgdn
Weighted Avg. 0.706	0.008	0.712	0.706	0.706	0.973	

Table 6-27. Combined-Parameters Three-Phase Short Circuit Case (With Current) Accuracy Table

=== Detailed Accuracy By Class ===						
TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.42	0.024	0.323	0.42	0.365	0.96	Base
0.84	0	0.988	0.84	0.908	0.998	Xdup
0.84	0.004	0.857	0.84	0.848	0.993	Xddn
1	0	1	1	1	1	Xqup
0.9	0	1	0.9	0.947	0.999	Xqdn
0.99	0	1	0.99	0.995	0.999	Tdsup
1	0.002	0.935	1	0.966	1	Xdsup
0.95	0	0.99	0.95	0.969	1	Xdsdn
1	0.001	0.971	1	0.985	1	Tdssdn
1	0	0.99	1	0.995	1	Tqssdn
0.99	0	0.99	0.99	0.99	1	Xdssdn
0.99	0.002	0.943	0.99	0.966	1	Xqssdn
0.98	0.007	0.79	0.98	0.875	0.998	Xlup
0.91	0.001	0.968	0.91	0.938	0.993	Trup
0.79	0.007	0.752	0.79	0.771	0.984	Trdn
0.97	0.002	0.933	0.97	0.951	0.998	Kaup
0.81	0.001	0.953	0.81	0.876	0.99	Kadn
0.92	0	1	0.92	0.958	0.994	Taup
0.81	0.006	0.779	0.81	0.794	0.993	Tadn
0.85	0.004	0.85	0.85	0.85	0.995	Keup
0.92	0.003	0.893	0.92	0.906	0.998	Kedn
0.96	0.001	0.98	0.96	0.97	1	Teup
0.92	0	1	0.92	0.958	0.994	Kfup
0.94	0.001	0.969	0.94	0.954	0.992	Tfup
0.39	0.015	0.415	0.39	0.402	0.934	Rup
0.35	0.017	0.368	0.35	0.359	0.94	Rdn
0.44	0.021	0.364	0.44	0.398	0.955	Rtup
0.66	0	1	0.66	0.795	0.947	Rtdn
0.51	0.018	0.443	0.51	0.474	0.957	GTrup
0.37	0.016	0.398	0.37	0.383	0.96	GTrdn
0.42	0.022	0.35	0.42	0.382	0.945	GTfup
0.3	0.027	0.234	0.3	0.263	0.941	GTfdn
0.81	0.001	0.942	0.81	0.871	0.968	Twup
0.47	0.009	0.588	0.47	0.522	0.953	Atup
0.55	0.026	0.367	0.55	0.44	0.961	Atdn
0.13	0.011	0.245	0.13	0.17	0.934	Tgup
0.66	0.006	0.759	0.66	0.706	0.977	Tgdn
Weighted Avg. 0.75	0.007	0.766	0.75	0.754	0.98	

6.3 Multi-Parameter Error Experiment

6.3.1 Introduction

The multi-parameter error test evaluates each parameter class' detectability in the presence of other active errors by defining a dedicated classifier for each previously defined class. The subject classifiers have a simple binary output of 'on' and 'off', which corresponds to an active and inactive subject parameter error respectively, run at the same time as the other selected random parameter errors. The test is first run with all the errors active, then switches off errors one at a time to determine the parameter whose removal causes the largest increase in accuracy. Once this is determined, the selected parameter is permanently switched off, and another series of tests is run to reveal the next parameter with the biggest effect on accuracy. This is continued until only two parameter errors remain, or when all error rates reach 0%, whichever comes first. The measure of accuracy used in the test is the out-of-bag (OOB) error rate computed by Weka when generating a classifier.

Xqup ← Subject Parameter								Ave	Gain (%)	Contribution (%)
All	**0.2450** ← Default Error Rate (All on)							0.2450	0.0000	0.0000
	Xd	Tds	Xds	Tdss	Tqss	Xdss	Xqss			
Set A	0.2575	0.2375	0.2025	0.1525	0.1925	0.1975	0.2525	0.2132	12.9738	12.9738
	Xd	Tds	Xds	Tqss	Xdss	Xqss				
Set B	0.2225	0.2750	0.1675	0.1750	0.1875	0.2000		0.2046	16.4966	3.5228
	Xd	Tds	Tqss	Xdss	Xqss					
Set C	0.1575	0.2425	0.1275	0.0875	0.1100			0.1450	40.8163	24.3197
	Xd	Tds	Tqss	Xqss						
Set D	0.0550	0.0875	0.0350	0.0575				0.0588	76.0204	35.2041
	Xd	Tds	Xqss							
Set E	0.0225	0.0925	0.0225					0.0458	81.2925	5.2721
	Tds	Xqss								
Set F	0.0025	0.0175						0.0100	95.9184	14.6259

Blue Highlight: Lowest OOB error of set

Green Highlight: Duplicate low OOB rate

Figure 6-3. Sample Test Result

A sample table of results is given in Figure 6-3 . At the top left corner, the table indicates the parameter under test, or the subject parameter. Under this label is its default classifier error rate, which is the rate for a classifier generated with all errors active. Below the default rate are the results for the series of classifiers with eliminated parameters. It is important to note that each item identified in a set is a unique classifier, with the specified eliminated error as label. The test classifier highlighted in blue is the lowest OOB in a set, which is eliminated in the next set below

it. Sometimes there would be more than one low OOB rate, and this is indicated by the green highlight. The test classifiers highlighted in green are not selected for elimination in the next set. The ***Ave*** column contains the average OOB rate for each set. The ***Gain*** column indicates the difference in percent of the set average from the default set average. Finally, the column ***Contribution*** is the difference of the gain value of one set from the previous set.

Due to the large amount of simulations required to accomplish the task, this test only covers the standard generator parameters. The parameter Xlup was eliminated to extend the dynamic range of the subtransient parameters in the simulation. Moreover, note that the parameter errors in each set accounts for the whole range of the parameter error, and no 'up' and 'down' separation is made. Each classifier in the multiple-error experiment was generated using 200 random forest trees from the best performing sequence information, which is the negative sequence data. Also, classifications were all based on a single line-to-ground disturbance.

6.3.2 Results

Table 6-28. Xdup Test

Xdup								**Ave**	**Gain (%)**	**Contribution (%)**
All	**0.5275**							0.5275	0.0000	0.0000
	Xq	Tds	Xds	Tdss	Tqss	Xdss	Xqss			
Set A	0.4525	0.3650	0.4775	0.5150	0.4975	0.4950	0.4825	0.4693	11.0359	11.0359
	Xq	Xds	Tdss	Tqss	Xdss	Xqss				
Set B	0.3700	0.2825	0.3000	0.2975	0.3600	0.3875		0.3329	36.8878	25.8520
	Xq	Tdss	Tqss	Xdss	Xqss					
Set C	0.1925	0.2550	0.1750	0.2475	0.2800			0.2300	56.3981	19.5103
	Xq	Tdss	Xdss	Xqss						
Set D	0.0500	0.1200	0.1375	0.1425				0.1125	78.6730	22.2749
	Tdss	Xdss	Xqss							
Set E	0.0250	0.0450	0.0325					0.0342	93.5229	14.8499
	Xdss	Xqss								
Set F	0.0175	0.0050						0.0113	97.8673	4.3444

The parameter class **Xdup** has a very high out-of-bag error rate with all parameter errors active. It becomes reasonably detectable only after the removal of four errors, as shown on set D, with the error rate less than 15%. The next level down further drops the error to less than 5%.

Table 6-29. Xddn Test

Xddn								Ave	Gain (%)	Contribution (%)
All	**0.4500**							0.4500	0.0000	0.0000
	Xq	Tds	Xds	Tdss	Tqss	Xdss	Xqss			
Set A	0.4100	0.2925	0.4225	0.4600	0.4500	0.4825	0.5150	0.4332	3.7302	3.7302
	Xq	Xds	Tdss	Tqss	Xdss	Xqss				
Set B	0.2125	0.2075	0.1750	0.1600	0.2400	0.2400		0.2058	54.2593	50.5291
	Xq	Xds	Tdss	Xdss	Xqss					
Set C	0.1250	0.1150	0.1950	0.1900	0.2025			0.1655	63.2222	8.9630
	Xq	Tdss	Xdss	Xqss						
Set D	0.0475	0.0725	0.0725	0.1450				0.0844	81.2500	18.0278
	Tdss	Xdss	Xqss							
Set E	0.0275	0.0400	0.0325					0.0333	92.5926	11.3426
	Xdss	Xqss								
Set F	0.0300	0.0050						0.0175	96.1111	3.5185

As with class **Xdup**, **Xddn** has a high out-of-bag error rate with all parameter errors active. However, by set D, most of the error rates are less than 10%. A big improvement in accuracy can be seen at the second removal of error at set B, which on average causes a 50% jump in error reduction. It is very clear that the removal of **Tds** from set A contributed largely to this accuracy increase.

Table 6-30. Xqup Test

Xqup								Ave	Gain (%)	Contribution (%)
All	**0.2450**							0.2450	0.0000	0.0000
	Xd	Tds	Xds	Tdss	Tqss	Xdss	Xqss			
Set A	0.2575	0.2375	0.2025	0.1525	0.1925	0.1975	0.2525	0.2132	12.9738	12.9738
	Xd	Tds	Xds	Tqss	Xdss	Xqss				
Set B	0.2225	0.2750	0.1675	0.1750	0.1875	0.2000		0.2046	16.4966	3.5228
	Xd	Tds	Tqss	Xdss	Xqss					
Set C	0.1575	0.2425	0.1275	0.0875	0.1100			0.1450	40.8163	24.3197
	Xd	Tds	Tqss	Xqss						
Set D	0.0550	0.0875	0.0350	0.0575				0.0588	76.0204	35.2041
	Xd	Tds	Xqss							
Set E	0.0225	0.0925	0.0225					0.0458	81.2925	5.2721
	Tds	Xqss								
Set F	0.0025	0.0175						0.0100	95.9184	14.6259

The parameter class **Xqup** has a moderate detectability with maximum active errors. By set C the removal of the third parameter error in one test, **Xdss**, resulted to a less than 10% out-of-bag error rate. By set D, all of the error rates are already well within 10%.

Table 6-31. Xqdn Test

Xqdn								Ave	Gain (%)	Contribution (%)
All	**0.1350**							0.1350	0.0000	0.0000
	Xd	Tds	Xds	Tdss	Tqss	Xdss	Xqss			
Set A	0.1400	0.1700	0.0950	0.1125	0.1025	0.1225	0.1025	0.1207	10.5820	10.5820
	Xd	Tds	Tdss	Tqss	Xdss	Xqss				
Set B	0.0925	0.1300	0.0725	0.0775	0.0575	0.0800		0.0850	37.0370	26.4550
	Xd	Tds	Tdss	Tqss	Xqss					
Set C	0.0425	0.0750	0.0575	0.0400	0.0750			0.0580	57.0370	20.0000
	Xd	Tds	Tdss	Xqss						
Set D	0.0425	0.0675	0.0675	0.0175				0.0488	63.8889	6.8519
	Xd	Tds	Tdss							
Set E	0.0325	0.0925	0.0175					0.0475	64.8148	0.9259
	Xd	Tds								
Set F	0.0100	0.0450						0.0275	79.6296	14.8148

The class **Xqdn** has an exceptionally low out-of-bag error rate for simultaneously active parameter errors. At set A, the error rates are already less than 20%, with one of the tests going as low as 9.5%. By set C all the error rates are less than 10%.

Table 6-32. Tdsup Test

Tdsup								Ave	Gain (%)	Contribution (%)
All	**0.2725**							0.2725	0.0000	0.0000
	Xd	Xq	Xds	Tdss	Tqss	Xdss	Xqss			
Set A	0.1425	0.1900	0.2025	0.2100	0.1625	0.2125	0.2350	0.1936	28.9646	28.9646
	Xq	Xds	Tdss	Tqss	Xdss	Xqss				
Set B	0.0900	0.1125	0.1525	0.0900	0.1700	0.1150		0.1217	55.3517	26.3871
	Xds	Tdss	Tqss	Xdss	Xqss					
Set C	0.0475	0.0975	0.0225	0.0775	0.0925			0.0675	75.2294	19.8777
	Xds	Tdss	Xdss	Xqss						
Set D	0.0000	0.0125	0.0050	0.0225				0.0100	96.3303	21.1009
	Tdss	Xdss	Xqss							
Set E	0.0000	0.0100	0.0000					0.0033	98.7768	2.4465
Set F								0.0000	100.0000	1.2232

Although **Tdsup** only has a moderately low out-of-bag error rate in the presence of all parameter errors, its error rates at set C are already less than 10%. By set E two of the tests have tallied a perfect detectability, which omits the need for any further tests down the table.

Table 6-33. Xdsup Test

Xdsup								Ave	Gain (%)	Contribution (%)
All	**0.3225**							0.3225	0.0000	0.0000
	Xd	Xq	Tds	Tdss	Tqss	Xdss	Xqss			
Set A	0.3350	0.2650	0.4425	0.3525	0.3525	0.2775	0.3500	0.3393	-5.2049	-5.2049
	Xd	Tds	Tdss	Tqss	Xdss	Xqss				
Set B	0.2200	0.3400	0.2325	0.1575	0.0950	0.1675		0.2021	37.3385	42.5434
	Xd	Tds	Tdss	Tqss	Xqss					
Set C	0.0900	0.2150	0.0575	0.0700	0.0850			0.1035	67.9070	30.5685
	Xd	Tds	Tqss	Xqss						
Set D	0.0625	0.1875	0.0525	0.0625				0.0913	71.7054	3.7984
	Xd	Tds	Xqss							
Set E	0.0275	0.0625	0.0125					0.0342	89.4057	17.7003
	Xd	Tds								
Set F	0.0025	0.0225						0.0125	96.1240	6.7183

The parameter class **Xdsup** starts with a high out-of-bag error rate, but at set B, one of the tests, **Xdss**, pulled down the error rate at 9.5%, which perhaps indicates its significant role in hindering **Xdsup**'s detectability. Its removal resulted to set C having mostly less than 10% error rates, with the exception of **Tds**, which interestingly continued down the set of tests as having the least contribution to improvement in accuracy.

Table 6-34. Xdsdn Test

Xdsdn								Ave	Gain (%)	Contribution (%)
All	**0.3575**							0.3575	0.0000	0.0000
	Xd	Xq	Tds	Tdss	Tqss	Xdss	Xqss			
Set A	0.3150	0.2250	0.3225	0.2850	0.2725	0.2600	0.2750	0.2793	21.8781	21.8781
	Xd	Tds	Tdss	Tqss	Xdss	Xqss				
Set B	0.1850	0.2000	0.1975	0.1200	0.0600	0.1600		0.1538	56.9930	35.1149
	Xd	Tds	Tdss	Tqss	Xqss					
Set C	0.0375	0.1475	0.0900	0.0575	0.0550			0.0775	78.3217	21.3287
	Tds	Tdss	Tqss	Xqss						
Set D	0.0325	0.0625	0.0200	0.0275				0.0356	90.0350	11.7133
	Tds	Tdss	Xqss							
Set E	0.0075	0.0150	0.0125					0.0117	96.7366	6.7016
	Tdss	Xqss								
Set F	0.0100	0.0025						0.0063	98.2517	1.5152

An interesting feature of parameter class **Xdsdn** is that, similar to **Xdsup**, by set B one of the tests already registered an error rate less than 10%, and it is caused by the removal of the same parameter **Xdss**. It is however important to note that classes **Xdsup** and **Xdsdn** are all derived from parameter **Xds**, which could explain the matching behavior of the two classes at the first two set of tests.

Table 6-35. Tdssdn Test

Tdssdn								Ave	Gain (%)	Contribution (%)
All	**0.3150**							0.3150	0.0000	0.0000
	Xd	Xq	Tds	Xds	Tqss	Xdss	Xqss			
Set A	0.3225	0.2325	0.3725	0.2050	0.1575	0.3750	0.3250	0.2843	9.7506	9.7506
	Xd	Xq	Tds	Xds	Xdss	Xqss				
Set B	0.1875	0.0975	0.2725	0.1200	0.3050	0.1775		0.1933	38.6243	28.8738
	Xd	Tds	Xds	Xdss	Xqss					
Set C	0.0825	0.1900	0.0425	0.1950	0.1550			0.1330	57.7778	19.1534
	Xd	Tds	Xdss	Xqss						
Set D	0.0425	0.0825	0.0800	0.0450				0.0625	80.1587	22.3810
	Tds	Xdss	Xqss							
Set E	0.0225	0.0300	0.0350					0.0292	90.7407	10.5820
	Xdss	Xqss								
Set F	0.0050	0.0100						0.0075	97.6190	6.8783

The parameter class, like **Xdsup** and **Xdsdn**, starts with a high out-of-bag error rate but by set B contains a test with a less than 10% error. As with most of the other parameter tests, the error rates do not go under 10% until it reaches set D.

Table 6-36. TqssdnTest

Tqssdn								Ave	Gain (%)	Contribution (%)
All	**0.1725**							0.1725	0.0000	0.0000
	Xd	Xq	Tds	Xds	Tdss	Xdss	Xqss			
Set A	0.1500	0.0725	0.2700	0.1375	0.1925	0.1725	0.1550	0.1643	4.7619	4.7619
	Xd	Tds	Xds	Tdss	Xdss	Xqss				
Set B	0.0400	0.1975	0.0750	0.1025	0.1100	0.0975		0.1038	39.8551	35.0932
	Tds	Xds	Tdss	Xdss	Xqss					
Set C	0.0975	0.0575	0.0725	0.0825	0.0825			0.0785	54.4928	14.6377
	Tds	Tdss	Xdss	Xqss						
Set D	0.0350	0.0175	0.0450	0.0425				0.0350	79.7101	25.2174
	Tds	Xdss	Xqss							
Set E	0.0125	0.0300	0.0175					0.0200	88.4058	8.6957
	Xdss	Xqss								
Set F	0.0100	0.0000						0.0050	97.1014	8.6957

The class **Tqssdn** is another parameter performing well with simultaneous active errors. All of its error rates are less than 10% by set C and less than 5% by set D.

Table 6-37. Xdssdn Test

Xdssdn								Ave	Gain (%)	Contribution (%)
All	**0.2875**							0.2875	0.0000	0.0000
	Xd	Xq	Tds	Xds	Tdss	Tqss	Xqss			
Set A	0.2975	0.1550	0.3150	0.1675	0.2725	0.2550	0.1375	0.2286	20.4969	20.4969
	Xd	Xq	Tds	Xds	Tdss	Tqss				
Set B	0.1550	0.0825	0.1400	0.1025	0.1400	0.1600		0.1300	54.7826	34.2857
	Xd	Tds	Xds	Tdss	Tqss					
Set C	0.0700	0.1000	0.0005	0.0550	0.0600			0.0571	80.1391	25.3565
	Xd	Tds	Tdss	Tqss						
Set D	0.0100	0.0450	0.0200	0.0125				0.0219	92.3913	12.2522
	Tds	Tdss	Tqss							
Set E	0.0125	0.0150	0.0250					0.0175	93.9130	1.5217
	Tdss	Tqss								
Set F	0.0075	0.0050						0.0063	97.8261	3.9130

The class parameter **Xdssdn** starts with a moderately high out-of-bag error rate, but was able to achieve a 10% or better error rate by set C and less than 5% by set D. It can be deduced that the removal of the respective parameters on each set effectively improved the accuracy of detection early in the tests.

Table 6-38. XqssdnTest

Xqssdn								Ave	Gain (%)	Contribution (%)
All	**0.2825**							0.2825	0.0000	0.0000
	Xd	Xq	Tds	Xds	Tdss	Tqss	Xdss			
Set A	0.2400	0.2550	0.3200	0.2600	0.2775	0.1475	0.2625	0.2518	10.8723	10.8723
	Xd	Xq	Tds	Xds	Tdss	Xdss				
Set B	0.1725	0.1075	0.1975	0.1325	0.1450	0.1625		0.1529	45.8702	34.9979
	Xd	Tds	Xds	Tdss	Xdss					
Set C	0.0700	0.1525	0.1025	0.1150	0.0500			0.0980	65.3097	19.4395
	Xd	Tds	Xds	Tdss						
Set D	0.0300	0.0775	0.0250	0.0525				0.0463	83.6283	18.3186
	Xd	Tds	Tdss							
Set E	0.0050	0.0500	0.0100					0.0217	92.3304	8.7021
	Tds	Tdss								
Set F	0.0025	0.0000						0.0013	99.5575	7.2271

The class parameter **Xqssdn** has a moderately high out-of-bag error rate in the presence of all active errors. As is typical with the other parameters, it reaches an error rate of less than 10% at set D.

6.3.3 Conclusion

The multi-parameter error test has shown that a parameter class' error, typically, is hardly detectable with simultaneous multiple errors present. It usually takes at least three error removals

out of seven given before a good error rate is achieved. There are however, a few classes like **Xqdn** and **Tqssdn** that have good default out-of-bag error rates, which will allow a good starting point for the process of finding model errors. Moreover, there are classes like **Xddn** that, upon removal of a parameter error, improves significantly. This analysis of parameters can reveal a wealth of details that can be useful for developing a method for systematically tweaking network component models. From what can be gathered in the experiment, the process of validation for models with an undetermined number of errors can proceed as follows:

1. Select component model(s) suspected of errors.
2. Identify candidate parameters that will likely require correction.
3. Run multi-parameter error simulations as described in the study.
4. Rank the parameter classes according to computed default class error rates.
5. Determine if there are highly detectable classes from the pool of parameters. If such parameters are present, assign these as starting classifiers. Otherwise, continue with multi-parameter simulations, removing one error at a time, and rate each removal according to its effect on accuracy.
6. Based on the results of the detailed simulations, validate the models starting with the most detectable parameter classes, followed by classes which are most affected in terms of accuracy by the preceding corrected class.

It is very important to understand that this form of model validation may be specific to the event and system under test, where the computed parameter error rankings and ratings may not apply to a different network subjected to a different disturbance. It is also essential to this procedure that the parameters are well-selected, as it can be argued that the probability of errors being simultaneously present in the real system is very low. One of the advantages of this type of validation is that the whole process can be automated, where the amount of data and the speed of computing necessary is quickly becoming trivial. However, an appropriate method for precisely applying the required correction for each ranked parameter has yet to be determined.

The process of correction upon identification of an error can possibly be accomplished using only the classifiers, but this involves running the simulations as incremental changes are made, rebuilding RF classifiers, and checking if the specific error still can be detected, all in a

recursive fashion, for each detected error. The problem to this approach is that it is inefficient, although again, the constantly improving computing power can easily accelerate the whole procedure, eliminating the concern. A related issue is that, without an idea of the weight of each parameter adjustment, convergence towards the correct model setting will unnecessarily take more time to complete. The preferred solution is an immediate tuning of the identified parameter error based on the dynamic response. The difficulty to this method is that a ‘perfect fit’ cannot be accomplished while other unidentified errors are present. Once this issue is addressed, a completely automated model validation program can be realized, where the only required input are disturbance measurements, which are provided as they occur.

7 Conclusion

The study has shown that a machine learning-based method for validating models through dynamic simulation can successfully make accurate associations between specific model parameter errors, and the disturbance response pattern. The discrete error classes, consisting of error ranges defined in the methodology, was demonstrated to have distinguishable footprints in the dynamics. The discretization of the model errors simplifies the process of error location in power system simulations, as it can give a clear direction of the specific corrections needed to get a matching outcome between the simulation and recorded information.

The single parameter error test comparing sequence components has conclusively determined that the negative sequence-based classifiers yield the highest accuracy rates, among other sequence data, for exciter, governor and standard generator parameters. The unbalanced short-circuit disturbances imposed on generators, which are pure positive sequence devices, are inferred to account for the negative sequence information's excellent classification accuracy. This is an important finding, considering that a PMU is mostly used as a positive sequence device. Most commercial PMUs have the capability of providing the single phase information from which the negative sequence data can be obtained, but this information is rarely used or collected.

Out of the three generator component models, the standard parameter errors consistently resulted to very high classification accuracy rates, followed by the exciter, then the governor. The governor has a few parameters identified to rate poorly. It can be speculated that the disturbance induced for the experiment cannot elicit a strong response from the governor model. This validates the fact that a disturbance-based model correction can only highlight a specific set of variables sensitive to the event. However, this can also potentially indicate the participation, and the sensitivity of the array of parameters describing a model, to an event. This is a relevant observation for disturbance analysis and parameter validation, as it indirectly suggests which set of parameters are actively contributing to the event response, thereby supporting a potential reduction in the list of suspect errors. The single parameter error validation, however, will only be valid for tracking single errors. An application of this method of error correction will be limited to models that are historically verified to have isolated parameters that constantly require tuning, or perhaps on

models that have mutually exclusive error characteristics. Single parameter error validation can also be used as an alarm for models that have parameters with varying rates of change. Early detection of fast evolving model settings will avoid simultaneous errors occurring.

An effort was made to test machine learning for identifying parameter errors simultaneously present with other random errors. It has been verified, in general, that target parameter errors in the wild with undetermined model characteristics only has a fair chance of being identified. It must be pointed out nonetheless, that the probability of multiple errors existing in a model, such as those in a generator, is reasonably low. The only likely exception to this are the load models used in power systems, where sweeping assumptions and approximations are made due to the complex and unpredictable nature of energy consumers. The experiment has revealed that for the standard parameter set, on average, an error has good detectability in the presence of three other random errors. It is also important to note that there are exceptional parameter classes that have very good classification accuracy with the default, or maximum number of errors. This can be seen as an opening for initiating corrections on models with multiple errors, where high-accuracy parameter classes can be tuned first, followed by either the next class in accuracy ranking, or the class which is known to be largely obscured or affected by the previous error. However, after an erroneous parameter is identified, an effective method for applying the correct tuning has yet to be identified.

Overall, this method of validation has revealed to be a promising alternative to existing techniques. The biggest advantage of the machine learning method is its adaptability. It can easily be implemented to validate all kinds of models, without exactly knowing the intricacies of the modeling equations involved in the dynamics. Moreover, the evaluation of models does not have to be all-inclusive. The parameters suspected of errors can merely be selected for the testing process, which reduces the burden of computation and the probability of incurring unwanted errors. On the other hand, it is imperative that all the models loaded into the simulation application are of sufficient detail. Otherwise, the simulation inaccuracies can be misclassified, and possibly lead to more errors. It is also perhaps ideal that for large power networks, partitioning is carried out, such as what was described in Section 2.2.1, to facilitate effective model error identification.

Another advantage is that this form of model evaluation allows for a parallel computing approach, which can speed up the process significantly. For example, the simulation tasks required for generating the parameter error classes can conveniently be partitioned into multiple machines, after which all the results can all be simply pooled into one database. Furthermore, as mentioned in [36], the Random Forest algorithm, likewise, can be executed in parallel.

A disadvantage for validation by classification is that the precise setting required is not directly determined. For single error model correction it is just a matter of tuning until a satisfactory match is achieved, according to the classifier's suggested direction. For multiple error tuning however, the amount of adjustment necessary at each step may demand a very close inspection of the variations in the dynamic response. Nevertheless, an iterative approach to model correction can be applied to simulations with manageable number of errors. This process can be automated, and run until an optimal match is realized.

7.1 Contributions

My main contribution, as detailed in this study, is the development of a method for model validation through classification, using disturbance measurements from phasor measurement units. Specifically, the two approaches tested were:

1. Single-error classification, which was designed to track lone parameter errors for tuning simulation models. The classifier can encompass all possible parameter errors from the model under test, through an identifiable power system event that can be recreated through simulation.
2. Multiple-error classification, where classifiers are generated for the verification of the existence, or absence of individual target parameter errors, in the presence of an undetermined number of other random errors. Like the first type, the classifiers are generated using a power system event that can be recreated through simulation.

All preceding methods were designed to break down possible errors into discrete classes. Consequently, it will not give precise corrections, but a direction for improvement. In its current form, the techniques can be used as outstanding aids for troubleshooting power system simulation models. Moreover, as a result of initial comparisons, another contribution from the study is the

determination of the importance of PMU negative sequence data for validating generator parameters against unbalanced events.

7.2 Future Research

One of the biggest potential improvements for the methodology described in this dissertation is the development or determination of a systematic method for sizing the margins used for defining the error classes. Each parameter in a model has a different weight on the event dynamic response, and thus will require careful application. What comprises a relevant change from the base or stock simulation model may be deterministic, or subjective, depending on the requirements of the power utility personnel performing the analysis. Also, it is important to accurately determine the expected working range of each parameter. Underestimating or overshooting the practical limits will be critical to the proper identification of errors.

Another possible research is an investigation into optimal methods for automatically correcting errors for single and multiple-error parameter validation. While the former can be solved using various curve fitting techniques, the latter will need more involved methods. Ideally, the corrections applied at each step for a multiple-error setting should be final, in which case the errors are effectively corrected, reducing the probability of misclassification for the next target parameter error. This will be the key to a fully automated validation method for simulation models using classification.

To reduce computation time, it is also very important to weigh the relevance of each predictor entry in the dataset prior to training. Whether the variables used in generating the classifier vary in significance from one event to another is a problem that should be explored. The measurement of variable importance can also improve detection of parameter errors in a multi-parameter error setting, as it can specifically track predictors that are strongly associated with a particular parameter class. This information can be used to quickly verify the validity of an error classification. Also, an analysis of the predictor variables using variable importance will help uncover the specific contributions of the parameter variations to the changes in the different sequence data. This information can produce a definitive guide to the selection of sequence information for validating parameters for a given disturbance. Random Forest itself can be used to

test variable importance using dataset permutations, as discussed in Section 3.2.3, however, this feature is not readily available in the software used for this study.

Finally, the machine learning-based validation should be tested in conjunction with hybrid simulation (Section 2.2.1). For small systems such as what was used in this study, the simulations required for the training and subsequent validation is trivial. However, for actual systems consisting of buses in the order of thousands, the network should be partitioned. Aside from speeding up the dynamic simulations, working in small sections accordingly reduces the number of parameter errors factored in for training, facilitating a more efficient model correction process.

8 References

[1] D. N. Kosterev, C. W. Taylor, and W. A. Mittelstadt, "Model validation for the August 10, 1996 WSCC system outage," *Power Systems, IEEE Transactions on*, vol. 14, pp. 967-979, 1999.

[2] EPRI, "System Disturbance Stability Studies for Western Systems Coordinating Council (WSCC)," EPRI April 1997.

[3] EPRI, "Power Plant Modeling and Parameter Derivation for Power System Studies," Power Delivery & Utilization June 19 2007.

[4] PTI, "Program Application Guide," vol. I, I. Power Technologies, Ed., PSS/E 29 Simulation Software Manual ed, 2002.

[5] H. Zhenyu, R. T. Guttromson, and J. F. Hauer, "Large-scale hybrid dynamic simulation employing field measurements," in *Power Engineering Society General Meeting, 2004. IEEE*, 2004, pp. 1570-1576 Vol.2.

[6] A. G. Phadke, "ARPA-E Project Proposal on Model Validation," Virginia Tech 2012.

[7] J. Sykes, Koellner, K., Premerlani W., Kasztenny, B., Adamiak, M., "Synchrophasors: A Primer and Practical Applications," presented at the Clemson Power Systems Conference, Clemson University, 2007.

[8] A. G. Phadke, *Synchronized phasor measurements and their applications*, 1st ed. New York: Springer, 2008.

[9] WSCC, "Test Guidelines for Synchronous Unit Dynamic Testing and Model Validation," Control Work Group and Modeling & Validation Work GroupFebruary 1997.

[10] H. Zhenyu, T. Nguyen, D. N. Kosterev, and R. Guttromson, "Model Validation of Power System Components Using Hybrid Dynamic Simulation," in *Transmission and Distribution Conference and Exhibition, 2005/2006 IEEE PES*, 2006, pp. 153-160.

[11] I. A. Hiskens and A. Koeman, "Parameter estimation from power system disturbance measurements," in *Energy Management and Power Delivery, 1998. Proceedings of EMPD '98. 1998 International Conference on*, 1998, pp. 667-672 vol.2.

[12] I. A. Hiskens and M. A. Pai, "Trajectory sensitivity analysis of hybrid systems," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on,* vol. 47, pp. 204-220, 2000.

[13] P. Kundur, N. J. Balu, and M. G. Lauby, *Power system stability and control*. New York: McGraw-Hill, 1994.

[14] F. P. De Mello and J. R. Ribeiro, "Derivation of synchronous machine parameters from tests," *Power Apparatus and Systems, IEEE Transactions on,* vol. 96, pp. 1211-1218, 1977.

[15] E. da Costa Bortoni and J. A. Jardini, "Identification of synchronous machine parameters using load rejection test data," *Energy Conversion, IEEE Transactions on,* vol. 17, pp. 242-247, 2002.

[16] C. C. Lee and O. T. Tan, "A weighted-least-squares parameter estimator for synchronous machines," *Power Apparatus and Systems, IEEE Transactions on,* vol. 96, pp. 97-101, 1977.

[17] J. H. Chow, M. T. Glinkowski, R. J. Murphy, T. W. Cease, and N. Kosaka, "Generator and exciter parameter estimation of Fort Patrick Henry Hydro Unit 1," *Energy Conversion, IEEE Transactions on,* vol. 14, pp. 923-929, 1999.

[18] D. N. Kosterev, "Hydro turbine-governor model validation in pacific northwest," *Power Systems, IEEE Transactions on,* vol. 19, pp. 1144-1149, 2004.
[19] H. Zhenyu, D. N. Kosterev, R. Guttromson, and T. Nguyen, "Model validation with hybrid dynamic simulation," in *Power Engineering Society General Meeting, 2006. IEEE,* 2006, p. 9 pp.
[20] H. Zhenyu, D. Pengwei, D. Kosterev, and S. Yang, "Generator dynamic model validation and parameter calibration using phasor measurements at the point of connection," *Power Systems, IEEE Transactions on,* vol. 28, pp. 1939-1949, 2013.
[21] S. M. Benchluch and J. H. Chow, "A trajectory sensitivity method for the identification of nonlinear excitation system models," *Energy Conversion, IEEE Transactions on,* vol. 8, pp. 159-164, 1993.
[22] I. A. Hiskens and J. Alseddiqui, "Sensitivity, Approximation, and Uncertainty in Power System Dynamic Simulation," *Power Systems, IEEE Transactions on,* vol. 21, pp. 1808-1820, 2006.
[23] E. P. T. Cari, L. F. C. Alberto, and N. G. Bretas, "A new methodology for parameter estimation of synchronous generator from disturbance measurements," in *Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*, 2008, pp. 1-7.
[24] T. Chin-Chu, L. Wei-Jen, E. Nashawati, W. Chin-Chung, and L. Hong-Wei, "PMU based generator parameter identification to improve the system planning and operation," in *Power and Energy Society General Meeting, 2012 IEEE*, 2012, pp. 1-8.
[25] J. Ma, D. Han, W. J. Sheng, R. M. He, C. Y. Yue, and J. Zhang, "Wide area measurements-based model validation and its application," *Generation, Transmission & Distribution, IET*, vol. 2, pp. 906-916, 2008.
[26] B. Hua, Z. Pei, and V. Ajjarapu, "A Novel Parameter Identification Approach via Hybrid Learning for Aggregate Load Modeling," *Power Systems, IEEE Transactions on,* vol. 24, pp. 1145-1154, 2009.
[27] C. Yunzhi, L. Wei-Jen, H. Shun-Hsien, and J. Adams, "A hybrid method for the dynamic parameter identification of generators via on-line measurements," in *Industrial and Commercial Power Systems Technical Conference (I&CPS), 2010 IEEE*, 2010, pp. 1-7.
[28] W. Jin-Cheng, C. Hsiao-Dong, C.-T. Huang, C. Yung-Tung, C. Chung-Liang, and H. Chiung-Yi, "Identification of excitation system models based on on-line digital measurements," *Power Systems, IEEE Transactions on,* vol. 10, pp. 1286-1293, 1995.
[29] P. Pourbeik, "Automated parameter derivation for power plant models based on staged tests," in *Power Systems Conference and Exposition, 2009. PSCE '09. IEEE/PES*, 2009, pp. 1-9.
[30] P. Pourbeik, R. Rhinier, H. Shih-Min, B. Agrawal, and R. Bisbee, "Semiautomated Model Validation of Power Plant Equipment Using Online Measurements," *Energy Conversion, IEEE Transactions on,* vol. 28, pp. 308-316, 2013.
[31] X. Longya, Z. Zhengming, and J. Jianguo, "On-line estimation of variable parameters of synchronous machines using a novel adaptive algorithm. Estimation and experimental verification," *Energy Conversion, IEEE Transactions on,* vol. 12, pp. 200-210, 1997.
[32] Z. Zhengming, X. Longya, and J. Jianguo, "On-line estimation of variable parameters of synchronous machines using a novel adaptive algorithm-principles and procedures," *Energy Conversion, IEEE Transactions on,* vol. 12, pp. 193-199, 1997.

[33] H. B. Karayaka, A. Keyhani, G. T. Heydt, B. L. Agrawal, and D. A. Selin, "Synchronous generator model identification and parameter estimation from operating data," *Energy Conversion, IEEE Transactions on*, vol. 18, pp. 121-126, 2003.
[34] I. Kamwa, S. R. Samantaray, and G. Joos, "On the Accuracy Versus Transparency Trade-Off of Data-Mining Models for Fast-Response PMU-Based Catastrophe Predictors," *Smart Grid, IEEE Transactions on*, vol. 3, pp. 152-161, 2012.
[35] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5-32, 2001.
[36] C. Zhang and Y. Ma. (2012). *Ensemble machine learning methods and applications*. Available: http://dx.doi.org/10.1007/978-1-4419-9326-7
[37] I. Kamwa, S. R. Samantaray, and G. Joos, "Catastrophe Predictors From Ensemble Decision-Tree Learning of Wide-Area Severity Indices," *Smart Grid, IEEE Transactions on*, vol. 1, pp. 144-158, 2010.
[38] S. R. Samantaray, I. Kamwa, and G. Joos, "Ensemble decision trees for phasor measurement unit-based wide-area security assessment in the operations time frame," *Generation, Transmission & Distribution, IET*, vol. 4, pp. 1334-1348, 2010.
[39] A. Kusiak and A. Verma, "A Data-Mining Approach to Monitoring Wind Turbines," *Sustainable Energy, IEEE Transactions on*, vol. 3, pp. 150-157, 2012.
[40] S. R. Samantaray, "A Data-Mining Model for Protection of FACTS-Based Transmission Line," *Power Delivery, IEEE Transactions on*, vol. 28, pp. 612-618, 2013.
[41] M. Bramer, *Principles of data mining*, 2nd ed. New York: Springer, 2013.
[42] L. Breiman and A. Cutler. (2014). *Random Forests*. Available: http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
[43] (2014). *Weka*. Available: http://www.cs.waikato.ac.nz/ml/weka/
[44] J. F. Hauer, C. J. Demeure, and L. L. Scharf, "Initial results in Prony analysis of power system response signals," *Power Systems, IEEE Transactions on*, vol. 5, pp. 80-89, 1990.
[45] J. F. Hauer, "Application of Prony analysis to the determination of modal content and equivalent models for measured power system response," *Power Systems, IEEE Transactions on*, vol. 6, pp. 1062-1068, 1991.
[46] D. J. Trudnowski and J. E. Dagle, "Effects of generator and static-load nonlinearities on electromechanical oscillations," *Power Systems, IEEE Transactions on*, vol. 12, pp. 1283-1289, 1997.
[47] D. J. Trudnowski, J. M. Johnson, and J. F. Hauer, "Making Prony analysis more accurate using multiple signals," *Power Systems, IEEE Transactions on*, vol. 14, pp. 226-231, 1999.
[48] C. E. Grund, J. J. Paserba, J. F. Hauer, and S. L. Nilsson, "Comparison of Prony and eigenanalysis for power system control design," *Power Systems, IEEE Transactions on*, vol. 8, pp. 964-971, 1993.
[49] D. A. Pierre, D. J. Trudnowski, and J. F. Hauer, "Identifying linear reduced-order models for systems with arbitrary initial conditions using Prony signal analysis," *Automatic Control, IEEE Transactions on*, vol. 37, pp. 831-835, 1992.
[50] J. W. Pierre, D. J. Trudnowski, and M. K. Donnelly, "Initial results in electromechanical mode identification from ambient data," *Power Systems, IEEE Transactions on*, vol. 12, pp. 1245-1251, 1997.
[51] J. J. Sanchez-Gasca and J. H. Chow, "Performance comparison of three identification methods for the analysis of electromechanical oscillations," *Power Systems, IEEE Transactions on*, vol. 14, pp. 995-1002, 1999.

[52] J. J. Sanchez-Gasca, K. Clark, N. W. Miller, H. Okamoto, A. Kurita, and J. H. Chow, "Identifying linear models from time domain simulations," *Computer Applications in Power, IEEE,* vol. 10, pp. 26-30, 1997.
[53] D. J. Trudnowski, "Order reduction of large-scale linear oscillatory system models," *Power Systems, IEEE Transactions on,* vol. 9, pp. 451-458, 1994.
[54] O. Chaari, P. Bastard, and M. Meunier, "Prony's method: an efficient tool for the analysis of earth fault currents in Petersen-coil-protected networks," *Power Delivery, IEEE Transactions on,* vol. 10, pp. 1234-1241, 1995.
[55] J. Faiz, S. Lotfi-Fard, and S. H. Shahri, "Prony-Based Optimal Bayes Fault Classification of Overcurrent Protection," *Power Delivery, IEEE Transactions on,* vol. 22, pp. 1326-1334, 2007.
[56] I. C. Cheng and G. W. Chang, "An Efficient Prony-Based Solution Procedure for Tracking of Power System Voltage Variations," *Industrial Electronics, IEEE Transactions on,* vol. 60, pp. 2681-2688, 2013.
[57] J. A. de la O Serna, "Synchrophasor Estimation Using Prony's Method," *Instrumentation and Measurement, IEEE Transactions on,* vol. 62, pp. 2119-2128, 2013.
[58] J. C. H. Peng and N. K. C. Nair, "Adaptive sampling scheme for monitoring oscillations using Prony analysis," *Generation, Transmission & Distribution, IET,* vol. 3, pp. 1052-1060, 2009.
[59] S. L. Marple, *Digital spectral analysis : with appplications*. Englewood Cliffs, N.J.: Prentice-Hall, 1987.
[60] T. W. Parks and C. S. Burrus, *Digital filter design*. New York: Wiley, 1987.
[61] (2014). *DIgSILENT PowerFactory*. Available: http://www.digsilent.de/index.php/products-powerfactory.html
[62] "DIgSILENT PowerFactory 15 User Manual," 2013.
[63] J. L. Blackburn, *Protective relaying : principles and applications*. New York: M. Dekker, 1987.
[64] (2014). *Prony Method for Filter Design*. Available: http://www.mathworks.com/help/signal/ref/prony.html
[65] A. Swami. (2003). *HOSA - Higher Order Spectral Analysis Toolbox*. Available: http://www.mathworks.com/matlabcentral/fileexchange/3013-hosa-higher-order-spectral-analysis-toolbox
[66] "Weka Manual for Version 3-6-9," ed. The University of Waikato, 2013.
[67] P. Stober and S.-T. Yeh, "An Explicit Functional Form Specification Approach to Estimate the Area Under a Receiver Operating Characteristic Curve," ed, 2002.
[68] T. Tape, "The Area Under An ROC Curve," in *Interpreting Diagnostic Tests*, ed. University of Nebraska Medical Center, 2014.

Appendix A DIgSILENT Codes

A.1 Single Parameter Error Main Script

```
! MainEventGen.ComDpl
! Author: Noah Garcia Badayos, 2014

object lne;
object modl;
object modle;
object modlg;
object obex;
string svpath;

lne = line;
modl = testModelSG;
modle = testExciter;
modlg = testGov;
obex = exporter;

svpath = 'C:\\Users\\Noah\\Desktop\\SimData\\';

! Reference Case

output('Starting base dynamic simulation');
Base.Execute(lne,50,obex,InitCom,RunSim,svpath,numcases);

! Synchronous Machine Parameters

output('Starting parameter Xd+ variation');
Xdup.Execute(lne,50,modl,0.1,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Xd- variation');
Xddn.Execute(lne,50,modl,0.1,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Xq+ variation');
Xqup.Execute(lne,50,modl,0.1,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Xq- variation');
Xqdn.Execute(lne,50,modl,0.1,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Tds+ variation');
Tdsup.Execute(lne,50,modl,0.5,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Xds+ variation');
Xdsup.Execute(lne,50,modl,0.025,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Xds- variation');
Xdsdn.Execute(lne,50,modl,0.025,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Tdss- variation');
Tdssdn.Execute(lne,50,modl,0.01,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Tqss- variation');
Tqssdn.Execute(lne,50,modl,0.01,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Xdss- variation');
Xdssdn.Execute(lne,50,modl,0.025,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Xqss- variation');
Xqssdn.Execute(lne,50,modl,0.025,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Xl+ variation');
Xlup.Execute(lne,50,modl,0.025,obex,InitCom,RunSim,svpath,numcases);

! Exciter Parameters
```

```
output('Starting parameter Tr+ variation');
Trup.Execute(lne,50,modle,0.0025,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Tr- variation');
Trdn.Execute(lne,50,modle,0.0025,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Ka+ variation');
Kaup.Execute(lne,50,modle,5,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Ka- variation');
Kadn.Execute(lne,50,modle,5,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Ta+ variation');
Taup.Execute(lne,50,modle,0.0025,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Ta- variation');
Tadn.Execute(lne,50,modle,0.0025,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Ke+ variation');
Keup.Execute(lne,50,modle,0.1,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Ke- variation');
Kedn.Execute(lne,50,modle,0.1,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Te+ variation');
Teup.Execute(lne,50,modle,0.1,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Kf+ variation');
Kfup.Execute(lne,50,modle,0.0005,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Tf+ variation');
Tfup.Execute(lne,50,modle,0.1,obex,InitCom,RunSim,svpath,numcases);

! Governor Parameters

output('Starting parameter R+ variation');
Rup.Execute(lne,50,modlg,0.0025,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter R- variation');
Rdn.Execute(lne,50,modlg,0.0025,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter r+ variation');
Rtup.Execute(lne,50,modlg,0.0025,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter r- variation');
Rtdn.Execute(lne,50,modlg,0.0025,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter GTr+ variation');
GTrup.Execute(lne,50,modlg,0.5,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter GTr- variation');
GTrdn.Execute(lne,50,modlg,0.5,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter GTf+ variation');
GTfup.Execute(lne,50,modlg,0.0025,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter GTf- variation');
GTfdn.Execute(lne,50,modlg,0.0025,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Tw+ variation');
Twup.Execute(lne,50,modlg,0.025,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter At+ variation');
Atup.Execute(lne,50,modlg,0.025,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter At- variation');
Atdn.Execute(lne,50,modlg,0.025,obex,InitCom,RunSim,svpath,numcases);

output('Starting parameter Tg+ variation');
Tgup.Execute(lne,50,modlg,0.025,obex,InitCom,RunSim,svpath,numcases);
```

```
output('Starting parameter Tg- variation');
Tgdn.Execute(lne,50,modlg,0.025,obex,InitCom,RunSim,svpath,numcases);
```

A.2 Single Parameter Error Case Subroutine Samples

The following DPL scripts are sample subroutines for randomizing the dynamic parameters in a model for DIgSILENT. Each subroutine also includes the fault location randomization, and the initialization and execution of the simulation.

A.2.1 Xqup Parameter Randomizer

```
! Xqup Subroutine
! Author: Noah Garcia Badayos, 2014

int err1,err2,err3,i;
double rand;
double randPar;
double cei;
double rang;
double oVal;

EchoOff();

cei = 2.3;

oVal = modl:xq;
rang = cei - oVal - marg;

printf('xq = %f | positive ceiling = %f | margin = %f', modl:xq,cei,marg);

for (i = 0; i < iter; i = i+1) {

  rand = Random(-10,10);
  lne:fshcloc = floc + rand;

  randPar = Random(rang);
  modl:xq = oVal + marg + randPar;

  printf('%i fault location= %f%% ',i+1, lne:fshcloc);
  printf('  modified parameter xq(+) = %f', modl:xq);

  err1 = InitCom.Execute();
  if(err1) {
    output('Initialization returns an error');
    exit();
  }

  err2 = RunSim.Execute();
  if(err2) {
    output('Simulation returns an error');
    exit();
  }

  obex:f_name = sprintf('%s%s%i.txt',svpath,'Xqup',i);
  err3 = obex.Execute();
  if(err3) {
    output('Output function returns an error');
    exit();
  } else {
    printf('  %s%i.txt successfully written','Xqup',i);
  }
}

modl:xq = oVal;
printf('==modified parameter restored xq = %f', modl:xq);
```

```
EchoOn();
```

A.2.2 Xqdn Parameter

```
! Xqdn Subroutine
! Author: Noah Garcia Badayos, 2014

int err1,err2,err3,i;
double rand;
double randPar;
double cei;
double rang;
double oVal;

EchoOff();

cei = 0.4;

oVal = modl:xq;
rang = oVal - cei - marg;

printf('xq = %f | negative ceiling = %f | margin = %f', modl:xq,cei,marg);

for (i = 0; i < iter; i = i+1) {

  rand = Random(-10,10);
  lne:fshcloc = floc + rand;

  randPar = Random(rang);
  modl:xq = oVal - marg - randPar;

  printf('%i fault location= %f%% ',i+1, lne:fshcloc);
  printf('  modified parameter xq(-)= %f', modl:xq);

  err1 = InitCom.Execute();
  if(err1) {
    output('Initialization returns an error');
    exit();
  }

  err2 = RunSim.Execute();
  if(err2) {
    output('Simulation returns an error');
    exit();
  }

  obex:f_name = sprintf('%s%s%i.txt',svpath,'Xqdn',i);
  err3 = obex.Execute();
  if(err3) {
    output('Output function returns an error');
    exit();
  } else {
    printf('  %s%i.txt successfully written','Xqdn',i);
  }
}

modl:xq = oVal;
printf('==modified parameter restored xq = %f', modl:xq);

EchoOn();
```

A.3 Multiple Parameter Error Main Script

```
! MultiParameter.ComDpl
! Author: Noah Garcia Badayos, 2014

int err1,err2,err3,i,p;
double rand;
double ovalxq;
```

```
double ovalxd;
double ovaltds;
double ovalxds;
double ovaltdss;
double ovaltqss;
double ovalxdss;
double ovalxqss;
double ovalxl;
double rand1,rand2,rand3;

object lne;
object modl;
object modle;
object modlg;
object obex;
string svpath;

int mainpar;
string mainpfx;
string onpref;
string offpref;
string fname;
int mswitch;

lne = line;
modl = testModelSG;
modle = testExciter;
modlg = testGov;
obex = exporter;

svpath = folderpath;

!======================initialization=================================
! (looking for the main test parameter)

if (vXdup = 3) {
  mainpfx = 'Xdup';
}

if (vXddn = 3) {
  mainpfx = 'Xddn';
}

if (vXqup = 3) {
  mainpfx = 'Xqup';
}

if (vXqdn = 3) {
  mainpfx = 'Xqdn';
}

if (vTdsup = 3) {
  mainpfx = 'Tdsup';
}

if (vXdsup = 3) {
  mainpfx = 'Xdsup';
}

if (vXdsdn = 3) {
  mainpfx = 'Xdsdn';
}

if (vTdssdn = 3) {
  mainpfx = 'Tdssdn';
}

if (vTqssdn = 3) {
  mainpfx = 'Tqssdn';
}
```

```
if (vXdssdn = 3) {
  mainpfx = 'Xdssdn';
}

if (vXqssdn = 3) {
  mainpfx = 'Xqssdn';
}

onpref = sprintf('%sOn',mainpfx);
offpref = sprintf('%sOff',mainpfx);

!=====================================================================

for (p = 0; p < 2; p = p+1) {

if (p = 0) {
  fname = onpref;
  mswitch = 1;
} else {
  fname = offpref;
  mswitch = 0;
}

  for (i = 0; i < iter; i = i+1) {

    rand = Random(-10,10);
    lne:fshcloc = floc + rand;

    printf('%i fault location= %f%% ',i+1, lne:fshcloc);

    !=========================parameter mod==============================

    rand1 = Random();
    rand2 = Random();
    rand3 = Random();

    ovalxd = modl:xd;              !Xd
    if(vXdup = 1 .and. rand1 >= 0.5 .or. vXdup = 3 .and. mswitch = 1) {
      Xdup.Execute(modl,0.1,1);
    }
    if(vXddn = 1 .and. rand1 < 0.5 .or. vXddn = 3 .and. mswitch = 1) {
      Xddn.Execute(modl,0.1,1);
    }

    ovalxq = modl:xq;              !Xq
    if(vXqup = 1 .and. rand2 >= 0.5 .or. vXqup = 3 .and. mswitch = 1) {
      Xqup.Execute(modl,0.1,1);
    }
    if(vXqdn = 1 .and. rand2 < 0.5 .or. vXqdn = 3 .and. mswitch = 1) {
      Xqdn.Execute(modl,0.1,1);
    }

    ovaltds = modl:tds;            !Tds
    if(vTdsup = 1 .or. vTdsup = 3 .and. mswitch = 1) {
      Tdsup.Execute(modl,0.5,1);
    }

    ovalxds = modl:xds;            !Xds
    if(vXdsup = 1 .and. rand3 >= 0.5 .or. vXdsup = 3 .and. mswitch = 1) {
      Xdsup.Execute(modl,0.025,1);
    }
    if(vXdsdn = 1 .and. rand3 < 0.5.or. vXdsdn = 3 .and. mswitch = 1) {
      Xdsdn.Execute(modl,0.025,1);
    }

    ovaltdss = modl:tdss;          !Tdss
    if(vTdssdn = 1 .or. vTdssdn = 3 .and. mswitch = 1) {
      Tdssdn.Execute(modl,0.01,1);
    }

    ovaltqss = modl:tqss;          !Tqss
```

```
  if(vTqssdn = 1 .or. vTqssdn = 3 .and. mswitch = 1) {
    Tqssdn.Execute(modl,0.01,1);
  }

  ovalxdss = modl:xdss;         !Xdss
  if(vXdssdn = 1 .or. vXdssdn = 3 .and. mswitch = 1) {
    Xdssdn.Execute(modl,0.025,1);
  }

  ovalxqss = modl:xqss;         !Xqss
  if(vXqssdn = 1 .or. vXqssdn = 3 .and. mswitch = 1) {
    Xqssdn.Execute(modl,0.025,1);
  }
!   ovalxl = modl:xl;             !Xl
!   if(vXlup = 1 .or. vXlup = 3 .and. mswitch = 1) {
!     Xlup.Execute(modl,0.025,1);
!   }

  !=======================parameter mod ends=========================

   EchoOff();

   err1 = InitCom.Execute();
   if(err1) {
     output('Initialization returns an error');
     exit();
   }

   err2 = RunSim.Execute();
   if(err2) {
     output('Simulation returns an error');
     exit();
   }

   obex:f_name = sprintf('%s%s%i.txt',svpath,fname,i);
   err3 = obex.Execute();
   if(err3) {
     output('Output function returns an error');
     exit();
   } else {
     printf('  %s%i.txt successfully written',fname,i);
   }

   EchoOn();

  !==================parameter restore=========================

  modl:xd = ovalxd;
  printf('==modified parameter restored xd = %f',modl:xd);

  modl:xq = ovalxq;
  printf('==modified parameter restored xq = %f',modl:xq);

  modl:tds = ovaltds;
  printf('==modified parameter restored tds = %f',modl:tds);

  modl:xds = ovalxds;
  printf('==modified parameter restored xds = %f',modl:xds);

  modl:tdss = ovaltdss;
  printf('==modified parameter restored tdss = %f',modl:tdss);

  modl:tqss = ovaltqss;
  printf('==modified parameter restored tqss = %f',modl:tqss);

  modl:xdss = ovalxdss;
  printf('==modified parameter restored xdss = %f',modl:xdss);

  modl:xqss = ovalxqss;
  printf('==modified parameter restored xqss = %f',modl:xqss);
```

```
  !   modl:xl = ovalxl;
  !   printf('==modified parameter restored xl = %f',modl:xl);

    !=================parameter restore ends============================
  }
}
```

A.4 Multiple Parameter Error Subroutine Sample

The following script is a sample subroutine used for the multiple parameter error test.

```
! Xqup subroutine for multiple parameter test
! Author: Noah Garcia Badayos, 2014

double rand;
double randPar;
double cei;
double rang;
double oVal;


EchoOff();

cei = 2.3;

oVal = modl:xq;
rang = cei - oVal - marg;

printf('xq = %f | positive ceiling = %f | margin = %f', modl:xq,cei,marg);

rand = Random();
randPar = Random(rang);

if(rand >= 0.5 .or. randx = 1) {
  modl:xq = oVal + marg + randPar;
}

printf('  modified parameter xq(+) = %f', modl:xq);

EchoOn();
```

Appendix B Custom C# Application Codes

B.1 DIgSILENT Text Output Processor

B.1.1 MainForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Collections;

// The TextFileDataExtractor Application was specifically written for the model validation process
described in this study.
// The code can be used for educational purposes and may not be commercialized.
// Author: Noah Garcia Badayos, 2014

namespace TextFileDataExtractor
{
    public partial class MainForm : Form
    {
        private FolderBrowserDialog folderBrowserDialog1;
        private string folderName;

        StringBuilder dummyConsole = new StringBuilder();
        int reps;
        int dbID;
        int startInd;
        int pmax;
        string dbTableName;

        bool opswitch = true; //optimization control

        DatabaseAccess db;
        MLApp.MLApp matlab;

        public MainForm()
        {
            InitializeComponent();

            this.folderBrowserDialog1 = new System.Windows.Forms.FolderBrowserDialog();
            this.folderBrowserDialog1.Description = "Select DIgSILENT Output Folder";

            db = new DatabaseAccess();
            matlab = new MLApp.MLApp();

            for (int i = 0; i <  chklbox1.Items.Count; i++)
            {
                chklbox1.SetItemChecked(i, true);
            }

        }

        private void btnSelect_Click(object sender, EventArgs e)
        {
            DialogResult result = folderBrowserDialog1.ShowDialog();
            if (result == DialogResult.OK)
            {
                folderName = folderBrowserDialog1.SelectedPath + "\\";
```

```
                dummyConsole.AppendLine(string.Format("The path {0} has been selected... ",
folderName.ToString()));
                txtTest.Text = dummyConsole.ToString();
                btnRead.Enabled = true;
            }
        }

        private void btnRead_Click(object sender, EventArgs e)
        {
            btnRead.Enabled = false;
            btnSelect.Enabled = false;
            groupBox1.Enabled = false;
            chklbox1.Enabled = false;

            try
            {
                reps = Convert.ToInt16(txtRep.Text);
                dbID = Convert.ToInt16(txtDBID.Text);
                startInd = Convert.ToInt16(txtStart.Text);
                pmax = Convert.ToInt16(txtPmax.Text);
                dbTableName = Convert.ToString(txtDBName.Text);

                db.setStartID(dbID);
                matlab.Execute("clear");

                foreach (object itemChecked in chklbox1.CheckedItems)
                {
                    dummyConsole.AppendLine("processing...");
                    txtTest.Text = dummyConsole.ToString();

                    for (int num = 0; num < reps; num++)
                    {
                        string filename;
                        string paramtxt = itemChecked as string;

                        filename = string.Format(@"{0}{1}.txt", paramtxt, num);
                        if (File.Exists(folderName + filename))
                        {
                            db.incrementID(dbTableName);
                            db.addScenario(paramtxt, dbTableName);
                            parseData(filename, num);

                        }
                        else
                        {
                            dummyConsole.AppendLine(string.Format("The file {0} is not found in
the directory", filename));
                            txtTest.Text = dummyConsole.ToString();
                        }
                    }
                }

                dummyConsole.AppendLine("SQL transfer completed!");
                txtTest.Text = dummyConsole.ToString();
                btnRead.Enabled = true;
                btnSelect.Enabled = true;
                groupBox1.Enabled = true;
                chklbox1.Enabled = true;
            }
            catch (Exception err)
            {
                dummyConsole.AppendLine("The file could not be read:");
                dummyConsole.AppendLine(err.Message);
                txtTest.Text = dummyConsole.ToString();
            }
        }

        private void parseData(string filename, int iterNum)
        {
            ArrayList tlist = new ArrayList();
            ArrayList cllist = new ArrayList();
```

```
ArrayList c2list = new ArrayList();
ArrayList c3list = new ArrayList();
ArrayList c4list = new ArrayList();
ArrayList c5list = new ArrayList();
ArrayList c6list = new ArrayList();
ArrayList c7list = new ArrayList();
ArrayList c8list = new ArrayList();
ArrayList c9list = new ArrayList();
ArrayList c10list = new ArrayList();
ArrayList c11list = new ArrayList();
ArrayList c12list = new ArrayList();
ArrayList c13list = new ArrayList();
ArrayList c14list = new ArrayList();

int counter = 0;
int iter = 0; // control order optimization

StreamReader sr = new StreamReader(folderName + filename);

while (sr.Peek() != -1)
{
    counter++;
    if (counter <= 2)
    {
        sr.ReadLine();
        continue;
    }

    string[] srcString = sr.ReadLine().Split('  ');

    //creating time array
    tlist.Add(Convert.ToDouble(srcString[0]));

    if (c1.Checked)
    {
        c1list.Add(Convert.ToDouble(srcString[1]));
    }
    if (c2.Checked)
    {
        c2list.Add(Convert.ToDouble(srcString[2]));
    }
    if (c3.Checked)
    {
        c3list.Add(Convert.ToDouble(srcString[3]));
    }
    if (c4.Checked)
    {
        c4list.Add(Convert.ToDouble(srcString[4]));
    }
    if (c5.Checked)
    {
        c5list.Add(Convert.ToDouble(srcString[5]));
    }
    if (c6.Checked)
    {
        c6list.Add(Convert.ToDouble(srcString[6]));
    }
    if (c7.Checked)
    {
        c7list.Add(Convert.ToDouble(srcString[7]));
    }
    if (c8.Checked)
    {
        c8list.Add(Convert.ToDouble(srcString[8]));
    }
    if (c9.Checked)
    {
        c9list.Add(Convert.ToDouble(srcString[9]));
    }
    if (c10.Checked)
    {
```

```
            c10list.Add(Convert.ToDouble(srcString[10]));
        }
        if (c11.Checked)
        {
            c11list.Add(Convert.ToDouble(srcString[11]));
        }
        if (c12.Checked)
        {
            c12list.Add(Convert.ToDouble(srcString[12]));
        }
        if (c13.Checked)
        {
            c13list.Add(Convert.ToDouble(srcString[13]));
        }
        if (c14.Checked)
        {
            c14list.Add(Convert.ToDouble(srcString[14]));
        }
    }


    if (opswitch) //optimization control
    {
        iter = 0;
    }
    else
    {
        iter = 1;
    }

    matlab.PutWorkspaceData("start","base",startInd);
    matlab.PutWorkspaceData("pmax", "base", pmax);
    matlab.PutWorkspaceData("tlist", "base", tlist.Cast<double>().ToArray());
    matlab.Execute(@"tlist=trimwave(tlist,start);");

    if (c1.Checked)
    {
        matlab.PutWorkspaceData("vmag0", "base", c1list.Cast<double>().ToArray());
        matlab.Execute(@"vmag0=trimwave(vmag0,start);");
        if (iter == 0)
        {
            matlab.Execute(@"p1=optimalpoles(vmag0,pmax);");
        }
        matlab.Execute(@"[ma,ar]=pronysim(vmag0,p1,pmax);");

        if (chkProny.Checked)
        {
            matlab.Execute(@"[vec]=pronycomp(ma,ar,pmax);");
            db.addVoltageMag0(getSimArrayProny(), dbTableName);
        }
        else
        {
            db.addVoltageMag0(getSimArray(), dbTableName);
        }
    }
    if (c2.Checked)
    {
        matlab.PutWorkspaceData("vmag1", "base", c2list.Cast<double>().ToArray());
        matlab.Execute(@"vmag1=trimwave(vmag1,start);");
        if (iter == 0)
        {
            matlab.Execute(@"p2=optimalpoles(vmag1,pmax);");
        }
        matlab.Execute(@"[ma,ar]=pronysim(vmag1,p2,pmax);");

        if (chkProny.Checked)
        {
            matlab.Execute(@"[vec]=pronycomp(ma,ar,pmax);");
            db.addVoltageMag1(getSimArrayProny(), dbTableName);
        }
        else
```

```
        {
            db.addVoltageMag1(getSimArray(), dbTableName);
        }
    }
    if (c3.Checked)
    {
        matlab.PutWorkspaceData("vmag2", "base", c3list.Cast<double>().ToArray());
        matlab.Execute(@"vmag2=trimwave(vmag2,start);");
        if (iter == 0)
        {
            matlab.Execute(@"p3=optimalpoles(vmag2,pmax);");
        }
        matlab.Execute(@"[ma,ar]=pronysim(vmag2,p3,pmax);");

        if (chkProny.Checked)
        {
            matlab.Execute(@"[vec]=pronycomp(ma,ar,pmax);");
            db.addVoltageMag2(getSimArrayProny(), dbTableName);
        }
        else
        {
            db.addVoltageMag2(getSimArray(), dbTableName);
        }
    }
    if (c4.Checked)
    {
        matlab.PutWorkspaceData("vang0", "base", c4list.Cast<double>().ToArray());
        matlab.Execute(@"vang0=trimwave(vang0,start);");
        matlab.Execute(@"vang0=angleadjust(vang0);");
        if (iter == 0)
        {
            matlab.Execute(@"p4=optimalpoles(vang0,pmax);");
        }
        matlab.Execute(@"[ma,ar]=pronysim(vang0,p4,pmax);");

        if (chkProny.Checked)
        {
            matlab.Execute(@"[vec]=pronycomp(ma,ar,pmax);");
            db.addVoltageAngle0(getSimArrayProny(), dbTableName);
        }
        else
        {
            db.addVoltageAngle0(getSimArray(), dbTableName);
        }
    }
    if (c5.Checked)
    {
        matlab.PutWorkspaceData("vang1", "base", c5list.Cast<double>().ToArray());
        matlab.Execute(@"vang1=trimwave(vang1,start);");
        matlab.Execute(@"vang1=angleadjust(vang1);");
        if (iter == 0)
        {
            matlab.Execute(@"p5=optimalpoles(vang1,pmax);");
        }
        matlab.Execute(@"[ma,ar]=pronysim(vang1,p5,pmax);");

        if (chkProny.Checked)
        {
            matlab.Execute(@"[vec]=pronycomp(ma,ar,pmax);");
            db.addVoltageAngle1(getSimArrayProny(), dbTableName);
        }
        else
        {
            db.addVoltageAngle1(getSimArray(), dbTableName);
        }
    }
    if (c6.Checked)
    {
        matlab.PutWorkspaceData("vang2", "base", c6list.Cast<double>().ToArray());
        matlab.Execute(@"vang2=trimwave(vang2,start);");
        matlab.Execute(@"vang2=angleadjust(vang2);");
```

```
        if (iter == 0)
        {
            matlab.Execute(@"p6=optimalpoles(vang2,pmax);");
        }
        matlab.Execute(@"[ma,ar]=pronysim(vang2,p6,pmax);");

        if (chkProny.Checked)
        {
            matlab.Execute(@"[vec]=pronycomp(ma,ar,pmax);");
            db.addVoltageAngle2(getSimArrayProny(), dbTableName);
        }
        else
        {
            db.addVoltageAngle2(getSimArray(), dbTableName);
        }
    }
    if (c7.Checked)
    {
        matlab.PutWorkspaceData("imag0", "base", c7list.Cast<double>().ToArray());
        matlab.Execute(@"imag0=trimwave(imag0,start);");
        if (iter == 0)
        {
            matlab.Execute(@"p7=optimalpoles(imag0,pmax);");
        }
        matlab.Execute(@"[ma,ar]=pronysim(imag0,p7,pmax);");

        if (chkProny.Checked)
        {
            matlab.Execute(@"[vec]=pronycomp(ma,ar,pmax);");
            db.addCurrentMag0(getSimArrayProny(), dbTableName);
        }
        else
        {
            db.addCurrentMag0(getSimArray(), dbTableName);
        }
    }
    if (c8.Checked)
    {
        matlab.PutWorkspaceData("imag1", "base", c8list.Cast<double>().ToArray());
        matlab.Execute(@"imag1=trimwave(imag1,start);");
        if (iter == 0)
        {
            matlab.Execute(@"p8=optimalpoles(imag1,pmax);");
        }
        matlab.Execute(@"[ma,ar]=pronysim(imag1,p8,pmax);");

        if (chkProny.Checked)
        {
            matlab.Execute(@"[vec]=pronycomp(ma,ar,pmax);");
            db.addCurrentMag1(getSimArrayProny(), dbTableName);
        }
        else
        {
            db.addCurrentMag1(getSimArray(), dbTableName);
        }
    }
    if (c9.Checked)
    {
        matlab.PutWorkspaceData("imag2", "base", c9list.Cast<double>().ToArray());
        matlab.Execute(@"imag2=trimwave(imag2,start);");
        if (iter == 0)
        {
            matlab.Execute(@"p9=optimalpoles(imag2,pmax);");
        }
        matlab.Execute(@"[ma,ar]=pronysim(imag2,p9,pmax);");

        if (chkProny.Checked)
        {
            matlab.Execute(@"[vec]=pronycomp(ma,ar,pmax);");
            db.addCurrentMag2(getSimArrayProny(), dbTableName);
        }
```

```
        else
        {
            db.addCurrentMag2(getSimArray(), dbTableName);
        }
    }
    if (c10.Checked)
    {
        matlab.PutWorkspaceData("iang0", "base", c10list.Cast<double>().ToArray());
        matlab.Execute(@"iang0=trimwave(iang0,start);");
        matlab.Execute(@"iang0=angleadjust(iang0);");
        if (iter == 0)
        {
            matlab.Execute(@"p10=optimalpoles(iang0,pmax);");
        }
        matlab.Execute(@"[ma,ar]=pronysim(iang0,p10,pmax);");

        if (chkProny.Checked)
        {
            matlab.Execute(@"[vec]=pronycomp(ma,ar,pmax);");
            db.addCurrentAngle0(getSimArrayProny(), dbTableName);
        }
        else
        {
            db.addCurrentAngle0(getSimArray(), dbTableName);
        }
    }
    if (c11.Checked)
    {
        matlab.PutWorkspaceData("iang1", "base", c11list.Cast<double>().ToArray());
        matlab.Execute(@"iang1=trimwave(iang1,start);");
        matlab.Execute(@"iang1=angleadjust(iang1);");
        if (iter == 0)
        {
            matlab.Execute(@"p11=optimalpoles(iang1,pmax);");
        }
        matlab.Execute(@"[ma,ar]=pronysim(iang1,p11,pmax);");

        if (chkProny.Checked)
        {
            matlab.Execute(@"[vec]=pronycomp(ma,ar,pmax);");
            db.addCurrentAngle1(getSimArrayProny(), dbTableName);
        }
        else
        {
            db.addCurrentAngle1(getSimArray(), dbTableName);
        }
    }
    if (c12.Checked)
    {
        matlab.PutWorkspaceData("iang2", "base", c12list.Cast<double>().ToArray());
        matlab.Execute(@"iang2=trimwave(iang2,start);");
        matlab.Execute(@"iang2=angleadjust(iang2);");
        if (iter == 0)
        {
            matlab.Execute(@"p12=optimalpoles(iang2,pmax);");
        }
        matlab.Execute(@"[ma,ar]=pronysim(iang2,p12,pmax);");

        if (chkProny.Checked)
        {
            matlab.Execute(@"[vec]=pronycomp(ma,ar,pmax);");
            db.addCurrentAngle2(getSimArrayProny(), dbTableName);
        }
        else
        {
            db.addCurrentAngle2(getSimArray(), dbTableName);
        }
    }
    if (c13.Checked)
    {
        matlab.PutWorkspaceData("freq", "base", c13list.Cast<double>().ToArray());
```

```
            matlab.Execute(@"freq=trimwave(freq,start);");
            if (iter == 0)
            {
                matlab.Execute(@"p13=optimalpoles(freq,pmax);");
            }
            matlab.Execute(@"[ma,ar]=pronysim(freq,p13,pmax);");

            if (chkProny.Checked)
            {
                matlab.Execute(@"[vec]=pronycomp(ma,ar,pmax);");
                db.addFreq(getSimArrayProny(), dbTableName);
            }
            else
            {
                db.addFreq(getSimArray(), dbTableName);
            }
        }
        if (c14.Checked)
        {
            matlab.PutWorkspaceData("frat", "base", c14list.Cast<double>().ToArray());
            matlab.Execute(@"frat=trimwave(frat,start);");
            if (iter == 0)
            {
                matlab.Execute(@"p14=optimalpoles(frat,pmax);");
            }
            matlab.Execute(@"[ma,ar]=pronysim(frat,p14,pmax);");

            if (chkProny.Checked)
            {
                matlab.Execute(@"[vec]=pronycomp(ma,ar,pmax);");
                db.addFreqRate(getSimArrayProny(), dbTableName);
            }
            else
            {
                db.addFreqRate(getSimArray(), dbTableName);
            }
        }

        opswitch = false; //optimization control

        sr.Close();

        dummyConsole.AppendLine(string.Format("{0} has been processed",filename));
        txtTest.Text = dummyConsole.ToString();
    }

    private void txtTest_TextChanged(object sender, EventArgs e)
    {
        txtTest.SelectionStart = txtTest.Text.Length;
        txtTest.ScrollToCaret();
    }

    private Array getSimArray()
    {
        object matResult = null;
        object matResult2 = null;
        matlab.GetWorkspaceData("ma", "base", out matResult);
        matlab.GetWorkspaceData("ar", "base", out matResult2);
        System.Array m1 = (System.Array)matResult;
        System.Array m2 = (System.Array)matResult2;

        Array resultArr1 = m1.Cast<double>().ToArray();
        Array resultArr2 = m2.Cast<double>().ToArray();

        Array m = new double[resultArr1.Length + resultArr2.Length];
        resultArr1.CopyTo(m, 0);
        resultArr2.CopyTo(m, resultArr1.Length);

        return m;
    }
```

```
        private Array getSimArrayProny()
        {
            object matResult = null;
            matlab.GetWorkspaceData("vec", "base", out matResult);
            System.Array m1 = (System.Array)matResult;

            Array resultArr1 = m1.Cast<double>().ToArray();

            return resultArr1;
        }
    }


}
```

B.1.2 DatabaseAccess.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;

// The TextFileDataExtractor Application was specifically written for the model validation process
described in this study.
// The code can be used for educational purposes and may not be commercialized.
// Author: Noah Garcia Badayos, 2014

namespace TextFileDataExtractor
{
    class DatabaseAccess
    {
        SqlConnection myConnection;
        static int id = 0;

        public DatabaseAccess()
        {
            initializeConnection();
        }

        public void initializeConnection()
        {
            myConnection = new SqlConnection("server=.\\SQLEXPRESS;" +
                                        "Trusted Connection=yes;" +
                                        "database=psvalidation; " +
                                        "connection timeout=30;" +
                                        "Integrated Security=SSPI");
        }

        public void addScenario(string scenario,string tableName)
        {
            if (this.OpenConnection() == true)
            {
                SqlCommand cmd = new SqlCommand();

                cmd.CommandText = string.Format("UPDATE {0} set resParameter= @scenVal where
id={1}", tableName, id.ToString());
                cmd.Parameters.AddWithValue("@scenVal", scenario);
                cmd.Connection = myConnection;
                cmd.ExecuteNonQuery();

                this.CloseConnection();
            }
        }


        public void addVoltageMag0(Array vMag, string tableName)
        {
            if (this.OpenConnection() == true)
            {
                var parameters = new string[vMag.Length];
```

```
                var parStatement = new string[vMag.Length];

                SqlCommand cmd = new SqlCommand();
                for (int i = 0; i < vMag.Length; i++)
                {
                    parameters[i] = string.Format("@vmz{0}", i);
                    parStatement[i] = string.Format("vmz{0}=@vmz{0}", i);
                    cmd.Parameters.AddWithValue(parameters[i], vMag.GetValue(i));
                }

                cmd.CommandText = string.Format("UPDATE {0} set {1} WHERE id={2}", tableName,
string.Join(", ", parStatement), id.ToString());
                cmd.Connection = myConnection;
                cmd.ExecuteNonQuery();

                this.CloseConnection();
            }
        }

        public void addVoltageMag1(Array vMag, string tableName)
        {
            if (this.OpenConnection() == true)
            {
                var parameters = new string[vMag.Length];
                var parStatement = new string[vMag.Length];

                SqlCommand cmd = new SqlCommand();
                for (int i = 0; i < vMag.Length; i++)
                {
                    parameters[i] = string.Format("@vmp{0}", i);
                    parStatement[i] = string.Format("vmp{0}=@vmp{0}", i);
                    cmd.Parameters.AddWithValue(parameters[i], vMag.GetValue(i));
                }

                cmd.CommandText = string.Format("UPDATE {0} set {1} WHERE id={2}", tableName,
string.Join(", ", parStatement), id.ToString());
                cmd.Connection = myConnection;
                cmd.ExecuteNonQuery();

                this.CloseConnection();
            }
        }

        public void addVoltageMag2(Array vMag, string tableName)
        {
            if (this.OpenConnection() == true)
            {
                var parameters = new string[vMag.Length];
                var parStatement = new string[vMag.Length];

                SqlCommand cmd = new SqlCommand();
                for (int i = 0; i < vMag.Length; i++)
                {
                    parameters[i] = string.Format("@vmn{0}", i);
                    parStatement[i] = string.Format("vmn{0}=@vmn{0}", i);
                    cmd.Parameters.AddWithValue(parameters[i], vMag.GetValue(i));
                }

                cmd.CommandText = string.Format("UPDATE {0} set {1} WHERE id={2}", tableName,
string.Join(", ", parStatement), id.ToString());
                cmd.Connection = myConnection;
                cmd.ExecuteNonQuery();

                this.CloseConnection();
            }
        }

        public void addVoltageAngle0(Array vAng, string tableName)
        {
            if (this.OpenConnection() == true)
            {
```

```
                var parameters = new string[vAng.Length];
                var parStatement = new string[vAng.Length];

                SqlCommand cmd = new SqlCommand();
                for (int i = 0; i < vAng.Length; i++)
                {
                    parameters[i] = string.Format("@vaz{0}", i);
                    parStatement[i] = string.Format("vaz{0}=@vaz{0}", i);
                    cmd.Parameters.AddWithValue(parameters[i], vAng.GetValue(i));
                }

                cmd.CommandText = string.Format("UPDATE {0} set {1} WHERE id={2}", tableName,
string.Join(", ", parStatement), id.ToString());
                cmd.Connection = myConnection;
                cmd.ExecuteNonQuery();

                this.CloseConnection();
            }
        }

        public void addVoltageAngle1(Array vAng, string tableName)
        {
            if (this.OpenConnection() == true)
            {
                var parameters = new string[vAng.Length];
                var parStatement = new string[vAng.Length];

                SqlCommand cmd = new SqlCommand();
                for (int i = 0; i < vAng.Length; i++)
                {
                    parameters[i] = string.Format("@vap{0}", i);
                    parStatement[i] = string.Format("vap{0}=@vap{0}", i);
                    cmd.Parameters.AddWithValue(parameters[i], vAng.GetValue(i));
                }

                cmd.CommandText = string.Format("UPDATE {0} set {1} WHERE id={2}", tableName,
string.Join(", ", parStatement), id.ToString());
                cmd.Connection = myConnection;
                cmd.ExecuteNonQuery();

                this.CloseConnection();
            }
        }

        public void addVoltageAngle2(Array vAng, string tableName)
        {
            if (this.OpenConnection() == true)
            {
                var parameters = new string[vAng.Length];
                var parStatement = new string[vAng.Length];

                SqlCommand cmd = new SqlCommand();
                for (int i = 0; i < vAng.Length; i++)
                {
                    parameters[i] = string.Format("@van{0}", i);
                    parStatement[i] = string.Format("van{0}=@van{0}", i);
                    cmd.Parameters.AddWithValue(parameters[i], vAng.GetValue(i));
                }

                cmd.CommandText = string.Format("UPDATE {0} set {1} WHERE id={2}", tableName,
string.Join(", ", parStatement), id.ToString());
                cmd.Connection = myConnection;
                cmd.ExecuteNonQuery();

                this.CloseConnection();
            }
        }

        public void addCurrentMag0(Array iMag, string tableName)
        {
            if (this.OpenConnection() == true)
```

```
            {
                var parameters = new string[iMag.Length];
                var parStatement = new string[iMag.Length];

                SqlCommand cmd = new SqlCommand();
                for (int i = 0; i < iMag.Length; i++)
                {
                    parameters[i] = string.Format("@imz{0}", i);
                    parStatement[i] = string.Format("imz{0}=@imz{0}", i);
                    cmd.Parameters.AddWithValue(parameters[i], iMag.GetValue(i));
                }

                cmd.CommandText = string.Format("UPDATE {0} set {1} WHERE id={2}", tableName,
string.Join(", ", parStatement), id.ToString());
                cmd.Connection = myConnection;
                cmd.ExecuteNonQuery();

                this.CloseConnection();
            }
        }

        public void addCurrentMag1(Array iMag, string tableName)
        {
            if (this.OpenConnection() == true)
            {
                var parameters = new string[iMag.Length];
                var parStatement = new string[iMag.Length];

                SqlCommand cmd = new SqlCommand();
                for (int i = 0; i < iMag.Length; i++)
                {
                    parameters[i] = string.Format("@imp{0}", i);
                    parStatement[i] = string.Format("imp{0}=@imp{0}", i);
                    cmd.Parameters.AddWithValue(parameters[i], iMag.GetValue(i));
                }

                cmd.CommandText = string.Format("UPDATE {0} set {1} WHERE id={2}", tableName,
string.Join(", ", parStatement), id.ToString());
                cmd.Connection = myConnection;
                cmd.ExecuteNonQuery();

                this.CloseConnection();
            }
        }

        public void addCurrentMag2(Array iMag, string tableName)
        {
            if (this.OpenConnection() == true)
            {
                var parameters = new string[iMag.Length];
                var parStatement = new string[iMag.Length];

                SqlCommand cmd = new SqlCommand();
                for (int i = 0; i < iMag.Length; i++)
                {
                    parameters[i] = string.Format("@imn{0}", i);
                    parStatement[i] = string.Format("imn{0}=@imn{0}", i);
                    cmd.Parameters.AddWithValue(parameters[i], iMag.GetValue(i));
                }

                cmd.CommandText = string.Format("UPDATE {0} set {1} WHERE id={2}", tableName,
string.Join(", ", parStatement), id.ToString());
                cmd.Connection = myConnection;
                cmd.ExecuteNonQuery();

                this.CloseConnection();
            }
        }

        public void addCurrentAngle0(Array iAng, string tableName)
        {
```

```
            if (this.OpenConnection() == true)
            {
                var parameters = new string[iAng.Length];
                var parStatement = new string[iAng.Length];

                SqlCommand cmd = new SqlCommand();
                for (int i = 0; i < iAng.Length; i++)
                {
                    parameters[i] = string.Format("@iaz{0}", i);
                    parStatement[i] = string.Format("iaz{0}=@iaz{0}", i);
                    cmd.Parameters.AddWithValue(parameters[i], iAng.GetValue(i));
                }

                cmd.CommandText = string.Format("UPDATE {0} set {1} WHERE id={2}", tableName,
string.Join(", ", parStatement), id.ToString());
                cmd.Connection = myConnection;
                cmd.ExecuteNonQuery();

                this.CloseConnection();
            }
        }

        public void addCurrentAngle1(Array iAng, string tableName)
        {
            if (this.OpenConnection() == true)
            {
                var parameters = new string[iAng.Length];
                var parStatement = new string[iAng.Length];

                SqlCommand cmd = new SqlCommand();
                for (int i = 0; i < iAng.Length; i++)
                {
                    parameters[i] = string.Format("@iap{0}", i);
                    parStatement[i] = string.Format("iap{0}=@iap{0}", i);
                    cmd.Parameters.AddWithValue(parameters[i], iAng.GetValue(i));
                }

                cmd.CommandText = string.Format("UPDATE {0} set {1} WHERE id={2}", tableName,
string.Join(", ", parStatement), id.ToString());
                cmd.Connection = myConnection;
                cmd.ExecuteNonQuery();

                this.CloseConnection();
            }
        }

        public void addCurrentAngle2(Array iAng, string tableName)
        {
            if (this.OpenConnection() == true)
            {
                var parameters = new string[iAng.Length];
                var parStatement = new string[iAng.Length];

                SqlCommand cmd = new SqlCommand();
                for (int i = 0; i < iAng.Length; i++)
                {
                    parameters[i] = string.Format("@ian{0}", i);
                    parStatement[i] = string.Format("ian{0}=@ian{0}", i);
                    cmd.Parameters.AddWithValue(parameters[i], iAng.GetValue(i));
                }

                cmd.CommandText = string.Format("UPDATE {0} set {1} WHERE id={2}", tableName,
string.Join(", ", parStatement), id.ToString());
                cmd.Connection = myConnection;
                cmd.ExecuteNonQuery();

                this.CloseConnection();
            }
        }
        public void addFreq(Array freq, string tableName)
        {
```

```
            if (this.OpenConnection() == true)
            {
                var parameters = new string[freq.Length];
                var parStatement = new string[freq.Length];

                SqlCommand cmd = new SqlCommand();
                for (int i = 0; i < freq.Length; i++)
                {
                    parameters[i] = string.Format("@freq{0}", i);
                    parStatement[i] = string.Format("freq{0}=@freq{0}", i);
                    cmd.Parameters.AddWithValue(parameters[i], freq.GetValue(i));
                }

                cmd.CommandText = string.Format("UPDATE {0} set {1} WHERE id={2}", tableName,
string.Join(", ", parStatement), id.ToString());
                cmd.Connection = myConnection;
                cmd.ExecuteNonQuery();

                this.CloseConnection();
            }
        }

        public void addFreqRate(Array fRate, string tableName)
        {
            if (this.OpenConnection() == true)
            {
                var parameters = new string[fRate.Length];
                var parStatement = new string[fRate.Length];

                SqlCommand cmd = new SqlCommand();
                for (int i = 0; i < fRate.Length; i++)
                {
                    parameters[i] = string.Format("@frat{0}", i);
                    parStatement[i] = string.Format("frat{0}=@frat{0}", i);
                    cmd.Parameters.AddWithValue(parameters[i], fRate.GetValue(i));
                }

                cmd.CommandText = string.Format("UPDATE {0} set {1} WHERE id={2}", tableName,
string.Join(", ", parStatement), id.ToString());
                cmd.Connection = myConnection;
                cmd.ExecuteNonQuery();

                this.CloseConnection();
            }
        }

        public void incrementID(string tableName)
        {
            id++;

            if (this.OpenConnection() == true)
            {
                SqlCommand cmd = new SqlCommand();

                cmd.CommandText = string.Format("INSERT INTO {0} (id) values ({1})", tableName,
id.ToString());
                cmd.Connection = myConnection;
                cmd.ExecuteNonQuery();

                this.CloseConnection();
            }
        }

        public void setStartID(int sID)
        {
            id = sID - 1;
        }

        private bool OpenConnection()
        {
            try
```

```
            {
                myConnection.Open();
                return true;
            }
            catch (SqlException ex)
            {
                switch (ex.Number)
                {
                    case 0:
                        System.Console.WriteLine("Cannot connect to server. Contact
administrator");
                        break;

                    case 1045:
                        System.Console.WriteLine("Invalid username/password, please try again");
                        break;
                }
                return false;
            }
        }

        private bool CloseConnection()
        {
            try
            {
                myConnection.Close();
                return true;
            }
            catch (SqlException ex)
            {
                System.Console.WriteLine(ex.Message);
                return false;
            }
        }
    }
}
```

B.2 Schema Generator

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;

// The Schema Generator Application was specifically written for the model validation process
described in this study.
// The code can be used for educational purposes and may not be commercialized.
// Author: Noah Garcia Badayos, 2014

namespace SchemaGenerator
{
    public partial class frmMain : Form
    {
        SaveFileDialog dialog = new SaveFileDialog();
        System.IO.Stream fileStream;
        int width = 0;

        public frmMain()
        {
            InitializeComponent();
            dialog.Title = "Save file as...";
            dialog.Filter = "SQL Scripts (*.sql)|*.sql|All files (*.*)|*.*";
            dialog.RestoreDirectory = true;
        }
```

```
        private void btnSave_Click(object sender, EventArgs e)
        {
            if (dialog.ShowDialog() == DialogResult.OK)
            {
                fileStream = dialog.OpenFile();
                btnSave.Enabled = false;
                btnRun.Enabled = true;
            }
        }

        private void btnRun_Click(object sender, EventArgs e)
        {
            width = Convert.ToInt16(txtWidth.Text);
            StreamWriter sw = new StreamWriter(fileStream);
            startingScript(sw);
            predictorScript(sw);
            endScript(sw);
            sw.Flush();
            sw.Close();
            MessageBox.Show("Schema has been created and saved!");
            btnSave.Enabled = true;
            btnRun.Enabled = false;
        }

        private void startingScript(StreamWriter sw)
        {
            sw.Write(string.Format("use psvalidation; \r\n" +
                "IF (EXISTS (SELECT * FROM INFORMATION_SCHEMA.TABLES  WHERE  TABLE_NAME = '{0}'))
\r\n" +
                "BEGIN \r\n" +
                "   drop table {0}; \r\n" +
                "END \r\n" +
            "create table {0} \r\n" +
            "(id INTEGER PRIMARY KEY, \r\n" ,Convert.ToString(txtTableName.Text)));
        }

        private void predictorScript(StreamWriter sw)
        {
            if (c1.Checked)
            {
                for (int i = 0; i < width; i++)
                {
                    sw.WriteLine(string.Format("vmz{0} FLOAT,",i));
                }
            }
            if (c2.Checked)
            {
                for (int i = 0; i < width; i++)
                {
                    sw.WriteLine(string.Format("vmp{0} FLOAT,", i));
                }
            }
            if (c3.Checked)
            {
                for (int i = 0; i < width; i++)
                {
                    sw.WriteLine(string.Format("vmn{0} FLOAT,", i));
                }
            }
            if (c4.Checked)
            {
                for (int i = 0; i < width; i++)
                {
                    sw.WriteLine(string.Format("vaz{0} FLOAT,", i));
                }
            }
            if (c5.Checked)
            {
                for (int i = 0; i < width; i++)
                {
```

```
            sw.WriteLine(string.Format("vap{0} FLOAT,", i));
        }
    }
    if (c6.Checked)
    {
        for (int i = 0; i < width; i++)
        {
            sw.WriteLine(string.Format("van{0} FLOAT,", i));
        }
    }
    if (c7.Checked)
    {
        for (int i = 0; i < width; i++)
        {
            sw.WriteLine(string.Format("imz{0} FLOAT,", i));
        }
    }
    if (c8.Checked)
    {
        for (int i = 0; i < width; i++)
        {
            sw.WriteLine(string.Format("imp{0} FLOAT,", i));
        }
    }
    if (c9.Checked)
    {
        for (int i = 0; i < width; i++)
        {
            sw.WriteLine(string.Format("imn{0} FLOAT,", i));
        }
    }
    if (c10.Checked)
    {
        for (int i = 0; i < width; i++)
        {
            sw.WriteLine(string.Format("iaz{0} FLOAT,", i));
        }
    }
    if (c11.Checked)
    {
        for (int i = 0; i < width; i++)
        {
            sw.WriteLine(string.Format("iap{0} FLOAT,", i));
        }
    }
    if (c12.Checked)
    {
        for (int i = 0; i < width; i++)
        {
            sw.WriteLine(string.Format("ian{0} FLOAT,", i));
        }
    }
    if (c13.Checked)
    {
        for (int i = 0; i < width; i++)
        {
            sw.WriteLine(string.Format("freq{0} FLOAT,", i));
        }
    }
    if (c14.Checked)
    {
        for (int i = 0; i < width; i++)
        {
            sw.WriteLine(string.Format("frat{0} FLOAT,", i));
        }
    }
}

private void endScript(StreamWriter sw)
{
    sw.WriteLine("resParameter VARCHAR(20),");
```

```
            sw.Write(");");
        }

        private void btnClose_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

Appendix C Matlab Prony Codes

C.1 Original HPRONY by A. Swami

```
function  [a,theta,alpha,fr] = hprony (x,p)
%HPRONY Prony's method for modeling transients
%   [a,theta,alpha,fr] = hprony (x,p)
%   x  - transient signal
%   p  - order    [default = length(x)/10]
%   HPRONY fits the model:
%        x(n) = sum_{1 to p} A(k) z(k)^n
%   where A(k) = a(k) exp (j * theta(k))
%   and   z(k) = exp( alpha(k) + j * 2 * pi * fr (k) )
%   The estimated values of a(k), theta(k), alpha(k) and fr(k)
%      k=1,...,p, are returned in the vectors, a, theta, alpha and fr.


%  Copyright (c) 1991-2001 by United Signals & Systems, Inc.
%       $Revision: 1.7 $
%  A. Swami   January 20, 1995

%     RESTRICTED RIGHTS LEGEND
% Use, duplication, or disclosure by the Government is subject to
% restrictions as set forth in subparagraph (c) (1) (ii) of the
% Rights in Technical Data and Computer Software clause of DFARS
% 252.227-7013.
% Manufacturer: United Signals & Systems, Inc., P.O. Box 2374,
% Culver City, California 90231.
%
%  This material may be reproduced by or for the U.S. Government pursuant
%  to the copyright license under the clause at DFARS 252.227-7013.

%------------- parameter checks --------
[nsamp,ncol] = size(x);
if (min(nsamp,ncol) ~= 1)
   error('hprony: x must be a vector');
end

if (nsamp == 1)
   nsamp = ncol; ncol=1; x = x(:);
end

ind = find(abs(x)>0);           %  leading/trailing zeros cause problems
x   = x(min(ind):max(ind)); %  with prony
nsamp = length(x);
xone  = x(1);
x     = x/x(1);                 % to handle v3.5/v4.0 toeplitz incompatibility

if (exist('p') ~= 1) p = fix(nsamp/10); end

% --------------------------
[ma,ar] = prony (x, p-1, p);            % fit an ARMA model

% junk = toeplitz([1; j]);      % to handle v3.5/v4.0 toeplitz incompatibility
                % prony calls toeplitz
% if (junk(2,1) ~= j )
%    ma = conj(ma);  ar = conj(ar);
% end

ma = ma * xone;

[res,poles,delay] = residue(ma,ar);   % convert to poles-residues

a = abs(res);                         % required conversions
theta = angle(res);
alpha = real(log(poles));
fr    = imag(log(poles)) / (2*pi);
```

```
% ---------------------- plot the data and the model ---------------
xest = ( (a .* exp(j*theta)) * ones(1,nsamp) ) .*  ...
    exp( (j * 2*pi* fr + alpha) * [0:(nsamp-1)] );
if (p > 1) xest = sum(xest).';
else       xest = xest(:);
end
x    = x * xone;

if (exist('debug') ==  1)
  if (all(imag(x) == 0))
    plot([x,xest]),
    title('True (-) and estimated (--)  data')
  else
    clf, subplot(211)
    plot(real([x,xest])),
    title('True (-) and estimated (--)  data: real part')
    subplot(212)
    plot(imag([x,xest])),
    title('True (-) and estimated (--)  data: imaginary part')
  end
  mse = (x-xest)'  * (x-xest) / length(x);
  disp(['Mean squared error = ',num2str(mse)])
end


% ---------------------- display parameter estimates ------------

if (all(imag(x)==0))
   disp('AR parameters are:')
   disp(ar')
   disp('MA parameters are:')
   disp(ma')
   disp(['delay = ',int2str(delay)])
else
   format compact
   disp('amplitudes, thetas, alphas, frequencies')
   disp(a.'), disp(theta.'), disp(alpha.'), disp(fr.')
end

return
```

C.2 Prony Codes For Study

These codes are directly or indirectly called inside the custom C# application (Appendix B) used in the study.

C.2.1 Pronycheck()

```
function  [ma,ar,mse] = pronycheck (x,p)
% PRONYCHECK solves for transfer function coefficients and the mean squared
% error (MSE) between input signal and the estimate. This is a modified
% version of HPRONY, written by A. Swami. The open source code can be used
% for educational purposes and may not be commercialized.
% Author: Noah Garcia Badayos, 2014

%------------- parameter checks --------
[nsamp,ncol] = size(x);
if (min(nsamp,ncol) ~= 1)
   error('pronycheck: x must be a vector');
end

if (nsamp == 1)
   nsamp = ncol; ncol=1; x = x(:);
end
```

```
ind = find(abs(x)>0);           %  leading/trailing zeros cause problems
x   = x(min(ind):max(ind)); %  with prony
nsamp = length(x);
xone  = x(1);
x     = x/x(1);                 % to handle v3.5/v4.0 toeplitz incompatibility

% --------------------------
[ma,ar] = prony (x, p-1, p);            % fit an ARMA model

ma = ma * xone;

[res,poles,delay] = residue(ma,ar);   % convert to poles-residues

a = abs(res);                         % required conversions
theta = angle(res);
alpha = real(log(poles));
fr    = imag(log(poles)) / (2*pi);

% ---------------------- error computation ---------------
xest = ( (a .* exp(j*theta)) * ones(1,nsamp) ) .*  ...
    exp( (j * 2*pi* fr + alpha) * [0:(nsamp-1)] );
if (p > 1) xest = sum(xest).';
else       xest = xest(:);
end
x    = x * xone;
mse = (x-xest)'  * (x-xest);
```

C.2.2 Optimalpoles()

```
function  [p] = optimalpoles (x,pmax)
% OPTIMALPOLES determines the optimal number of poles based on the fit of
% the recomposed signals from the Prony decomposition. The open source code
% can be used for educational purposes and may not be commercialized.
% Author: Noah Garcia Badayos, 2014

eval = 0;
p = 0;
diff = max(x)-min(x);
valRat = abs(diff/min(x));

if(valRat > 0.00001)
    for j=1:pmax
        [b,a,mse] = pronycheck(x,j);

        if (j == 1)
            eval = mse;
            p = j;
            continue;
        end
        if(isnan(mse))
            break;
        end
        if(mse < eval)
            eval = mse;
            p = j;
        end
    end
end
```

C.2.3 Pronysim()

```
function  [maz,arz] = pronysim (x,p,pmax)
%PRONYSIM solves for the transfer function coeffients after the optimal
% number of poles is determined. This is a modified version of HPRONY,
% written by A. Swami. The open source code can be used for educational
% purposes and may not be commercialized.
% Author: Noah Garcia Badayos, 2014

%------------- parameter checks --------
[nsamp,ncol] = size(x);
```

```
if (min(nsamp,ncol) ~= 1)
   error('pronysim: x must be a vector');
end

if (nsamp == 1)
   nsamp = ncol; ncol=1; x = x(:);
end

ind = find(abs(x)>0);           %  leading/trailing zeros cause problems
x   = x(min(ind):max(ind)); %  with prony
nsamp = length(x);
xone  = x(1);
x     = x/x(1);                 % to handle v3.5/v4.0 toeplitz incompatibility

% ---------------------------
[ma,ar] = prony (x, p-1, p);            % fit an ARMA model

ma = ma * xone;

maz = zeros(1,pmax);
arz = zeros(1,pmax+1);

maz(1:length(ma)) = ma;
arz(1:length(ar)) = ar;
```

C.2.4 Pronycomp()

```
function  [vec] = pronycomp(ma,ar,pmax)
% PRONYCOMP Solves for the decaying sinusoidal signals' properties. The
% open source code can be used for educational purposes and may not be
% commercialized.
% Author: Noah Garcia Badayos, 2014

[res,poles,delay] = residue(ma,ar);    % convert to poles-residues

a = abs(res);                          % required conversions
c1 = zeros(1,pmax);
c1(1:length(a)) = a;

theta = angle(res);
c2 = zeros(1,pmax);
c2(1:length(theta)) = theta;

alpha = real(log(poles));
c3 = zeros(1,pmax);
c3(1:length(alpha)) = alpha;

fr = imag(log(poles)) / (2*pi);
c4 = zeros(1,pmax);
c4(1:length(fr)) = fr;

vec = [c1 c2 c3 c4]';
```

C.2.5 Angleadjust()

```
function [naVal] = angleadjust(aVal)
% ANGLEADJUST performs angle array correction for angle jumps found in
% DIgSILENT measurements. The open source code can be used
% for educational purposes and may not be commercialized.
% Author: Noah Garcia Badayos, 2014

jump = 340;
nlength = length(aVal);

mark1 = zeros(1,2); %mark indicates -1 for negative jump, 1 for positive, 0 for reset, and the
corresponding point
mark2 = zeros(1,2); %e.g. mark1[-1,23] indicates a negative jump at point 23

prev = aVal(1); %container

for i=2:nlength
```

```
   diff = aVal(i) - prev;
   if(diff > jump)
      if(mark1(1,1) == 0)
          mark1(1,1) = 1;
          mark1(1,2) = i;
      else
          mark2(1,1) = 1;
          mark2(1,2) = i;
      end
   elseif (diff < -jump)
       if(mark1(1,1) == 0)
          mark1(1,1) = -1;
          mark1(1,2) = i;
       else
          mark2(1,1) = -1;
          mark2(1,2) = i;
       end
   end

   if(mark2(1,1) ~= 0)
      if(mark1(1,1) == 1 && mark2(1,1) == -1) %positive jump and return
         adjust = -360;
      elseif(mark1(1,1) == -1 && mark2(1,1) == 1) %negative jump and return
         adjust = 360;
      elseif(mark1(1,1) == mark2(1,1)) %unstable case truncates array (note: the time array should
be truncated too for plotting)
         aVal = aVal(1:mark1(1,2)-1);
         mark1 = zeros(1,2);
         break;
      end

      for p = mark1(1,2):mark2(1,2) - 1 %adjustment
         aVal(p) = aVal(p) + adjust;
      end

      mark1 = zeros(1,2);
      mark2 = zeros(1,2);
   end

   prev = aVal(i);
end

if(mark1(1,1) ~= 0) %single positive or negative jump
    if(mark1(1,1) == 1)
       adjust = -360;
    elseif(mark1(1,1) == -1)
       adjust = 360;
    end

    for p = mark1(1,2):nlength %adjustment
       aVal(p) = aVal(p) + adjust;
    end
end

naVal = aVal;
```

C.3 Prony Test Codes

These codes were mainly used for evaluation of the Prony decomposition for illustration purposes.

C.3.1 Oprony()

```
function  [maz,arz] = oprony (x,pmax)
% OPRONY solves for the transfer function coefficients based on the optimal
% number of poles. The open source code can be used for educational
% purposes and may not be commercialized.
% Author: Noah Garcia Badayos, 2014
```

```
eval = 0;
diff = max(x)-min(x);
valRat = abs(diff/min(x));
ma = 0;
ar = 0;

if(valRat > 0.00001)
    for j=1:pmax
        [b,a,mse] = pronycheck(x,j);

        if (j == 1)
            eval = mse;
            ma = b;
            ar = a;
            continue;
        end
        if(isnan(mse))
            break;
        end
        if(mse < eval)
            eval = mse;
            ma = b;
            ar = a;
        end
    end
end

maz = zeros(1,pmax);
arz = zeros(1,pmax+1);

maz(1:length(ma)) = ma;
arz(1:length(ar)) = ar;
```

C.3.2 Pdisplay()

```
function pDisplay (t,x,p)
% PDISPLAY plots the decaying sinusoidal components of the input signal for
% evaluation of Prony decomposition. This is a modified
% version of HPRONY, written by A. Swami. The open source code can be used
% for educational purposes and may not be commercialized.
% Author: Noah Garcia Badayos, 2014

%------------- parameter checks --------
[nsamp,ncol] = size(x);
if (min(nsamp,ncol) ~= 1)
   error('pronycheck: x must be a vector');
end

if (nsamp == 1)
   nsamp = ncol; ncol=1; x = x(:);
end

ind = find(abs(x)>0);           %  leading/trailing zeros cause problems
x   = x(min(ind):max(ind)); %  with prony

nsamp = length(x);
xone  = x(1);
x     = x/x(1);                 % to handle v3.5/v4.0 toeplitz incompatibility

% --------------------------
[ma,ar] = prony (x, p-1, p);            % fit an ARMA model

ma = ma * xone;

[res,poles,delay] = residue(ma,ar);   % convert to poles-residues

a = abs(res);                         % required conversions
theta = angle(res);
alpha = real(log(poles));
fr    = imag(log(poles)) / (2*pi);
```

```
% ---------------------- error computation ---------------
xest = ( (a .* exp(j*theta)) * ones(1,nsamp) ) .*  ...
    exp( (j * 2*pi* fr + alpha) * [0:(nsamp-1)] );


for c = 1:p
    rndclr = [rand, rand, rand];
    pline = plot(t,xest(c,:),'color',rndclr);
    xlabel('Time (sec)');
    ylabel('Magnitude');
    hold on;
end
```

C.3.3 Wavecheck()

```
function wavecheck(t,x,start,pmax)
% WAVECHECK plots input signal against recombined Prony-decomposed signals.
% The open source code can be used for educational purposes and may not be
% commercialized.
% Author: Noah Garcia Badayos, 2014

len = length(t);

t = t(start:len);
x = x(start:len);

[ma,ar] = oprony(x,pmax);

h = impz(ma,ar,length(x));

lin1 = plot(t,x);
set(lin1,'Color','black','LineWidth',1);
hold on;
lin2 = plot(t,h);
set(lin2,'Color','blue','LineWidth',1,'LineStyle', '--');

title('Prony Synthesis');
xlabel('Time (sec)');
ylabel('Magnitude');
```

Appendix D Three-Phase Test Confusion Matrix Tables

D.1 Test Without Current Measurement

=== Confusion Matrix ===

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	aa	ab	ac	ad	ae	af	ag	ah	ai	aj	ak	<-- classified as
49	0	0	0	0	0	0	0	0	0	0	0	0	4	0	3	0	0	4	0	0	0	0	1	1	3	1	0	0	0	9	25	0	0	0	0	0	a = Base
0	83	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	b = Xdup
0	0	87	0	1	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	c = Xddn
0	1	0	98	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	d = Xqup
0	0	6	0	88	0	0	0	1	2	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	e = Xqdn
0	1	0	0	0	98	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	f = Tdsup
0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	g = Xdsup
0	0	0	0	0	0	0	97	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	h = Xdsdn
0	0	0	0	0	0	0	0	97	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	i = Tdssdn
0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	j = Tqssdn
0	0	0	0	0	0	1	0	0	0	99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	k = Xdssdn
0	0	0	0	0	0	0	0	0	0	0	99	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	l = Xqssdn
0	0	0	0	0	0	1	0	0	0	0	0	89	4	1	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	1	0	0	0	0	0	0	m = Xlup
2	0	0	1	0	0	0	0	0	0	0	0	1	85	4	2	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	2	0	0	0	0	0	n = Trup
3	0	0	0	0	0	0	0	1	0	0	0	1	1	70	2	3	0	11	0	0	0	0	2	0	1	1	0	0	0	1	3	0	0	0	0	0	o = Trdn
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	83	0	0	1	0	0	0	0	0	1	8	0	0	2	0	1	1	0	0	0	0	1	p = Kaup
0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	83	0	1	0	0	0	0	4	1	0	0	0	0	2	0	2	2	0	0	1	1	q = Kadn
0	0	0	0	0	0	0	0	0	0	0	0	4	3	0	0	0	91	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	r = Taup
6	0	0	0	0	0	0	0	0	0	0	0	0	0	11	1	3	0	49	0	0	0	0	3	1	2	3	1	0	1	6	9	0	2	0	0	2	s = Tadn
0	0	0	0	0	0	0	0	0	0	0	4	5	0	0	0	0	0	0	91	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	t = Keup
0	0	11	0	0	0	4	0	0	0	0	0	0	0	3	0	0	0	1	0	81	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	u = Kedn
0	3	0	1	0	0	0	0	0	0	0	0	2	0	1	0	0	0	0	0	0	92	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	v = Teup
0	0	4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	91	0	0	0	0	0	0	0	0	0	2	0	0	1	0	w = Kfup
3	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	2	0	0	0	0	0	0	88	2	0	0	0	0	0	1	0	0	0	0	0	1	x = Tfup
5	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2	0	0	4	0	0	0	0	0	34	1	9	0	1	8	15	6	1	4	3	5	1	y = Rup
17	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	6	0	0	0	0	0	3	23	7	0	9	1	4	16	1	0	4	1	6	z = Rdn
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	26	0	3	16	1	1	9	3	24	10	2	aa = Rtup
0	0	0	0	0	0	0	0	1	0	0	0	0	0	6	1	0	0	1	0	0	0	0	1	1	1	1	62	3	0	0	2	1	15	0	0	4	ab = Rtdn
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	8	1	0	35	0	5	7	1	16	0	1	16	ac = GTrup
1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	13	0	7	0	0	40	4	0	5	0	16	12	0	ad = GTrdn
19	0	0	0	0	0	0	0	0	0	0	0	1	4	3	0	0	0	11	0	0	0	0	0	8	2	4	0	0	0	33	9	0	0	2	3	1	ae = GTfup
21	0	0	0	0	0	0	0	0	0	0	0	1	2	0	3	0	0	6	0	0	0	0	0	1	8	2	1	3	1	11	36	0	0	1	1	2	af = GTfdn
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	2	0	0	0	0	0	0	3	6	0	5	0	0	5	1	0	70	0	3	3	0	ag = Twup
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	1	3	1	3	1	12	0	0	2	0	61	0	0	14	ah = Atup
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	8	0	18	0	2	23	0	0	7	1	26	11	1	ai = Atdn
1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	14	1	15	0	0	22	5	1	5	0	18	15	0	aj = Tgup
4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	0	0	0	0	1	0	7	0	0	14	0	0	5	0	2	0	0	64	ak = Tgdn

D.2 Test With Current Measurement

=== Confusion Matrix ===

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	aa	ab	ac	ad	ae	af	ag	ah	ai	aj	ak	<-- classified as
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	9	1	0	1	0	21	21	0	0	0	2	1	a = Base
0	84	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	b = Xdup
0	0	84	0	0	0	3	0	1	0	0	0	1	0	0	0	0	0	0	0	10	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	c = Xddn
0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	d = Xqup
0	0	5	0	90	0	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	e = Xqdn
0	0	0	0	0	99	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	f = Tdsup
0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	g = Xdsup
0	0	0	0	0	0	0	95	0	0	0	0	4	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	h = Xdsdn
0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	i = Tdssdn
0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	j = Tqssdn
0	0	0	0	0	0	0	0	0	0	99	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	k = Xdssdn
0	0	0	0	0	0	0	0	0	0	0	99	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	l = Xqssdn
0	0	0	0	0	0	0	0	0	0	0	0	98	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	m = Xlup
1	0	0	0	0	0	0	0	0	0	0	0	1	91	0	5	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	n = Trup
0	0	0	0	0	0	0	0	0	0	0	0	0	0	79	2	0	0	18	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	o = Trdn
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	97	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	p = Kaup
1	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	81	0	1	0	0	0	0	0	1	0	1	0	0	2	3	0	1	0	1	2	0	q = Kadn
0	0	0	0	0	0	0	0	0	0	0	0	7	1	0	0	0	92	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	r = Taup
1	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0	81	0	0	0	0	1	0	0	0	0	0	0	2	3	0	0	0	0	0	s = Tadn
0	0	0	0	0	0	0	0	0	0	0	5	8	2	0	0	0	0	0	85	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	t = Keup
0	0	6	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	92	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	u = Kedn
0	1	0	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	v = Teup
0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	92	0	0	0	1	0	0	0	0	0	2	0	0	0	0	w = Kfup
0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	94	0	0	0	0	0	0	0	2	0	0	0	0	1	x = Tfup
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	39	3	6	0	6	6	13	9	0	2	8	4	0	y = Rup
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	3	35	4	0	8	5	10	14	0	2	5	1	1	z = Rdn
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	44	0	2	6	2	1	1	1	24	16	0	aa = Rtup
2	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	3	0	66	9	0	0	2	0	12	0	0	3	ab = Rtdn
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	9	1	0	51	2	0	11	0	11	1	0	2	ac = GTrup
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	3	9	0	0	37	5	3	0	0	27	5	0	ad = GTrdn
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	8	2	0	0	1	42	16	0	0	3	2	0	ae = GTfup
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	9	2	0	7	1	10	30	0	1	1	0	1	af = GTfdn
0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	2	0	1	0	0	0	0	0	1	1	2	0	0	2	3	1	81	0	1	2	0	ag = Twup
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	2	5	4	0	20	1	0	2	0	47	3	1	12	ah = Atup
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	15	0	2	14	1	2	0	0	55	5	0	ai = Atdn
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	1	29	0	1	16	6	2	0	0	20	13	0	aj = Tgup
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	8	0	0	8	0	1	7	0	4	0	0	66	ak = Tgdn