From Static to Dynamic Electric Power Network State Estimation: The Role of Bus Component Dynamics

Submitted in partial fulfillment of the requirements for
the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Ellery A. Blood

B.S., Computer and Systems Engineering, Rensselaer Polytechnic Institute
M.S.E.S., Mechanical Engineering, Naval Postgraduate School
M.S., Electrical and Computer Engineering, Carnegie Mellon University

Carnegie Mellon University
Pittsburgh, PA

May, 2011

Abstract

This thesis addresses the challenge of accurately and robustly estimating the network state on an electric power network despite its large size, infrequent measurement updates, and high likelihood of corrupted data. This is especially important as electrical transmission operators are increasingly being asked to operate the networks at their maximum allowable capacity. Accurate knowledge of the state is necessary to ensure adequate margin to these operating limits should a fault occur.

This thesis provides the following contributions. 1. Models describing the dynamics of slow machinery attached to and coupled via the electric power network were used to allow dynamic state estimation. 2. The detail of the coupled dynamic network model was evaluated to determine the level of modeling complexity required to achieve significant state estimation performance gains. 3. Improvements to bad data detection and identification by using information from the dynamic state estimator were demonstrated and evaluated. 4. The improvements to network static observability were discussed and evaluated.

Acknowledgments

I would like to thank my advisors, Professor Bruce Krogh and Professor Marija Ilić. Without them I would never have completed this work. It has been my privilege and honor to work with individuals with such thoughtful guidance, in-exhaustive patience, technical expertise, and deep insight. I've lost count of the number of times that I've been steered away from potential dead ends and towards truly productive paths.

I would also like to thank my wife, Alexa Broughton, whose support and patience kept me on track during my period of All-But-Dissertation status where I was teaching full time, raising a new baby, and suffering myriad other distractions.

I gratefully acknowledge the funding sources that made my Ph.D. work possible. I was supported by the The National Science Foundation Information Technology Research Program (CCR-0325892, SES-034578, SNC-042804) and the U.S. Department of Energy, National Energy Technology Laboratory (DE-AM26-04NT41817.305.01.21.002).

I would like to thank Claire Bauerle and Elaine Lawrence for being the people who always had the answers I needed regarding how to get things done at CMU.

Thanks to my USNA department associate chair, CAPT Jack Nicholson who allowed me a "lighter" load of collateral duties so that I could have time to better focus on my dissertation when my duties as a USNA instructor allowed. Also thanks to Laura Myers and Judy Raymond who helped me locate and utelize the necessary computing resources at the Naval Academy so that I could gather the needed simulation data.

I am grateful to my fellow group of graduate students who were always available to discuss ideas with and be generally good people to be with, especially Marija Prića, Xie Le, Yi Zhang, Juhua Liu, Jim Weimer, Hwaiwei Lau.

Special thanks to Jovan Ilić, whose technical expertise was invaluable in the formulation of my solution method and development of my computational simulations.

Thank you,

Ellery Blood

List of Tables

List of Figures

Contents

Chapter 1

Introduction

Accurate real-time estimates of bus voltages are essential for successful operation of electric power systems. These estimates make it possible to calculate power flows along transmission lines, proximity to operating limits, the operating points of electric loads and generators, and other critical information used by transmission network operators [4]. Commonly referred to as the state of the power system, we will call the bus voltages the *network state* in this dissertation, to distinguish this set of variables from the other variables that characterize the dynamic states of the many devices and components that comprise the complete power system.

In today's power systems, the network state is estimated at regular intervals using only the most recent set of measurements that give a "snapshot" of the network operating point [42]. Estimation of the network state is a nontrivial problem due to the large number of measurements and network state variables, the nonlinear network equations, the presence of measurement noise and the common occurrence of bad data due to intermittent sensor and communication failures. Bad data is so significant that a preprocessor is typically employed to identify and remove grossly inaccurate measurements from the measurement snapshot before the estimation process even begins. Removing these measurements often causes some components of the network state to be unobservable from the set of good data points. Thus, estimates for possibly large portions of the network cannot be updated at times. Moreover, since existing methods which compute estimates of the network state variables for the observable portion of the network use only the current set of measurements, these estimates do not benefit from the information available from past measurements, which are clearly correlated with the current set of measurements [11].

This dissertation develops a new method for on-line network state estimation that leverages the infor-

mation in the time history of power system measurements to improve estimation robustness and accuracy, including estimates of the network state variables that are unobservable using traditional methods. Methods that use previous measurements in addition to the current measurements are referred to as *dynamic network state estimation* since they estimate the network state variables using models that include dynamic state variables for the network and its components. In contrast to previously proposed methods for dynamic network state estimation, which are based on linear regression [29, 45] or a zero order hold [11], the approach proposed in this dissertation uses governor-turbine-generator and inertial load models to effectively track the behaviors of the components attached to the network rather than look at only the network itself. The proposed method is developed using the lossless, decoupled formulation for real power flows in the network [1, 41].

1.1 Contributions

The dissertation makes the following contributions:

1) The inclusion of dynamic models of the power system components at each bus, including conventional models of generation and new dynamic load models, as the basis for dynamic network state estimation. This is in contrast to previously proposed approaches to dynamic network state estimation that are based on ad hoc dynamic models rather than explicit models of the components at each bus. In this new dynamic model, the network state variables are output variables defined by the physical dynamic state variables for the complete system.

2) A systematic method for reducing the dynamic component models based on time-scale analysis of the local bus models and the strengths of couplings in the network admittance matrix. This model order reduction reduces the on-line computations for dynamic state estimation.

3) The ability to update estimates of network state variables that are statically unobservable due to intermittent bad data, with bounds on the estimation error based on the information matrix formulation of the Kalman filter [22].

4) A new method for using the dynamic model to identify bad data using predicted states that is significantly more accurate than current methods for bad data identification based on single snap shots.

5) A demonstration and evaluation of the effectiveness and robustness of these innovations using standard IEEE test systems. Comparisons are made to static estimation techniques presently used in industry and dynamic estimation techniques proposed in literature.

The new techniques under development in this research are implemented in MATLAB and evaluated using

Monte Carlo simulations. Model and algorithm verification was aided through the use of the MIPSYS analysis environment developed at CMU for static electric network analysis [18]. We show that the methods proposed herein perform comparably to existing methods in normal operating conditions, and perform considerably better in conditions where bad data causes a reduction in the number of measurements available to the state estimator.

Chapter 2

Background

Estimation of the network state has been heavily used in industry since the 1960's. This chapter provides background information on the existing methods of network state estimation, the measurement models used in that estimation, the dynamics of components attached to the electrical network, and considerations taken into account to improve the numerical process of estimating the network state.

2.1 Dynamics of Transmission Lines

The electric power transmission grid is primarily composed of a network of three-phase alternating current carrying transmission lines. An individual transmission line is typically modeled using a single phase π circuit [15], shown in Fig. 2.1. Due to the capacitance, inductance, and resistances of the lines, the network is a dynamic system with time constants on the order of 0.01 seconds or faster [38]. Transmitting and processing measurements sufficiently fast to capture these transients has been impractical for most of the history of the United States power grid. Therefore, the standard practice in today's industry is for operators to work on an assumption of steady state operations and to use a weighted least squares approach based on a static, or memoryless network model [1, 30] to estimate the state of the network.

This model has served the electric power industry reasonably well as it makes use of the strengths of the reporting frequency of the existing telemetry infrastructure. The supervisory control and data acquisition (SCADA) system used by the electric power industry transmits measurement data to the control centers roughly every two seconds [23]. Compared with the electrical transient time constants, this is more than an order of magnitude slower than that necessary to accurately capture transient information. If a transient

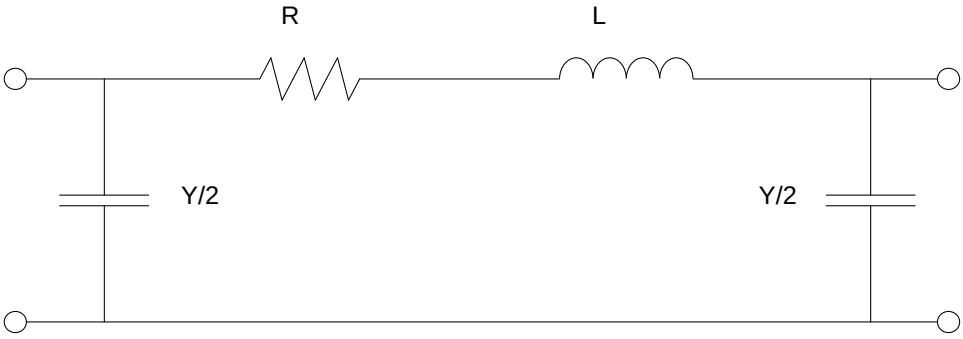


Figure 2.1: Standard transmission line π model

were to occur immediately following a measurement, the effect of that transient would have decayed to $e^{-20} \approx 2 \times 10^{-9}$ of its original value by the time the next measurement was taken, two seconds later. Without additional information, accurate dynamic state estimation is impractical.

2.2 Measurement Model

The measurements typically used in power system network state estimation are: real and reactive power injections at a bus, real and reactive power flows injected into a transmission line, current flows injected into a transmission line, and voltage magnitudes. For simplicity, this dissertation primarily focuses on real power injections and real power flows on a transmission grid composed of lossless lines (i.e., purely reactive lines so that $g_{ij} = 0$). In addition, the complex phasor voltage $\tilde{V} = V\cos(\delta) + iV\sin(\delta)$ is assumed to have a voltage magnitude which is constant at 1 p.u. and the voltage angle differences between adjacent busses are assumed to be small. These assumptions are necessary to facilitate accurate decoupling between voltage magnitude and angle and between real and reactive power, using the small angle approximation, $\cos(\delta) \approx 1$. A linear decoupled model can also be derived by further using the small angle approximation to assume $\sin(\delta) \approx \delta$. This decoupling allows the real power flows and injections to be treated as functions of the voltage angle, δ only [1]. The equations associated with these simplifications are listed in table 2.1 and describe the relationship between the network state and the measurements, where P_i is the real power injection into the network at bus i, P_{ij} is the real power flow along the line(s) connecting busses i and j, V_i is the voltage magnitude at bus i, g_{ij} is the line conductance between busses i and j, and b_{ij} is the line susceptance between busses i and j.

These assumptions are reasonable under the following conditions [1, 15].

1. Lossless Lines: The purely reactive lines are a reasonable assumption when the susceptance (imaginary

Measurement	Full Model	Decoupled Model	Linear Decoupled Model
Power Injection: P_i	$V_i \sum_{j \in N_{L_i}} V_j (g_{ij} \cos(\delta_{ij}) + b_{ij} \sin(\delta_{ij}))$	$\sum_{j \in N_{L_i}} b_{ij} \sin(\delta_{ij})$	$\sum_{j \in N_{L_i}} b_{ij} \delta_{ij}$
Power Flow: P_{ij}	$V_i V_j (g_{ij} \cos(\delta_{ij}) + b_{ij} \sin(\delta_{ij}))$	$b_{ij} \sin(\delta_{ij})$	$b_{ij} \delta_{ij}$

Table 2.1: Measurement model for power injections and flows

part of the admittance) is more than ten times the magnitude of the conductance (real part of the admittance).

2. $sin(\theta) = \theta$: The small angle *sin* approximation is reasonable when the differential bus angles across transmission lines are less than 10 degrees. For example, an angle of 14 degrees will introduce a 1% error in the *sin* calculation.

3. $cos(\theta) = 1$: The small angle *cos* approximation is reasonable when the differential bus angles across transmission lines are less than 10 degrees. For example, an angle of 8 degrees will introduce a 1% error in the *cos* calculation.

4. $|V| = 1$p.u. : This assumption is reasonable as long as the voltage magnitudes remain within approximately 1% of design values.

2.3 Weighted Least Squares Minimization

Calculation of the network state from the available measurements can be accomplished multiple ways. Iterative methods are typically employed and various simplifications of the measurement model may prove advantageous for certain algorithms under certain network conditions[24]. Table 2.2 lists a few measurement models typically used for network state estimation [1, 13, 47].

Model	Measurements	Coupling	Network State
1	P,Q	Coupled	δ,$\|V\|$
2	P,Q	Decoupled	δ,$\|V\|$
3	P,Q	Coupled	V_{real},V_{imag}
4	P	Decoupled	δ

Table 2.2: Simplified measurement models used for network state estimation

Each method has its own advantages. Item one converges in the fewest number of iterations, but requires the most computation for each iteration. Item two requires one or two more iterations to converge to the same tolerances, but requires approximately an order of magnitude less computation. Item three has faster convergence but tends to exhibit a bias to its final solution. Item four converges approximately two orders of magnitude faster than item one, but has the least accuracy due to its simplified model. As discussed in

Sec. 2.2, we will focus on model four, the decoupled δ method where only the voltage angle, δ, as a function of the real power, P, is estimated.

To calculate the state of a memoryless system where the measurements are corrupted with zero-mean additive-white-Gaussian-noise (AWGN), the standard technique is to use a weighted least squares [1]. The weighted least squares is derived from the maximum likelihood estimation problem [39].

Using a memoryless system model, the static estimation problem is stated $\hat{\delta}_{(k)} = \mathrm{E}\{\delta_{(k)}|\tilde{\mathbf{z}}_{(k)}\}$, where $\tilde{\mathbf{z}}_{(k)} = h(\delta_{(k)}) + v_{(k)}$ and $v_{(k)}$ is AWGN and $h(\delta)$ is the vector of measurement functions described in Sec. 2.2 . The probability distribution function of $\tilde{z}_k$ is

$$\frac{1}{(2\pi)^{n/2}|\mathbf{V}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{z}_{(k)} - \mathbf{h}(\delta_{(k)}))^T \mathbf{V}^{-1}(\mathbf{z}_{(k)} - \mathbf{h}(\delta_{(k)})\right),$$

where $\mathbf{V}$ is the covariance of the measurement noise and n is the number of measurements [33]. This function has its maximum value where

$$J(\delta_{(k)}) = (\tilde{\mathbf{z}}_{(k)} - \mathbf{h}(\delta_{(k)}))^T \mathbf{V}^{-1}(\tilde{\mathbf{z}}_{(k)} - \mathbf{h}(\delta_{(k)})$$

is minimized [33]. The maximum likelihood estimate is therefore $\hat{\delta}_{(k)} = \min_\delta J(\delta_{(k)})$

If $h(\delta)$ is approximated as $h(\delta) = h(\delta_0) + H(\delta - \delta_0)$ where $H = \left.\frac{\partial h(\delta)}{\partial \delta}\right|_{\delta=\delta_0}$, the minimum value can be found through the following iteration:

$$\begin{aligned} \text{set:} \quad & \hat{\delta}^{(0)} = 0 \\ \text{repeat:} \quad & \hat{\delta}^{(i+1)} = (H^T V^{-1} H)^{-1} H^T V^{-1}(z - h(\hat{\delta}^{(i)})) \\ \text{until:} \quad & \hat{\delta}^{(i)} - \hat{\delta}^{(i-1)} \leq \epsilon \end{aligned} \tag{2.1}$$

Here we have temporarily dropped the time index $\bullet_{(k)}$ for clarity. The index $\bullet^{(i)}$ indicates the iteration. Using this method, the expected value of the state error is

$$\mathrm{E}\{\delta - \hat{\delta}\} = 0$$

and the state error variance

$$\mathrm{E}\{(\delta - \hat{\delta})(\delta - \hat{\delta})^T\} \quad = \quad (H^T V^{-1} H)^{-1}$$

$$= \Psi. \tag{2.2}$$

One metric of how well a state estimator is performing is the trace of the state error covariance matrix [5]. This metric is discussed in Sec. 3.8.2.

Now that we have defined Ψ to be the network state error covariance matrix, we can rewrite (2.1) as

$$\hat{\delta}^{(i+1)} = \Psi H^T V^{-1}(z - h(\hat{\delta}^{(i)}))$$

2.4 Bad Data

One of the challenges in power system network state estimation is the existence of spurious events causing a subset of the measurements to be grossly inaccurate [4]. These bad data have values that are inconsistent with the measurement model used to relate the measurements and the state in the network state estimation process. Typically a measurement which is corrupted with noise of a magnitude greater than three standard deviations of the expected noise is considered bad data. These bad data must be removed if an accurate estimate is to be achieved. The removal comes in two parts: detecting the existence of bad data in the measurement vector, and identifying the bad elements of that measurement vector.

2.4.1 Bad Data Detection and Identification

Detecting the presence of bad data is typically accomplished through the use of a Chi-square test [1]. The measurements are assumed to be a function of the true network state and corrupted by additive white Gaussian noise (AWGN). The distribution of the sum of squared errors between the measurements and the measurement estimates, as calculated from the state estimate, should therefore conform to a Chi-square distribution [49].

A significance level is chosen and applied to the Chi-square distribution to determine a threshold, η, for the Chi-square hypothesis test. If the weighted sum of squared errors is less than η, the hypothesis that the sum is consistent with the expected distribution given by the assumed AWGN in the model is chosen. If the sum is greater than η, the null hypothesis is rejected and bad data is assumed to exist in the measurement vector.

The identification of the number and location of the bad data is more challenging. For a linear measurement model, flagging the measurement with the largest weighted residual as bad will typically prove accurate [1]. Once identified, the suspect measurement is removed from the measurement vector and the

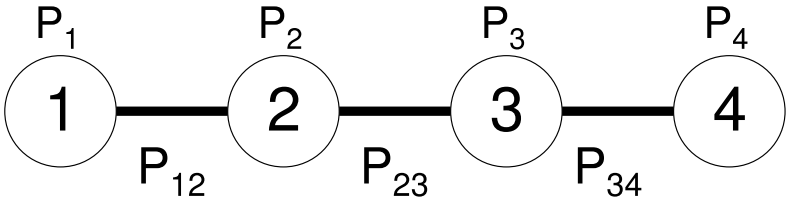


Figure 2.2: Sample network for smearing example

estimation-detection-identification process is repeated. This iteration continues until the Chi-square hypothesis performed during the detection step returns the null hypothesis, i.e., that the sum conforms to that given by AWGN within a given significance level and additional instances of bad data are unlikely.

Bad data detection and identification are further discussed in Sec. 5.2.

2.4.2 Smearing

Each measurement is a direct function of multiple elements of the network state vector and each element of the network state vector estimate is calculated from information from multiple measurements. The coupling between the states and measurements lead to an effect known as *smearing* [41]. A large error in a single measurement will lead to errors in all the network state variables associated with that measurement. All the measurements that are associated with those state variables (i.e., measurements that are strongly correlated with, or "near" the original bad measurement) will then be affected.

For example, in the following sample network (see Fig. 2.2), there are seven measurements consisting of real power injections (P_1, P_2, P_3, and P_4) and real power flows (P_{12}, P_{23}, P_{34}) providing information about the four states (δ_1, δ_2, δ_2, and δ_4).

The individual measurements affected by smearing due to one bad datum can be seen by analyzing the relationship between measurements and elements of the network state vector as a bipartite graph (see Fig. 2.3). The states are on the left side of the graph; the measurements are on the right. Connecting links indicate the elements of the state contributing to an individual measurements and vice versa.

The estimate of the network state is calculated through the a minimum-mean-squared-error optimization and therefore requires the weights to be equal to the variance of the AWGN affecting the measurements. If the noise affecting the measurements does not conform to the assumed distribution, the estimate will not be optimal. An individual measurement whose value is grossly inaccurate (typically more than $10\sigma_k$ from the true value [41]) will skew the estimate away from the true value.

The determination of which network state estimates, $\hat{\delta}_k$, are affected by a bad measurement is simply a

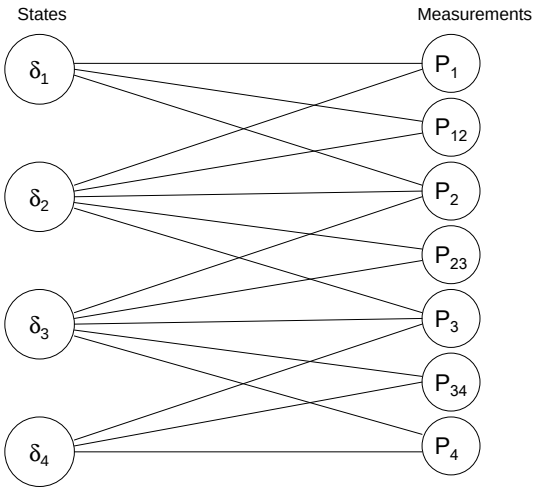


Figure 2.3: Bipartite graph for smearing example

traversal from that measurement node to the state nodes on the other side. In Fig. 2.4, bad data is present on the real power injection to bus one, P_1. Traversing the graph from right to left, we see that the elements of the state δ_1 and δ_2 will be affected by this bad measurements.

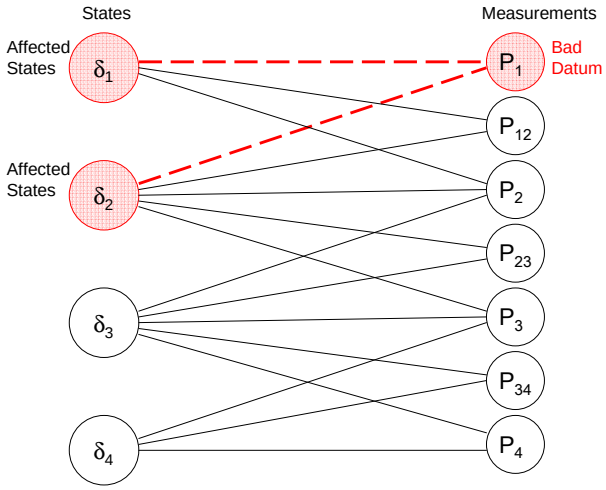


Figure 2.4: Smearing bipartite graph with bad data

Since one of the primary purposes of the state estimator is to provide estimates of the measured variables $\mathbf{h}_k(\hat{\delta})$, determining which measurement estimates are corrupted is important. The list of measurement estimates subject to smearing can likewise be determined by traversing the graph from the affected elements of the state back to the measurements. In Fig. 2.5 we follow the lines from affected states δ_1 and δ_2 to the measurements P_1, P_{12}, P_2, P_{23}, and P_3.

Numerically, this traversal can be interpreted as a multiplication by the adjacency matrix of the bipartite

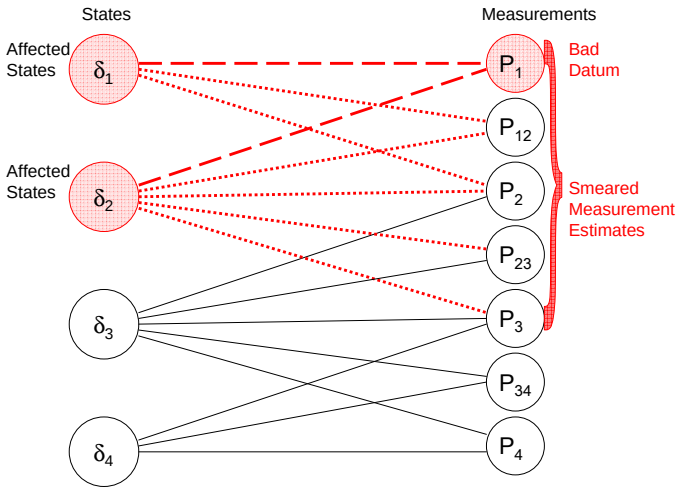


Figure 2.5: Smearing bipartite graph with affected measurements

graph. Define the nodes of the graph to be the vector $[\mathbf{z}^T, \delta^T]^T$. The adjacency matrix is then

$$\begin{bmatrix} \mathbf{0} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{0} \end{bmatrix},$$

where $\mathbf{B}$ is constructed by replacing the nonzero elements of the measurement Jacobian $\mathbf{H}$ with ones. The traversal from $\mathbf{z}$ to $\hat{\delta}$ and back to $\mathbf{h}(\hat{\delta})$ is accomplished by multiplying the adjacency matrix by a vector with a one in the location of the bad data and zeros otherwise to get the elements of $\hat{\delta}$ that are smeared. This result is then multiplied by the adjacency matrix again to determine which elements of $\mathbf{h}(\hat{\delta})$ which are smeared.

To understand the implication of this smearing effect, it is useful to recognize what the network state estimation process is doing in terms of subspace projections. One can view minimum mean squared error estimation as the projection of $\mathbf{z}$ onto the subspace $\mathbf{G} = \mathbf{V}^{-1/2}\mathbf{H}$ with an additional weighting term of $\mathbf{V}^{-1/2}$. The projection operator for $\mathbf{G}$,

$$\begin{aligned} P_{\mathbf{G}} &= \mathbf{G}(\mathbf{G}^T\mathbf{G})^{-1}\mathbf{G}^T \\ &= \mathbf{V}^{-1/2}\mathbf{H}(\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{V}^{-1/2} \\ &= \mathbf{V}^{-1/2}\Big(\mathbf{H}(\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{V}^{-1}\Big)\mathbf{V}^{1/2} \\ &= \mathbf{V}^{-1/2}\text{pinv}(\mathbf{H},\mathbf{V})\mathbf{V}^{1/2}, \end{aligned}$$

simplifies to just the weighted pseudoinverse and the square root of the measurement noise variance matrix. When the measurement noise variance matrix is diagonal, the weighting matrices cancel out and the projection operator further simplifies to the weighted pseudoinverse, pinv($\mathbf{H}, \mathbf{V}$).

Returning to the bipartite graph and the adjacency matrix described above, the nonzero elements of $\mathbf{P}_G$ are the same as those of $\mathbf{B}^T\mathbf{B}$. Therefore, the k^{th} column of the projection matrix reveals the elements of the $\mathbf{h}(\delta)$ vector smeared by bad data in $\mathbf{z}_k$. The projection provides additional information in that the magnitude of the smearing effect on each of the calculated measurements, $h(\hat{\delta})$, is revealed.

The smearing effect from any individual bad measurement does not extend beyond the calculated measurements, as indicated by the nonzero elements of the product computed above. This can be seen by recognizing that the product is already projected upon the subspace $\mathbf{H}$ so that projecting the new vector onto the subspace will only return that same vector again.

A few observations can be drawn from the above discussion:

1. As the connectivity of each node (number of other nodes it is connected to and the strength of the connecting transmission lines) increases, the number of other measurements affected by the smearing increases..
2. As the connectivity of each node, the magnitude of the smearing effect decreases. An increased number of measurements will tend to dilute the effect of the bad measurement.
3. As the expected variance of the measurement in question increases, the effect of the bad data will be minimized due to the weighting in the pseudoinverse. Conversely, bad data affecting a measurement which normally is highly accurate will tend to have a severe smearing impact on adjacent measurements.

In order to mitigate the effect of smearing, this thesis proposes using the predicted dynamic state to perform an initial sweep for bad data before the static network state estimator processes the measurements. As the predicted dynamic state is uncorrelated with the incoming measurements, they are not affected by smearing.

2.4.3 Bad Data and Static Observability

If sufficient redundancy in the measurement vector exists, the network state estimator will be able to successfully estimate the network state despite the removal of the bad data from the measurement vector [50]. This redundancy, however, cannot be guaranteed. Some elements of the network state may not be statically observable from the reduced measurement vector. Using the techniques described above to estimate the network state would return values for the unobservable network state elements that are not meaningful and

should not be used.

If sufficient redundancy is not present or is reduced due to removal of bad data, elements of the measurement vector may become *critical*, which means that a critical measurement's associated state is uniquely determined from that measurement. The estimate residual for this measurement will therefore be zero regardless of the measurement error. The practical implication is that a critical measurement cannot be tested for bad data [2].

In a linear measurement model, the network state is statically observable if the rank of the measurement Jacobian is equal to the number of elements in the network state vector, i.e., $\text{rank}(J(\delta)) = \text{rank}(\frac{\partial \mathbf{h}(\delta)}{\partial \delta}) = n_\delta$ [5]. This is a reasonable approximation for the nonlinear measurement model used in electrical power network state estimation [1] relating bus angle and real power flow. The accuracy of this approximation tends to degrade as the system loading increases, causing an increase in the relative bus angles.

The condition of a statically unobservable network state is handled by analyzing the measurements to determine which subset of the network state vector is statically observable [26]. The network state estimator then isolates and only attempts to estimate the network state of this statically observable subnetwork. Due to the static nature of the process, this results in a condition where no meaningful value of the network state is available in the unobservable subnetwork. The unobservable subnetwork may be composed of one or more unobservable islands [1, 30]. The network state estimator must therefore identify and flag those unobserved elements so that they will not be erroneously relied upon for operation.

2.5 Estimation Based on Multiple Scans of Measurements

Although the electrical time constants of the transmission network components are much faster than the typical sampling rate for power system network state estimators, a distinct correlation between scans is evident. This correlation can be useful for various purposes including network parameter estimation [30, 46], online measurement calibration [3, 44], and measurement error estimation [43].

In its most basic form, even the fully static network state estimator uses information from multiple scans. The static estimator must have an initial start value for its iterative minimization procedure. When the estimator receives its very first snapshot, it typically uses a flat start (voltage magnitude equal to 1 p.u. and all bus angles equal to zero). On each successive snapshot received, the estimator initializes the iterative minimization to the previous state estimate. This starts the iteration off on a known good value and therefore typically requires fewer iterations to reach convergence.

The literature also shows many proposals by which incorporating information from multiple scans may

offer improvements in network state estimation despite the disparity between sampling rate and network dynamic time constants. These methods take the standard dynamic model of $\mathbf{x}(k+1) = f(\mathbf{x}(k), \mathbf{u}(k)) + \mathbf{w}(k)$ and apply various assumptions to arrive at an approximate dynamic model suitable for the purposes of predicting the network state. The predicted network state allows for a Bayesian formulation of the network state estimate as opposed to a maximum likelihood formulation.

2.5.1 Tracking Network State Estimation

The tracking state estimator typically employs a linearized dynamic model expressed as

$$\mathbf{x}(k+1) = (F)\mathbf{x}(k) + \mathbf{G}\mathbf{x}(k) + \mathbf{w}(k), \tag{2.3}$$

where (F) is identity and $\mathbf{G}$ is zero, so that the network state dynamics are driven by the random noise $\mathbf{w}(\mathbf{k})$ only [11, 29]. This formulation works well as long as the incremental change in state remains below an assumed maximum ramp rate characterized by the magnitude of the noise.

Although the dynamic model is simple, the major contribution of this formulation is to provide an *a priori* estimate of the state, so that the state estimator needs only to update the estimate with new information rather than start the process from scratch. In addition, unlike a static network state estimator, the tracking network state estimator need not wait for a snapshot of the full measurement vector to be available to begin its update process. Due to the *a priori* state information, the new measurements can be processed sequentially; each new measurement providing an incremental update to the existing state estimate. These incremental updates allow the network state vector to be continuously updated with each new piece of information allowing faster feedback to operators and a reduced level of computation for each update [21].

2.5.2 Dynamic Network State Estimation

Dynamic network state estimation is similar to static network state estimation except that the network state can be predicted from previous network state values so that an *a priori* value for the network state is available when the measurements are incorporated. In other words, static network state estimation is an example of maximum likelihood estimation whereas dynamic network state estimation is an example of Bayesian estimation. It can be shown that the static estimation step is numerically equivalent to the update step for a dynamic estimator where the *a priori* network state estimate is included as additional measurements for the static estimator [5].

Despite the quasi static nature of network state estimation, numerous techniques have been proposed to define pseudo dynamic models to provide the prediction of the network state for Kalman filtering of the network state. Several of these techniques leverage the strong correlation between adjacent snapshots or used time series analysis of the sequence of static network state to define a pseudo-dynamic models [28, 11, 29, 30, 37, 45]. Simulations of estimators using both of these modeling philosophies demonstrate the potential for improved performance over static estimation [28, 11, 29, 45]

In recent years, increases in computation capability of data processing centers and installation of advanced metering equipment such as phaser measurement units (PMUs) has started to enable very accurate load forecasting of power systems [51, 21]. These forecasts aid in the modeling of system dynamics as the dynamic model's **G** matrix from (2.3) can represent the relationship between the load and the network state, providing improved modeling of the incremental changes in the network state between snapshots [7, 8, 51].

2.5.3 Limitations of Dynamic Network State Estimation

Dynamic network state estimation is only useful when *a priori* information contributes to the present state. For example, when a reconfiguration of the network occurs, the relationships between the voltage angles may undergo a drastic change. In this situation, the previous network state would have little relation to the present one. The typical procedure in these situations is to throw out the *a priori* information from the dynamic prediction and reinitialize the dynamic estimator from the static estimator solution.

In order for dynamic network state estimation methods to be beneficial in the event of topology changes, the dynamic model must be able to predict conditions of the network which are primarily unaffected by such transients. One such method is described in Ch. 3, where the components attached to the buses are modeled to provide continuity between static network state estimates. But even this technique has limitations. Some buses experience occasional discontinuities in load such as large step changes as would be seen with the operation of arc furnaces. Dynamic models are typically unable to predict such load behavior so that once again the *a priori* state information must be discarded.

2.6 Computation

In order to perform the state estimation process in near real time on any network of reasonable size (greater than 100 busses) requires significant computer processing power [50]. Power system state estimators have typically not been operated in real time. As computer processing power has increased, the size of the network

being estimated and the complexity of the estimation algorithms have also increased keeping pace with the processing power so that the rate of network state update increases only marginally [50].

The control center for a typical electrical transmission network in the United States processes 20,000 to 40,000 measurements and provides state estimation data on over 10,000 busses. For example PJM Interconnection, a regional transmission organization (RTO) covering 168,500 square miles of 12 different states, monitors approximately 13,500 buses [34]. Similarly, the Electric Reliability Council of Texas (ERCOT) monitors approximately 18,000 busses [12]. To perform these calculations, the state estimator must compute the solution to a set of equations containing 20,000 variables and matrices with sizes on the order of 20,000 by 40,000. To aid in this process, several standard computational techniques are typically employed and are described in the following subsections.

2.6.1 Matrix Manipulation

Much of the calculation work done by the state estimators can be categorized under the heading of simultaneous equation solvers. The general form is $\mathbf{y} = \mathbf{f}(\mathbf{x})$ or $\mathbf{y} = \mathbf{M}\mathbf{x}$ in the linear case. A naive solution to this problem would be to take the inverse (or pseudo inverse in the case where the length of the $\mathbf{y}$ and $\mathbf{x}$ vectors are unequal) of the $\mathbf{M}$ matrix. In reality, some form of matrix factorization and back-substitution is employed.

For systems of n equations and n unknowns, Gaussian elimination with back-substitution is typically employed. Gaussian elimination, however, is numerically unstable for large sets of equations. Therefore, alternative methods involving partial pivoting and matrix factoring are employed. Matrix factoring in this case is typically accomplished through LU decomposition, however Cholesky factorization may be employed for symmetric positive definite matrices for improved speed of computation.

For systems of n of r unknowns where $r < n$ and a least squares solution is desired, QR decomposition can be employed to assist in the calculations. In QR the original matrix is factored into an orthogonal matrix Q and an right (upper) triangular matrix R. The Q matrix may be inverted by $Q^{-1} = Q^T$, and the R matrix can easily be back substituted through.

Both these methods improve computation robustness and speed as long as $\mathbf{M}$ remains mostly unchanged so that re factorization is infrequent. The factored matrices can be cached for future use.

2.6.2 Sparse Matrix Methods

There are two distinct, and not always compatible, goals for any sparse matrix method: saving time and/or saving space [35]. The emphasis in this discussion will be on using sparse methods optimized for speed in order to enable real time processing of measurements for state estimation.

Applying a Gaussian elimination algorithm to a sparse matrix without accounting for the sparsity pattern can result in an intermediate matrix that is much less sparse than the original. Some decrease in sparsity is typically unavoidable, however a Gaussian elimination algorithm may exploit a specific pattern of sparsity to minimize the decrease in sparsity. If a given matrix does not exactly conform to an expected pattern of sparsity, pivoting can be employed to adjust a matrix's sparsity pattern for use with a specific optimized algorithm [35].

One hazard of applying pivoting with the goal of adjusting the sparsity pattern is that the algorithm is limited in its ability to apply pivoting for numerical stability. Luckily, the positive definiteness of the admittance (Y) matrix (heavily used in power system analysis) means that numerical stability is typically not negatively impacted by optimizing the pivoting for sparse methods [36].

Optimization of Gaussian elimination operations is typically achieved through one of various pre-factoring methods such as LU, Cholesky, or QR as discussed in Sec. 2.6.1. In order to preserve the maximum sparsity, the factoring operations typically employ successive applications of givens rotations rather than the more traditional householder reflections (Grahm-Schmidt is typically unsuitable for sparse matrices)[36]. Factorization using Givens rotations typically requires an approximate 50% increase in computation over Householder reflections for a full matrix; however, for sparse matrices isolating individual elements is more direct [10].

By employing these sparse methods, matrix operations can be sped up by several orders of magnitude. For a general sparse matrix, the processing time can potentially be reduced to a function of the number of nonzero elements, rather than the size. For example, the multiplication of an $n \times n$ matrix by a n vector would require n^2 multiplications and $(n-1)n$ additions. Conversely, if the sparse matrix has on average m nonzero elements per row, there would be nm multiplications and $n(m-1)$ additions. If the sparsity factor is 100 (i.e., $m = n/100$), these algorithms would realize nearly a 100 times speedup (the actual speedup will be less than 100 due to the increased overhead of the sparse routines). Operations on sparse matrices exhibiting specific patterns (e.g., diagonal, tri-diagonal, block-diagonal) can be computed on the order of n, and therefore can realize even higher levels of speedup [35].

Many of the calculations described in this thesis can be parallelized for distributed and parallel com-

putation. Computer graphics hardware can be particularly effective at tackling these types of problems. While designed for graphical transformations and projections, the single instruction multiple data (SIMD) architecture has been shown to offer further speed and efficiency improvements [25, 9, 16].

2.6.3 Kalman Filtering of Power Systems

The standard formulation for the Kalman filter is typically employed in dynamic systems where the number of states (n_x) is larger than the number of measurements (n_z). This formulation is computationally useful as it only requires the inverse of the innovation covariance matrix which is of size $n_z \times n_z$. In electric power systems, n_z is typically at least two times n_x, making the standard formulation less efficient.

The information filter is an implementation of the Kalman filter equations where the information matrix $\mathbf{Y} = \mathbf{P}^{-1}$ is used instead of the state error covariance matrix $\mathbf{P}$, where

$$\mathbf{P} = E[(\hat{\mathbf{x}} - \mathbf{x})(\hat{\mathbf{x}} - \mathbf{x})^T].$$

The formulation for the information filter does not require the inversion of the innovation covariance [14]. Instead, the state error covariance matrix is inverted to get the information matrix. This is useful as the dimensions of $\mathbf{P}$ being $n_\delta \times n_\delta$ are typically half that of the dimension of the innovation covariance matrix $\mathbf{Y}$ being of size $n_x \times n_x$.

The actual speedup can be seen by noting that matrix inversion typically requires computations on the order of $O(n^3)$. As $n_z \approx 2n_x$, $O(n_z^3) \approx 8O(n_x^3)$ so nearly an order of magnitude improvement in processing time can be achieved by using the information filter formulation.

Chapter 3

Modeling for Dynamic Network State Estimation

The pseudo-dynamic models in the tracking network state estimation algorithms rely on the correlation of the network state between snapshots but do not explain why this correlation exists. Looking at the power system as a whole, it is evident that the system contains many components attached to the busses of the network. Many of these components are large rotating machines that have time constants on the order of five to ten seconds. Other components are small switched loads which, when taken as a aggregate, appear as a large load affected by small stochastic perturbations. Both these types of components contribute to the inertia necessary for the tracking network state estimation algorithms to operate effectively. Standard models for these bus components exist and can be incorporated into a model-based dynamic network state estimator for the electric power system.

To illustrate these concepts, a simple three-bus example system (Fig. 3.1) will be used in this chapter. Busses one and two are generator busses; bus three is a load bus. The black dots indicate location of measurements (in this case, the real power injected at each bus). The admittances are given in per-unit.

3.1 Modeling the Component Dynamics

At each bus in the network, the dynamics of the system at that particular bus are modeled. We represent the dynamic state vector at bus i as $\mathbf{x}_i$. If the components at a bus are primarily composed of load components, the dynamic system takes as inputs ($\mathbf{u}$) the external load variations and the power injection from the network.

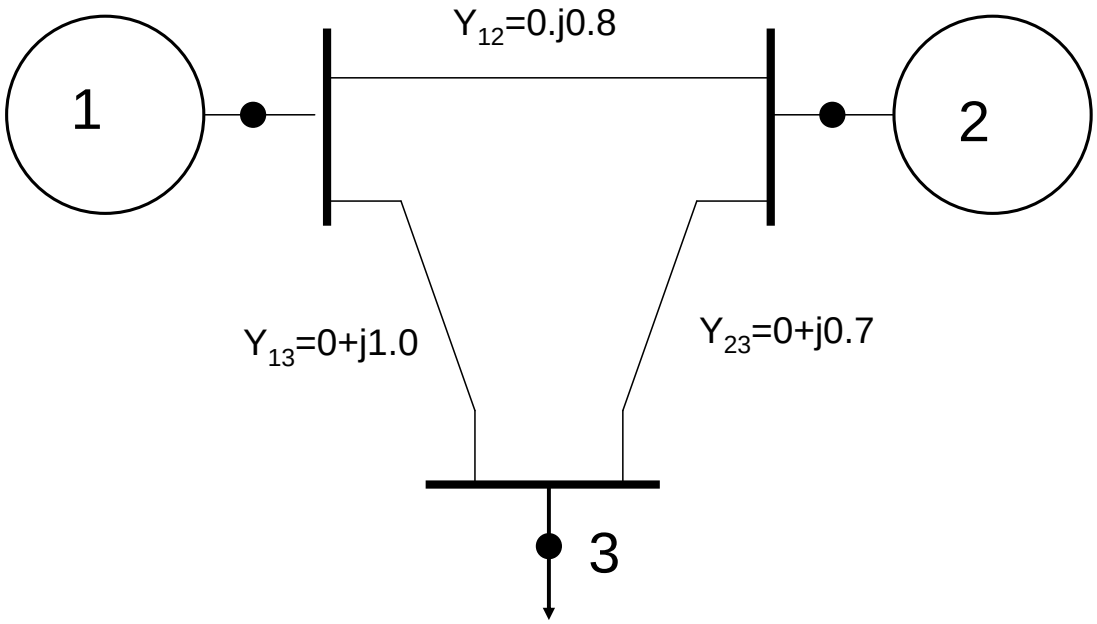


Figure 3.1: Three Bus Example

If the components are primarily composed of generation, they take as input the external generator setpoints and the power injection from the network.

For brevity, we will use the word *component* to refer to the dynamic system located at a bus. The following is the dynamic model for a component system primarily consisting of generation [27]:

$$\frac{d}{dt}\begin{bmatrix} \Delta a \\ \Delta\omega_r \\ \Delta P_m \\ \Delta\delta \end{bmatrix} = \begin{bmatrix} -kR & k & 0 & 0 \\ 0 & -D/M & 1/M & 0 \\ 1/T_{CH} & 0 & -1/T_{CH} & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta a \\ \Delta\omega_r \\ \Delta P_m \\ \Delta\delta \end{bmatrix} + \begin{bmatrix} -k & -k \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta\omega_0 \\ L_{ref} \end{bmatrix} + \begin{bmatrix} 0 \\ -1/M \\ 0 \\ 0 \end{bmatrix} [\Delta P_E], \tag{3.1}$$

where Δa is the differential prime mover valve position, $\Delta\omega_r$ is the differential generator shaft frequency, ΔP_m is the differential mechanical power, and $\Delta\delta$ is the divergence of the generator absolute shaft position from nominal. The parameters are: k, governor feedback gain; R, the droop characteristic; D, generator rotor damping characteristic; M, generator rotational inertia; and T_{CH}, prime mover flow inertial time constant. The inputs are: ΔP_L, exogenous differential load value; $\Delta\omega_0$, frequency differential setpoint; and L_{ref}, exogenous load adjustment setpoint (e.g., AGC setpoint).

Similarly, an aggregate load containing rotating machinery can be represented as:

$$\frac{d}{dt}\begin{bmatrix}\Delta\omega_r\\ \Delta P_L\\ \Delta\delta\end{bmatrix} = \begin{bmatrix}-D/M & -1/M & 0\\ 0 & 0 & 0\\ 1 & 0 & 0\end{bmatrix}\begin{bmatrix}\Delta\omega_r\\ \Delta P_L\\ \Delta\delta\end{bmatrix} + \begin{bmatrix}0\\ 1\\ 0\end{bmatrix}\left[P_L^{\text{rate}}\right] + \begin{bmatrix}1/M\\ 0\\ 0\end{bmatrix}\left[\Delta P_E\right], \qquad (3.2)$$

where P_L^{rate} is the rate of change of the load (modeled as a stochastic input) [11, 19].

The linear model for a dynamic system at bus i is then $\dot{\mathbf{x}}_i = \mathbf{A}_i\mathbf{x}_i + \mathbf{B}_i^{(u)}\mathbf{u}_i + \mathbf{B}_i^{(P)}\Delta P_{Ei}$, where $\mathbf{u}_i$ indicates external inputs and ΔP_{Ei} indicates differential power injection around a given equilibrium operating point at that bus.

For the three-bus example shown in figure 3.1, the parameters for the component at bus one, modeled as a generator (3.1), are: $D = 1.5$, $T_{CH} = 0.2$, $R = 0.05$, $M = 10$, $K = 1/(0.2R) = 100$, yielding a component dynamic model of :

$$\frac{d}{dt}\begin{bmatrix}\Delta a\\ \Delta P_m\\ \Delta\omega_r\\ \Delta\delta\end{bmatrix} = \begin{bmatrix}-5 & 0 & 100 & 0\\ 0.2 & -0.2 & 0 & 0\\ 0 & -0.1 & -0.15 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}\Delta a\\ \Delta P_m\\ \Delta\omega_r\\ \Delta\delta\end{bmatrix} + \begin{bmatrix}0 & 0\\ 1 & 0\\ 0 & -0.1\\ 0 & 0\end{bmatrix}\begin{bmatrix}\Delta\omega_0\\ L_{ref}\end{bmatrix} + \begin{bmatrix}0\\ 0\\ -0.1\\ 0\end{bmatrix}\left[\Delta P_E\right], \quad (3.3)$$

with eigenvalues at 0, $-0.1335 \pm 0.6365i$, and -5.0830.

The parameters for the component at bus two, modeled as a generator (3.1) are: $D = 1.5$, $T_{CH} = 0.3$, $R = 0.04$, $M = 5$, $K = 1/(0.2R) = 100$, yielding a component dynamic model of :

$$\frac{d}{dt}\begin{bmatrix}\Delta a\\ \Delta P_m\\ \Delta\omega_r\\ \Delta\delta\end{bmatrix} = \begin{bmatrix}-5 & 0 & 125 & 0\\ 0.3 & -0.3 & 0 & 0\\ 0 & -0.2 & -0.3 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}\Delta a\\ \Delta P_m\\ \Delta\omega_r\\ \Delta\delta\end{bmatrix} + \begin{bmatrix}0 & 0\\ 1 & 0\\ 0 & -0.2\\ 0 & 0\end{bmatrix}\begin{bmatrix}\Delta\omega_0\\ L_{ref}\end{bmatrix} + \begin{bmatrix}0\\ 0\\ -0.2\\ 0\end{bmatrix}\left[\Delta P_E\right], \quad (3.4)$$

with eigenvalues at 0, $-0.15 \pm 1.2155i$, and -5.3.

The parameters for the component at bus three, modeled as a load (3.2), are: $D = 1.5$, $M = 1$, yielding

a component dynamic model of :

$$\frac{d}{dt}\begin{bmatrix}\Delta P_L\\ \Delta\omega_r\\ \Delta\delta\end{bmatrix} = \begin{bmatrix}0 & 0 & 0\\ -1 & -1.5 & 0\\ 0 & 1 & 0\end{bmatrix}\begin{bmatrix}\Delta P_L\\ \Delta\omega_r\\ \Delta\delta\end{bmatrix} + \begin{bmatrix}1\\ 0\\ 0\end{bmatrix}\left[P_L^{\text{rate}}\right] + \begin{bmatrix}0\\ -1\\ 0\end{bmatrix}\left[\Delta P_E\right], \tag{3.5}$$

with eigenvalues at 0×2, and -1.5.

Each of the component dynamic models described above can be written more concisely as

$$\dot{x}_i = \mathbf{A}_i\mathbf{x}_i + \mathbf{B}_i\mathbf{u}_i, \tag{3.6}$$

where the subscript (i) indicates the applicable bus. The output equation is similarly expressed as

$$y_i = \mathbf{C}_i\mathbf{x}_i. \tag{3.7}$$

This notation will be used further in the next section.

3.2 Modeling the System Dynamics

The complete dynamic system model is constructed by combining the component models as follows [20]:

$$\begin{aligned}
\mathbf{x}_s &= [\mathbf{x}_1^T, \mathbf{x}_2^T, ...\mathbf{x}_n^T]^T\\
\mathbf{u}_s &= [\mathbf{u}_1^T, \mathbf{u}_2^T, ...\mathbf{u}_n^T]^T\\
\mathbf{P} &= [\Delta P_{E1}, \Delta P_{E2}, ...\Delta P_{En}]^T\\
\mathbf{A}_s &= \text{blockdiag}(\mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_n)\\
\mathbf{B}_s^{(u)} &= \text{blockdiag}(\mathbf{B}_1^{(u)}, \mathbf{B}_2^{(u)}, ..., \mathbf{B}_n^{(u)})\\
\mathbf{B}_s^{(P)} &= \text{blockdiag}(\mathbf{B}_1^{(P)}, \mathbf{B}_2^{(P)}, ..., \mathbf{B}_n^{(P)})
\end{aligned}$$

Some additional bookkeeping is required here. The $\Delta\delta$s above are angle deviations due to small deviations in ω_r from a nominal 60 Hz operating frequency (i.e., $\omega_r = 2\pi 60 + \Delta\omega_r$). These deviations increase through time at the rate of $\Delta\omega_r$. The important angle for network state estimation purposes is the instantaneous angle differences between the various buses. Therefore a reference bus is assigned and the network state is

defined as the angle difference between the remaining angles and the reference bus, $\delta_i = \Delta\delta_i - \Delta\delta_{\text{ref}}$

Continuing the three-bus example, (3.3), (3.4), (3.5), are combined to make the composite state vector in terms of absolute angles as,

$$\mathbf{x}_s = [\Delta a_1, \Delta P_{m1}, \Delta\omega_{r1}, \Delta\delta_1, \Delta a_2, \Delta P_{m2}, \Delta\omega_{r2}, \Delta\delta_2, \Delta P_{L3}, \Delta\omega_{r3}, \Delta\delta_3]^T$$

with a system state transition matrix of:

$$\begin{bmatrix}
-5 & 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.2 & -0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -0.1 & -0.15 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -5 & 0 & 125 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.3 & -0.3 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -0.2 & -0.3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1.5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}$$

which has eigenvalues of: 0×4, -5.083, $-0.1335 \pm 0.6365i$, $-0.1500 \pm 1.2155i$ -5.3, and -1.5.

Subtracting the reference bus angle, $\Delta\delta_{\text{ref}}$, from the other bus angles and reordering the state variables, the state vector is,

$$\mathbf{x}_s = [\Delta a_1, \Delta P_{m1}, \Delta\omega_{r1}, \Delta a_2, \Delta P_{m2}, \Delta\omega_{r2}, \Delta P_{L3}, \Delta\omega_{r3}, \delta_2, \delta_3]^T$$

with a system state transition matrix of:

$$\begin{bmatrix} -5 & 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.2 & -0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.1 & -0.15 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -5 & 0 & 125 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & -0.3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.2 & -0.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1.5 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

with eigenvalues of 0×3, -5.0830, $-0.1335 \pm 0.6365i$$-5.3000$, $-0.15 \pm 1.2155i$, and -1.5.

3.3 Coupling the System Dynamics through the Network

The power injections, ΔP_E in the above model, are a nonlinear function of the bus voltage angles, $\mathbf{P} = \mathbf{f}(\delta_s) = \mathbf{f}(\mathbf{S}\mathbf{x}_s)$, where δ_s is the network state and is extracted from the dynamic state using a selection matrix $\delta_s = \mathbf{S}\mathbf{x}_s$. Applying the assumptions of Sec. 2.2, plus the small angle linearization of $\sin(\delta) \approx \delta$, the power injections can be approximated as,

$$\mathbf{P} = \mathbf{B}\delta, \tag{3.8}$$

where $\mathbf{B}$ is the susceptance matrix (i.e., the imaginary part of the admittance matrix, $\mathbf{Y}$) [30]. The system dynamic model is therefore coupled as follows:

$$\begin{aligned} \dot{\mathbf{x}}_s &= \mathbf{A}_s\mathbf{x}_s + \mathbf{B}_s^{(P)}\mathbf{P} + \mathbf{B}_s^{(u)}\mathbf{u} \\ &= \mathbf{A}_s\mathbf{x}_s + \mathbf{B}_s^{(P)}\mathbf{B}\mathbf{S}\mathbf{x}_\mathbf{s} + \mathbf{B}_s^{(u)}\mathbf{u} \\ &= \left(\mathbf{A}_s + \mathbf{B}_s^{(P)}\mathbf{B}\mathbf{S}\right)\mathbf{x}_s + \mathbf{B}_s^{(u)}\mathbf{u}. \end{aligned} \tag{3.9}$$

Converting (3.9) to discrete-time yields a dynamic system equation in the form [5]

$$\mathbf{x}_{(k+1)} = \mathbf{A}_d \mathbf{x}_{(k)} + \mathbf{B}_d \mathbf{u}_{(k)} \tag{3.10}$$

where

$$\mathbf{A}_d = e^{\left(\mathbf{A}_s + \mathbf{B}_s^{(P)} \mathbf{BS}\right)}$$

and

$$B_d = \left(\mathbf{A}_s + \mathbf{B}_s^{(P)} \mathbf{BS}\right)^{-1} \left(\mathbf{A}_d - \mathbf{I}\right) \mathbf{B}_s^{(u)}. \tag{3.11}$$

The parenthetical subscript in 3.10 indicates the sample number. We will use this discrete-time version of the above equation for dynamic state estimation.

Continuing the three-bus example, the susceptance matrix for the network is

$$\begin{bmatrix} -2.25 & 1.2500 & 1.0000 \\ 1.25 & -2.6786 & 1.4286 \\ 1.00 & 1.4286 & -2.4286 \end{bmatrix}. \tag{3.12}$$

Using (3.9) and (3.12) to couple the component dynamic systems together, the state transition matrix is

$$\begin{bmatrix} -5 & 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.2 & -0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.1 & -0.15 & 0 & 0 & 0 & 0 & 0 & 0.1250 & 0.1 \\ 0 & 0 & 0 & -5 & 0 & 125 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & -0.3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.2 & -0.3 & 0 & 0 & -0.5357 & 0.2857 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1.5 & 1.4286 & -2.4286 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix},$$

with eigenvalues of -5.0823, -5.2949, -0.0948, $-0.1493 \pm 0.7405i$, $-0.6521 \pm 1.3505i$, $-0.1876 \pm 1.4110i$, and 0.

Here it is important to note that even in the coupled state transition matrix, we have an eigenvalue equal

to zero. This is problematic for multiple reasons, but most importantly if we want to discretize the model as shown above, $\left(\mathbf{A}_s+\mathbf{B}_s^{(P)}\mathbf{BS}\right)$ must be invertable to satisfy (3.11). This is where the reference bus comes in. By defining the network state as the angle difference between each bus and a bus designated as the reference, $\left(\mathbf{A}_s+\mathbf{B}_s^{(P)}\mathbf{BS}\right)$ becomes nonsingular and invertible.

3.4 Dynamic Estimation With Additional Measurements

An additional benefit of modeling the dynamics in this manner is that it opens the door for incorporating additional measurements that cannot be incorporated with the existing models. For example, because this model incorporates information about the generation, a measurement of the mechanical power supplied to a generator can also be incorporated and may improve the accuracy of the state estimation result. Similarly, incorporating load data or forecasts can lead to further improvements.

Continuing the three-bus example, assume that a direct measurement of P_{L3} is available. This element of the state vector can now be removed from the state and incorporated as an input at bus three as:

$$\frac{d}{dt}\begin{bmatrix}\Delta\omega_r\\ \delta\end{bmatrix}=\begin{bmatrix}-D/M & 0\\ 1 & 0\end{bmatrix}\begin{bmatrix}\Delta\omega_r\\ \delta\end{bmatrix}+\begin{bmatrix}-1/M\\ 0\end{bmatrix}[P_{L3}]+\begin{bmatrix}1/M\\ 0\end{bmatrix}[\Delta P_E]\,. \tag{3.13}$$

Replacing (3.2) with (3.13) and forming the coupled system equation as described in sections 3.1 and 3.3, the coupled state transition matrix after this modification is

$$\begin{bmatrix}
-5 & 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0\\
0.2 & -0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & -0.1 & -0.15 & 0 & 0 & 0 & 0 & 0.1250 & 0.1\\
0 & 0 & 0 & -5 & 0 & 125 & 0 & 0 & 0\\
0 & 0 & 0 & 0.3 & -0.3 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & -0.2 & -0.3 & 0 & -0.5357 & 0.2857\\
0 & 0 & 0 & 0 & 0 & 0 & -1.5 & 1.4286 & -2.4286\\
0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0\\
0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0
\end{bmatrix},$$

with eigenvalues of -5.0823, -5.2949, -0.0948, $-0.1493\pm0.7405i$, $-0.6521\pm1.3505i$, and $-0.1876\pm1.4110i$.

3.5 Dynamic Estimation Without Additional Measurements

If additional measurements are not available, the dynamic modeling methodology described above may still be employed to benefit the state estimation process. The expected behavior of the exogenous inputs can be modeled and incorporated into the overall system dynamic model. This typically requires the additional states to the dynamic state vector and the corresponding associated computational load. To distinguish these dynamic state vectors, the standard state estimate vector which only contains the bus component states is identified as $\hat{x}_{(k/\bullet)}$ with the corresponding network state identified as $\hat{\delta}_{(k/\bullet)}$. The state state vector augmented by additional states to model the exogenous inputs is identified as $\hat{x}'_{(k/\bullet)}$ with the corresponding network state estimate identified as $\hat{\delta}'_{(k/\bullet)}$.

Many models are available for the exogenous input [8, 11, 20, 29]. This thesis will focus on a simple accumulator model [11, 29]. The accumulator model treats the exogenous load as the accumulation of random perturbations. When applied to the power network, the magnitude of the perturbations (Gaussian noise) is related to the expected ramp rate of the power demand on the network.

3.6 Simplifying the Dynamic Model

For large power systems, the computational workload of the state estimator may become prohibitively expensive. It is important to identify areas in the dynamic model where improvements in modeling accuracy do not directly contribute to improvements in accuracy in the network state estimate. As discussed in Sec. 2.1, there may be some components of the dynamic state that are too fast to provide a significant contribution to the predicted state. As the state estimator receives only minimal improvement from this information, it is possible that these elements of the dynamic state are not worth the additional computational load to model. Therefore it is desirable to modify the model so that these non-contributing modes are no longer simulated.

One effective method of reducing the model is to perform a singular value decomposition on the state transition matrix, $\mathbf{A}$, and reduce the model by any singular values that are more than an order of magnitude faster than the SCADA sampling time. This is a well studied method [5] but requires the full system model to be developed and then simplified. This reduction leads to two potential difficulties. Either the $\mathbf{A}$ matrix will be rank deficient which will be important later on (see Sec. 3.8.2), or a change of variables needs to be applied to the dynamic state to maintain a full rank state transition matrix. If a change of variables is performed, then the dynamic state being estimated may no longer correspond to physical variables. An additional transform must be included to extract the original network state variables. These transformations

are typically not computationally intense so that the computational gains achieved through state reduction are maintained.

Another method uses the small dynamic systems formulated at each bus. When these component models are coupled through the network, the δ and $\Delta\omega$ elements of the dynamic state are strongly affected but the remaining elements are not. In effect, the rotating mass acts as a low pass filter between the fast elements of the various component dynamic states. This can be seen by looking at the poles of the uncoupled systems. There is one free integrator (i.e., a pole at the origin) corresponding to each δ. When the systems are coupled together, all these poles except the one corresponding to the reference angle, move towards the left half-plane. The derivative of δ, $\Delta\omega$ is therefore also strongly affected.

It is therefore necessary to keep δ and $\Delta\omega$ in the model to maintain the proper modeling of the network-component interaction. The other elements of the component dynamic state vectors are available to be simplified via singular perturbation or other methods of model reduction. This second method is especially useful when the network is configured with weak network coupling and large rotational inertias at the busses.

The effect of dynamic state reduction on network state estimation accuracy is explored in Sec. 4.5.

3.7 Dynamic Modeling and Estimation for Quasi-Static Systems

The modeling methodology described in earlier requires that large inertial components be present at every single bus of the network. In an actual power network, this may not always be the case. For example, loads such as an electric arc furnace typically used in steelmaking industries can exhibit large, nearly instantaneous multi-megawatt changes in real power load. In these situations it may be beneficial to take a different approach regarding the dynamic power system model.

If we return to the static network model described in Ch. 2, we see that the network state is treated as though it were algebraically dependent on the bus injections. This model is equivalent to assuming that the dynamics of the bus components (specifically $\dot{\omega}$) have reached steady state so that the network is in a quasi-static state. Furthermore, we recognize that the δ portion of A_s does not contribute to $\dot{x}_s$

We now apply this constraint to (3.9), and focus on the ω and δ terms. We are left with the network swing equation $\dot{\Delta\omega} = \frac{1}{M}(\Delta P_m - \Delta P_E)$ [27]. From (3.8), ΔP_E is a function of the network state, δ and from (3.1), ΔP_m and ΔP_L (collectively represented as Γ) is a function of the component dynamic state (not

including δ) and exogenous input. Equation (3.9) may now be expressed in the following form.

$$\begin{bmatrix} \Delta\dot{\omega}_r \\ \dot{\delta} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{B}/M \\ \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \Delta\omega_r \\ \delta \end{bmatrix} + \begin{bmatrix} \Gamma/M \\ 0 \end{bmatrix},$$

which can be simplified to

$$\mathbf{0} = \mathbf{B}\delta + \Gamma, \tag{3.14}$$

the algebraic relationship between δ and Γ.

At each measurement snapshot, incremental changes may have occurred to Γ and therefore to the network state δ. These incremental changes may be interpreted as a perturbation to the previous state $\delta_{(k-1)}$ driving the network towards the present state $\delta_{(k)}$.

Analyzing the effect of an incremental change in Γ from time $(k-1)$ to time (k), a pseudo-dynamic model may be derived from (3.14) as

$$\delta_{(k)} = \delta_{(k-1)} - \mathbf{B}^{-1}(\Gamma_{(k)} - \Gamma_{(k-1)}),$$

where $\Gamma_{(k)}$ would likely be the output from load forecast or some other external source [7, 8, 28, 37]. Using this formulation, the Kalman filter equations may be applied directly if $\Gamma_{(k)}$ is available, or if $\Gamma_{(k)}$ is unavailable an accumulator model,

$$\Gamma_{(k)} = \Gamma_{(k-1)} + \mathbf{v}_{(k-1)},$$

may be applied where the incremental change in Γ is modeled as additive Gaussian white noise $\mathbf{v}_{(k-1)}$ [11, 29].

3.8 Dynamic State Estimation

This section describes the modeling methodology used to effectively apply dynamic state estimation concepts to electric power system network state estimation.

3.8.1 Formulation

The purpose of dynamic state estimation is to find the expected value for the dynamic state given the measurements and the a priori value of the dynamic state given by the previous dynamic state estimate and the input. The dynamic state estimator is optimized over the following goals: the expected value of the dynamic state estimation error should be zero $\mathrm{E}\{e\} = \mathrm{E}\{\hat{x} - x\} = 0$, and the dynamic state estimate error variance $\mathrm{E}\{e^T e\} = \mathrm{trace}(\mathrm{E}\{ee^T\})$ should be minimized. The Kalman filter provides an optimal solution to

this dynamic state estimator formulation given a linear system with additive white Gaussian noise [22]. It is formulated as follows:

Process/Measurement Model

Dynamic Model: $\mathbf{x}_{(k)} = \mathbf{A}\mathbf{x}_{(k-1)} + \mathbf{B}\mathbf{u}_{(k-1)} + \mathbf{w}_{(k-1)}$

Measurement Model: $\tilde{\mathbf{z}}_{(k)} = \mathbf{h}(\mathbf{x}_{(k)}) + \mathbf{v}_{(k)}$

Initial Values:

State Estimation Error: $\mathbf{e}_{(k)} = \hat{\mathbf{x}}_{(k/k)} - \mathbf{x}_{(k)}$

Initial State Error Covariance Matrix: $\mathbf{P}_{(0/0)} = \mathrm{E}\{\mathbf{e}_{(0)}\mathbf{e}_{(0)}^T\}$

Initial State Estimate: $\hat{\mathbf{x}}_{(0)} = \mathrm{E}\{\mathbf{x}_{(0)}\}$

Measurement Error Covariance: $\mathbf{V} = \mathrm{E}\{\mathbf{v}\mathbf{v}^T\}$

Process Noise Covariance: $\mathbf{W} = \mathrm{E}\{\mathbf{w}\mathbf{w}^T\}$

Prediction:

Predicted State: $\hat{\mathbf{x}}_{(k/k-1)} = \mathbf{A}\hat{\mathbf{x}}_{(k-1/k-1)} + \mathbf{B}\tilde{\mathbf{u}}_{(k-1)}$

Predicted State Error Covariance: $\mathbf{P}_{(k/k-1)} = \mathbf{A}\mathbf{P}_{(k/k-1)}\mathbf{A}^T + \mathbf{B}\mathbf{W}\mathbf{B}^T$

Correction:

Kalman Gain: $\mathbf{K}_{(k)} = \mathbf{P}_{(k/k-1)}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{(k/k-1)}\mathbf{H}^T + \mathbf{V})^{-1}$

$\mathbf{H} = \frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}}$

Corrected State: $\hat{\mathbf{x}}_{(k/k)} = \hat{\mathbf{x}}_{(k/k-1)} + \mathbf{K}_{(k)}(\tilde{\mathbf{z}}_{(k)} - \mathbf{h}(\hat{\mathbf{x}}_{(k/k-1)}))$

Corrected State Error Covariance: $\mathbf{P}_{(k/k)} = (\mathbf{I} - \mathbf{K}_{(k)}\mathbf{H})\mathbf{P}_{(k/k-1)}$

When applied to a power system, the state estimate is initialized from the static network state estimator. This works well when the state in question is the network state. However, when the goal is to estimate the dynamic state, the subset of the dynamic state $\mathbf{x}_0$ that does not correspond to the network state δ is undefined. Additionally, any portion of the network state which is not statically observable will also be undefined [33]. This distribution would correspond to an infinite or undefined diagonal element in the state covariance matrix.

To avoid the difficulties of dealing with infinite matrix elements, the following alternate formulation of the Kalman filter, called the information filter, is considered. This is the optimal linear filter formulated to track the Fisher information matrix rather than the state error covariance matrix [14]. The information filter is formulated as follows:

Initial Values:

Initial Information Matrix: $\mathbf{Y}_{(0/0)} = \mathbf{P}_{(0/0)}^{-1}$

Initial Information: $\hat{\mathbf{y}}_{(0)} = \mathbf{Y}_{(0/0)}\hat{\mathbf{x}}_{(0)}$

Correction:

Measurement Information Matrix: $\mathbf{I}_{(k)} = \mathbf{H}^T\mathbf{V}^{-1}\mathbf{H}$

Measurement Information: $\mathbf{i}_{(k)} = \mathbf{H}^T\mathbf{V}^{-1}\tilde{\mathbf{z}}_{(k)}$

Information Matrix Update: $\mathbf{Y}_{(k/k)} = \mathbf{Y}_{(k/k-1)} + \mathbf{I}_{(k)}$

Information Update: $\mathbf{y}_{(k/k)} = \mathbf{y}_{(k/k-1)} + \mathbf{i}_{(k)}$

Prediction:

$\mathbf{M}_{(k)} = \mathbf{A}_{(k)}^{-T}\mathbf{Y}_{(k-1/k-1)}\mathbf{A}_{(k)}^{-1}$

$\mathbf{C}_{(k)} = \mathbf{M}_{(k)}\left(\mathbf{M}_{(k)} + \mathbf{W}^{-1}\right)^{-1}$

$\mathbf{L}_{(k)} = \mathbf{I} - \mathbf{C}_{(k)}$

Information Matrix Prediction: $\mathbf{Y}_{(k/k-1)} = \mathbf{L}_{(k)}\mathbf{M}_{(k)}\mathbf{L}_{(k)}^T + \mathbf{C}_{(k)}\mathbf{W}^{-1}\mathbf{C}_{(k)}^T$

Information Prediction: $\hat{\mathbf{y}}_{(k/k-1)} = \mathbf{L}_{(k)}\mathbf{A}^{-T}\hat{\mathbf{y}}_{(k-1/k-1)}$

Transitioning from tracking the state error covariance matrix to the information matrix moves the complexity from the update step to the prediction step. This formulation also requires that the state transition matrix $\mathbf{A}$ be invertible, which is important when choosing the method of model reduction as described in Sec. 3.6.

The use of the information matrix has multiple benefits when applied to an electric power system. 1) When a subset of the state is unobservable due to removal of bad data, the corresponding elements of $I_{(k)}$ may be set to zero to indicate that there is no information present for that element of the state. 2) The formulation of the update step lends itself well to parallel processing. This is important as a power system may have thousands of state variables and therefore the state estimation will have a large computational load.

3.8.2 Performance Metric

The standard performance metric for the Kalman filter is the trace of the state error covariance matrix. This trace is equal to the sum of the eigenvalues of the matrix, but more importantly it is equal to the sum of the individual state error variances,

$$\begin{aligned}\sum_k \mathbf{P}_{kk} &= \text{trace}(\mathbf{P}) \\ &= \sum_k \lambda_k^P\end{aligned}$$

$$= \sum_k \sigma_k^2 \tag{3.15}$$

where σ_k^2 is the variance of the kth state variable and λ_k^P is the kth eigenvalue of the matrix $\mathbf{P}$. In the event of an unobservable network state, this sum is undefined due to the error covariances of the unobservable states being undefined. It is desirable to have a defined value for the performance metric even when the network state is statically unobservable. Therefore an alternate metric is considered.

Consider the information matrix, $\mathbf{Y}$. The information matrix has the property that when a state is unobservable, the information for that state is zero. Thus, the trace of an information matrix $\sum_k \mathbf{Y}_{kk}$ with unobservable states will still be defined.

This metric provides an effective comparison against the performance of another state estimator that estimate the full dynamic state. However, this is not directly useful for the purpose of comparing against a state estimator that only estimates the network state as is the industry standard. It is therefore necessary to extract a metric from the information matrix that is suitable for comparison to estimators that only estimate the network state.

The dynamic state vector contains the network state δ within the first $n-1$ elements and θ is defined as the remaining state variables in x. The dynamic state x is therefore $[\delta^T, \theta^T]^T$. Given this method of partitioning the state, the state error covariance matrix $\mathbf{P}$ can be written as follows

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_{[\delta\delta]} & \mathbf{P}_{[\delta\theta]} \\ \mathbf{P}_{[\theta\delta]} & \mathbf{P}_{[\theta\theta]} \end{bmatrix} = \mathbf{Y}^{-1}.$$

Considering the discussion above, the desired performance metric as it pertains to δ only is trace($\mathbf{P}_{[\delta\delta]}^{-1}$). Applying the matrix inversion lemma [5],

$$\mathbf{P}_{[\delta\delta]}^{-1} = \mathbf{Y}_{[\delta\delta]} - \mathbf{Y}_{[\delta\theta]}\mathbf{Y}_{[\theta\theta]}^{-1}\mathbf{Y}_{[\theta\delta]} \tag{3.16}$$

$$= \mathbf{I}_{[\delta\delta]}^{dynamic} \tag{3.17}$$

gives us the information matrix corresponding to δ, $\mathbf{I}_{[\delta\delta]}^{dynamic}$ that we desire. It should be noted that the relationship between the full measurement matrix $\mathbf{I}$ and the portion corresponding to δ only is

$$\mathbf{I} = \begin{bmatrix} \mathbf{I}_{[\delta\delta]} & \mathbf{I}_{[\delta\theta]} \\ \mathbf{I}_{[\theta\delta]} & \mathbf{I}_{[\theta\theta]} \end{bmatrix} = \begin{bmatrix} \mathbf{I}_{[\delta\delta]} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{3.18}$$

as the measurements do not directly affect any of the θ states.

The trace of $\mathbf{Y}$ has several properties such as:

$$\begin{aligned}
\sum_k \mathbf{Y}_{kk} &= \text{trace}(\mathbf{Y}) \\
&= \text{trace}(\mathbf{P}^{-1}) \\
&= \sum_k \lambda_k^Y \qquad (3.19) \\
&= \sum_k \frac{1}{\lambda_k^P} \\
\text{and} \quad &= \sum_k \frac{1}{\sigma_k^2} \text{ for diagonal } \mathbf{Y}, \mathbf{P} \qquad (3.20)
\end{aligned}$$

The relationships in (3.15) and (3.20) are convenient as $\sum \sigma_P^2$ and $\sum 1/\sigma_P^2$ can easily be calculated from simulation.

Unfortunately, in the types of dynamic systems under study in this thesis, $\mathbf{P}$ is rarely diagonal, so this relationship in (3.20) does not always hold. Therefore, instead of $\sum_k \mathbf{Y}_{kk}$ we will use one of the byproducts from (3.15) and (3.20),

$$\sum_k \frac{1}{\mathbf{P}_{kk}} = \sum_k \frac{1}{\sigma_k^2} \qquad (3.21)$$

as our primary metric. This metric remains easily accessible from empirical data but retains the property of remaining defined in the instance of unobservable states.

A comparison between the two metrics tr($\mathbf{Y}$) and $\sum 1/\sigma^2$, is accomplished by analyzing their values when applied to a simulated run of the IEEE 14-bus test system given in App. B. Plots of the metrics are shown in Figs. 3.2 and 3.3. For readability, the natural logarithm of the metric is plotted.

Figure 3.2 shows plots of the expected values for each performance metric based on the Kalman filter and information filter equations. We expect the numerical values to differ between the two metrics as the state error covariance matrix is not diagonal. As expected, the values in the graph tr($\mathbf{Y}$) (upper plot) and $\sum(1/\sigma_i^2)$ (lower plot) do differ. The trends between the two plots, however, are similar, i.e. a value that is higher in one metric is higher in the other. The plot of the metric for the static state estimator (red line) is constant and consistently below the dynamic estimators in both graphs. Similarly, the dynamic estimator which makes use of additional load information (blue line) is consistently above the dynamic state estimator which only uses the network measurements (green line).

The same trends and consistency between metrics are in Fig. 3.3 when applied to experimental data. In this situation, a simulation of the IEEE 14-bus test system was run 50,000 times with random noise

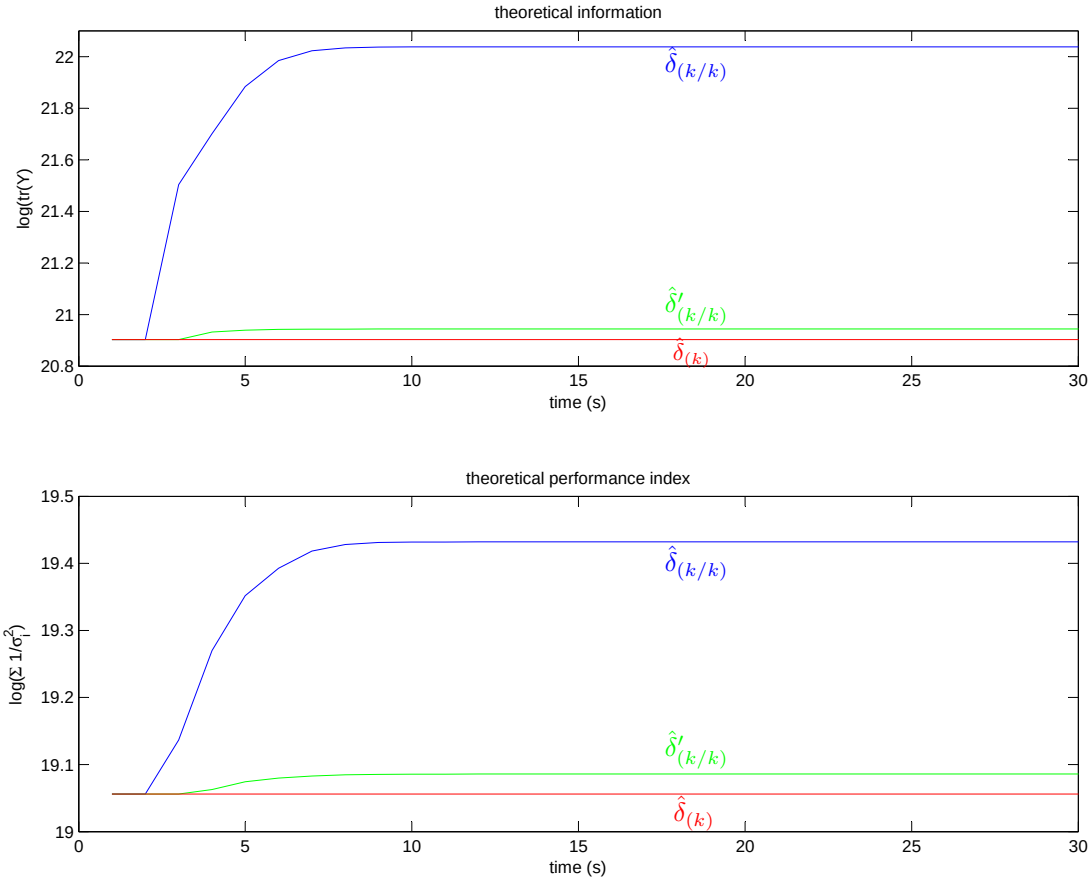


Figure 3.2: Comparison of expected performance metrics on the IEEE 14-bus test system.

corrupting the measurements and input. At each iteration the state error covariance matrix was calculated and recorded. The respective performance metrics were then calculated by either inverting the matrix and taking the trace ($\text{tr}(\mathbf{Y})$) or inverting the diagonal elements and taking the sum ($\sum(1/\sigma_i)$).

3.8.3 Implementation

The dynamic state estimator is implemented as follows.

1. **Process initial measurements through static network state estimator.** The measurements are passed to a static state estimator which evaluates the measurement vector to determine which states are observable. It then uses a weighted minimum squared error minimization algorithm to estimate the static state $\hat{\delta}_{(0)}$. This function also returns a vector indicating which states are observable and the expected uncertainty $\bar{\mathbf{I}}_{(0)}$ (measurement information matrix) of the state vector estimate returned.
2. **Initialize dynamic state and dynamic state error covariance matrix.** The dynamic state

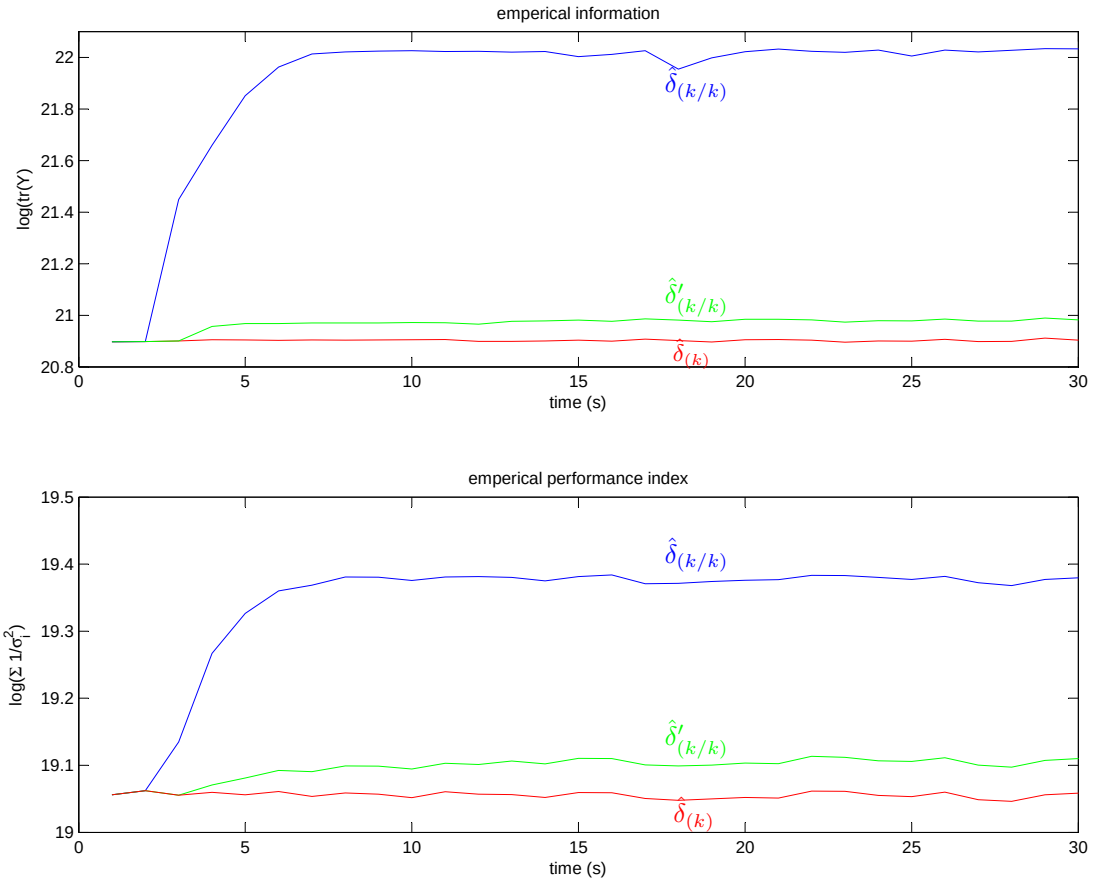


Figure 3.3: Comparison of actual performance metrics on the IEEE 14-bus test system.

estimate $\hat{\mathbf{x}}_{(k/0)}$ and dynamic state error covariance matrix $\hat{\mathbf{P}}_{(k/0)}$ are initialized as follows. The subset of the dynamic state corresponding to the network state is initialized to be equal to the value returned by the static state estimator. If additional initializing information regarding the initial value of the dynamic state is available, it is also incorporated at this time. The remainder of the dynamic state error covariance matrix is initialized to an arbitrarily large uncorrelated state error covariance matrix to correspond to the uncertainty in the initial guess of the dynamic state.

3. **Predict the dynamic state.** Using the dynamic system model and knowledge of the driving inputs, perform the dynamic estimation prediction step as described in Sec. 3.8.1. A value is calculated for both the dynamic state predicted estimate $\bar{\mathbf{x}}_{(k/k-1)}$ and the error covariance $\bar{\mathbf{P}}_{(k/k-1)}$.

4. **Process new measurements through static network state estimator.** As new measurements become available, process them through the static network state estimator to calculate a new static estimate for the network state $\hat{\delta}_{(k)}$.

5. **Update the dynamic state.** Using the new output from the static state estimator $\hat{\delta}_{(k)}$ and $\bar{\mathbf{I}}_{(k)}$, a correction to the dynamic state prediction is made. The result is an updated value for the dynamic state estimate $\hat{\mathbf{x}}_{(k/k)}$ and dynamic state covariance matrix $\hat{\mathbf{P}}_{(k/k)}$ as described in Sec. 3.8.1.

6. **Go to step 3.**

As can be seen in step 4, using the output from the static network state estimator to be an input to the information filter relieves us of the requirement to employ the more complicated nonlinear implementation of the Kalman or information filter [6]. This also allows us to leverage off of the decades of research and operation experience that has been invested into static network state estimation algorithms.

3.9 Conclusions

In this chapter we have shown that a dynamic model can be employed that couples together the dynamics of components attached to an electric power network. Furthermore, the connectivity provided by that electric network provides coupling between the dynamic systems to provide a model that accurately models the dynamic and interactions of the full system. This model may be employed to allow dynamic state estimation algorithms to be used in estimating the network state.

Chapter 4

Evaluation of Dynamic Network State Estimation

This chapter demonstrates the effectiveness of the dynamic network state estimation algorithms described in Ch. 3. These algorithms were evaluated using Monte Carlo simulation implemented in MATLAB. The algorithms were tested on the IEEE 14–bus (Fig. B.1) and IEEE 118–bus (Fig. B.2) test systems [17] described in appendix B over a 250 second simulated transient.

4.1 Simplifying Assumptions

The following simplifications were established to ease in computation and modeling.

1. **Real - Reactive power decoupling**. Real Power P, and Reactive Power Q are primarily functions of bus voltage angle δ and bus voltage magnitude V, respectively. Operational experience [50] and academic literature [24, 30, 1] have shown that state estimation results using decoupled $P - \delta$ and $Q - V$ equations provide accurate results.
2. **No Parallel Transmission Lines**. Parallel transmission lines in the IEEE test systems have been combined into single lines with the equivalent admittance of the two (or more) parallel lines. Reducing the network model in this way removes ambiguity in calculating branch flows or interpreting the effect of branch flow measurements on the network state. Individual branch flows can be calculated from the consolidated line flows through a simple division in post processing.

3. **Bus One is the Reference Bus**. The reference bus has been arbitrarily assigned to bus One for each of the test systems. This has no effect on the accuracy of the result, only the reference value that other angles are compared to. The effect of using any other bus as the reference can be achieved by subtracting the angle of the new reference bus from all other bus angles. Alternatively, the buses could be renumerated in order to set the desired reference bus to 1.

4. **No Zero Injection Busses**. All busses have been established with either a generator, synchronous condenser, or a load. For the dynamic algorithms to work correctly, each bus must have some nonzero value for its inertia. This could alternatively been accomplished by reducing the subnetwork containing the zero-injection by replacing it with an equivalent network. The original line flows could then be back-calculated from the results of the equivalent network in post-processing.

4.2 Simulation Test Data

Measurement data was generated using Matlab Simulink. The Simulink model (see App. B) implemented the dynamic bus models described in Sec. 3.1. Line flows were calculated based on the decoupled $P - \delta$ model where the real power flow is a function of bus angle difference only, $P_{ij} = \text{imag}(y_{ij}) \sin(\delta_i - \delta_j) = b_{ij} \sin(\delta_i - \delta_j)$. For comparison purposes, linearized measurement data was also computed where the small angle approximation is used so that $P_{ij} = b_{ij}(\delta_i - \delta_j)$.

For the IEEE 118-bus system, generators with time constants of 10 seconds for bus 69 and 5 seconds for bus 89 were used. Synchronous condensers with time constants of 5 seconds were used on the remaining PV Busses. Inertial loads with time constants of 1 second were applied to the remaining busses. For the IEEE 14-bus system, generators with time constants of 10 seconds for bus 1 and 5 seconds for bus 2 were used. Synchronous condensers with time constants of 5 seconds were used on the remaining PV Busses (3, 6, and 8). Inertial loads with time constants of 1 second were applied to the remaining busses. These time constants represent typical time constants associated with real power systems [27].

The following load perturbation was simulated. On both the 118-bus and 14-bus systems, bus 3 was subjected to a load increase from 0 to 0.2 p.u. at a ramp rate of 0.02 p.u. per sec. This load was held constant for 30 seconds. The load then decreased from 0.2 p.u. to 0.1 p.u. at a ramp rate of -0.1 p.u. per sec. This transient can be seen in Fig. 4.1.

The test systems were simulated for 251 seconds (0 to 250) with data collected each second. All the bus real power injections and line real power flows were recorded as well as the loads and bus angles.

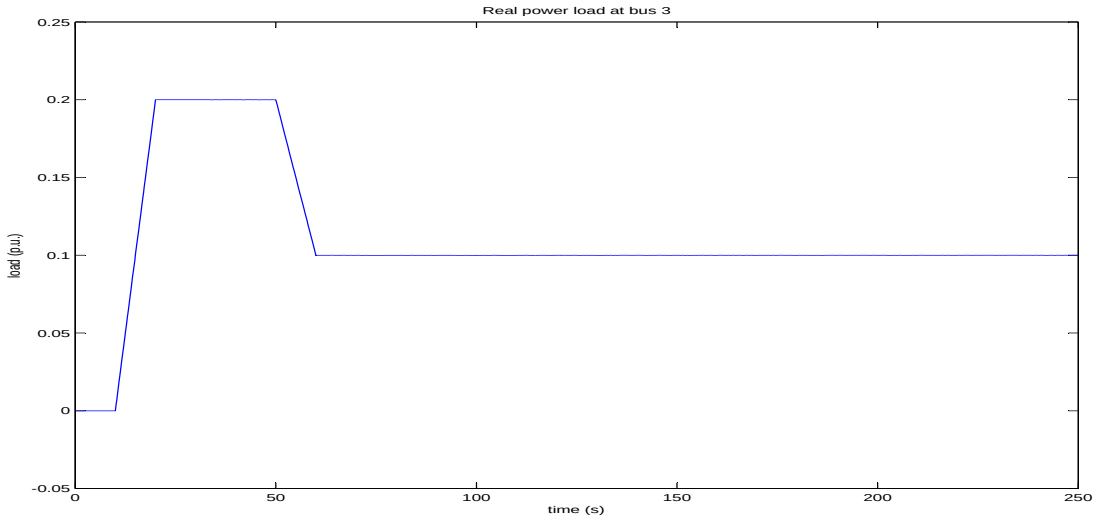


Figure 4.1: Load transient for simulation.

These measurements were then corrupted with additive white Gaussian noise (AWGN) and fed into the state estimation algorithms to evaluate their performance. At various times in the processing, situations are introduced where different subsets of the full measurement vector are available. The four possibilities for the measurement vector are as follows:

1. **All measurements available**. The full measurement vector including one measurement of each bus real power injection and one measurement of each branch real power flow.
2. **All injections available**. The measurement vector consists of one measurement of each bus real power injection.
3. **All flows available**. The measurement vector consists of one measurement of each branch real power flow.
4. **Most flows available**. The measurement vector consists of one measurement of each branch real power flow except for the branches connecting to bus 3, making that bus unobservable.

The state estimation algorithms were tested using two levels of simplifying assumptions regarding the linearization of the network (as realized in Simulink) and were analyzed using linear algorithms perform the static network state estimation. The specific methodologies employed were:

1. **Linear Data - Linear Estimation**. Data was generated using the linear approximation. The Static State Estimation step of the Dynamic State Estimator was accomplished using a linear measurement model. This was used as a test to verify the algorithms performed as expected and the results are presented in Sec. 4.4.

2. **Nonlinear Data - Linear Estimation**. Data was generated using the nonlinear measurements based on the sine of the angle difference. The Static State Estimation step of the Dynamic State Estimator was accomplished using a linear measurement model and the results are presented in Sec. 4.4.

4.3 Evaluation of Algorithms Using Linearized Test Data

Three methods of estimating the network state were evaluated:

1. **Static Network State Estimation**. The classic method of network state estimation in electric power industry assumes that no correlation exists between measurements to each measurement snapshot is estimated individually using a maximum likelihood method. The network state when estimated using this method is denoted as $\hat{\delta}_{(k)}$.
2. **Standard Dynamic Network State Estimation**. The dynamic network state model discussed in Ch. 3 is applied to a Kalman filter formulation. This method requires knowledge of the physical loads at the buses. The network state when estimated using this method is denoted as $\hat{\delta}_{(k/k)}$.
3. **Augmented Dynamic Network State Estimation**. The dynamic model used in estimating $\hat{\delta}_{(k/k)}$ is employed here except that knowledge of the physical loads is not available, thus this method uses only the information available to the static network state estimator. This is known as *augmented* because the dynamic state is augmented with the bus component mechanical loads to be estimated in addition to the standard dynamic state. The network state when estimated using this method is denoted as $\hat{\delta}'_{(k/k)}$.

The two performance metrics discussed in Sec. 3.8.2, the trace of the information matrix tr($\mathbf{Y}$) and the sum of the inverse state variances, $\sum 1/\sigma_i$, are applied to evaluate the comparative performance of the network state estimation methods above. The $\sum 1/\sigma_i$ performance metric is an indication of the ability of the estimator to track the network state directly. The tr($\mathbf{Y}$) performance metric gives an indication of the overall cross correlation between network state estimates.

4.3.1 Network State Estimation Performance : 14-Bus Test System

Three curves are shown in the following graphs. The lowest (red) curve represents the performance of $\hat{\delta}_{(k)}$. As expected, the performance of the static network state estimator is constant as long as the number of measurements is constant. This is because the network state estimate is based only on the measurements presently available to it at the present time.

The middle (green) curve represents the performance of $\hat{\delta}'_{(k/k)}$. The performance of the augmented dynamic network state estimator upon initialization is equal to that of the static network state estimator as they are processing exactly the same set of measurements. As more measurements are processed, the performance of the augmented dynamic network state estimator improves over the static network state estimator as the dynamic component model allows information from the past measurements to contribute to the new network state estimate.

The top (blue) curve represents the performance of $\hat{\delta}_{(k/k)}$. Similar to the augmented dynamic network state estimator, the performance standard dynamic network state estimator is equal to that of the static network state estimator. The performance immediately begins to improve as information about the bus loads allows for prediction of the network state. As expected, the inclusion of the additional bus load information as described in Sec. 3.4 allows for a further increase in performance above the augmented dynamic network state estimator.

Figure 4.2 shows the expected value of $\text{tr}(\mathbf{Y})$ and Fig. 4.3 shows the expected value of $\sum 1/\sigma_i$ based on the values of the respective state error covariance matrices from the Kalman filter equations. To improve graph readability, the natural logarithm of the metric is plotted.

Figure 4.4 shows the empirical value of $\text{tr}(\mathbf{Y})$ and Fig. 4.5 shows the empirical value of $\sum 1/\sigma_i$ based on a Monte Carlo simulation of the IEEE 14-bus test system simulated over 50,000 passes with random noise corrupting the measurements and measurements of the physical bus loads (inputs).

The empirical and theoretical graphs track each other well. There are a few points of interest to identify:

1. For numerical stability, additional noise is introduced into the information matrix so that it will not become singular. This is necessary to be able to invert it to retrieve the sum of inverse variances performance metric during simulation. As such, the expected values for the performance metrics are sometimes slightly lower than the experimental value.

2. The dip in performance at 150 seconds is due to a reduction in measurements. Only the 20 branch flows are available from 150 to 170 seconds.

3. The dip in the experimental performance at 200 seconds is due to a reduction in measurements. Only the 14 bus injections are available from 200 to 210 seconds.

4. A dip in performance of the dynamic estimator 10-20 seconds in Fig. 4.4 is due to the nonzero ramp rate of the load at 0.02 p.u. per second. Another dip at 50-60 seconds is due to the ramping down of the load at a rate of 0.01 p.u. per second. This is expected as the dynamic estimator assumes a distribution of the ramp rate to have a mean of 0 p.u..

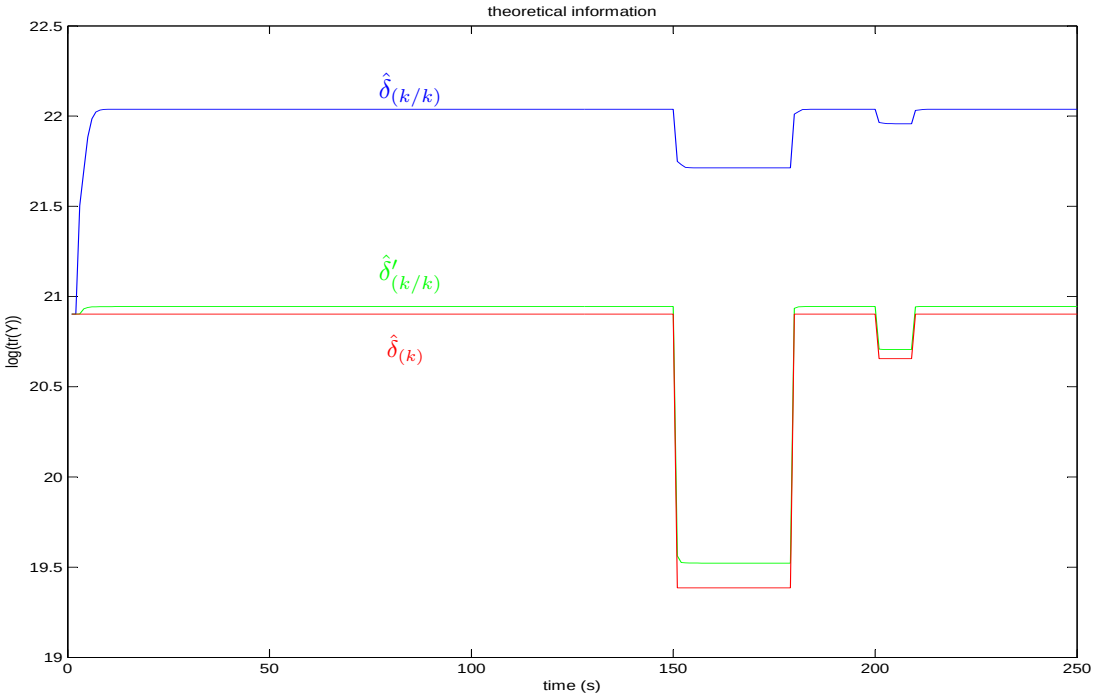


Figure 4.2: Expected value of tr($\mathbf{Y}$) applied to IEEE 14-bus test system: Linear test data, normal load.

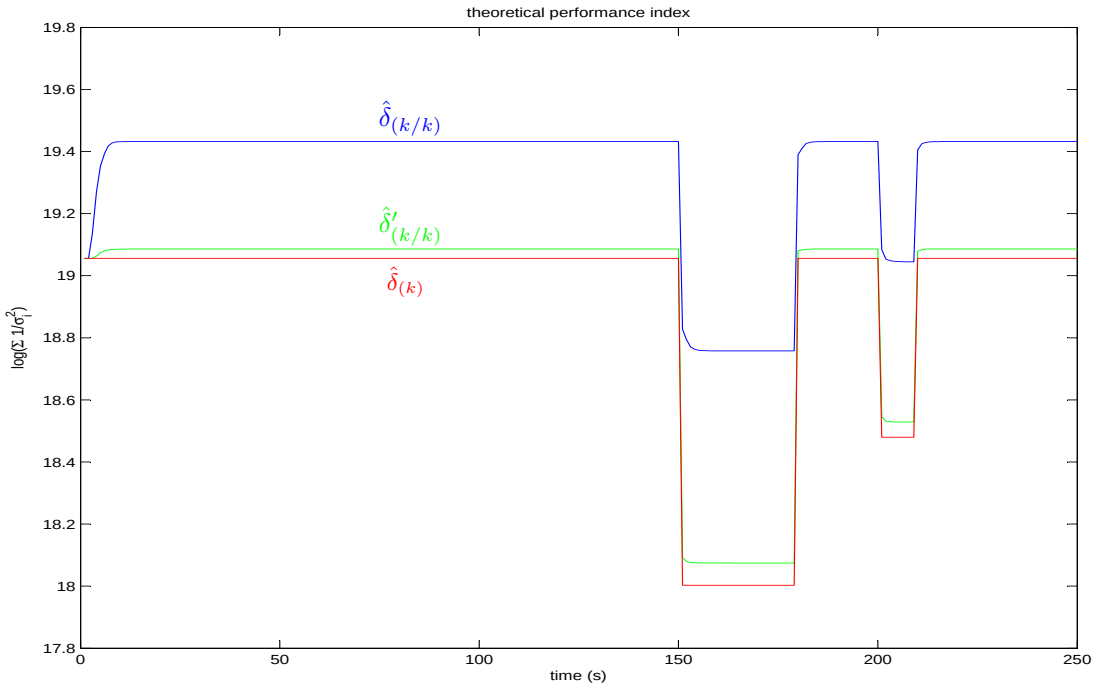


Figure 4.3: Expected value of $\sum 1/\sigma_i$ applied to IEEE 14-bus test system: Linear test data, normal load.

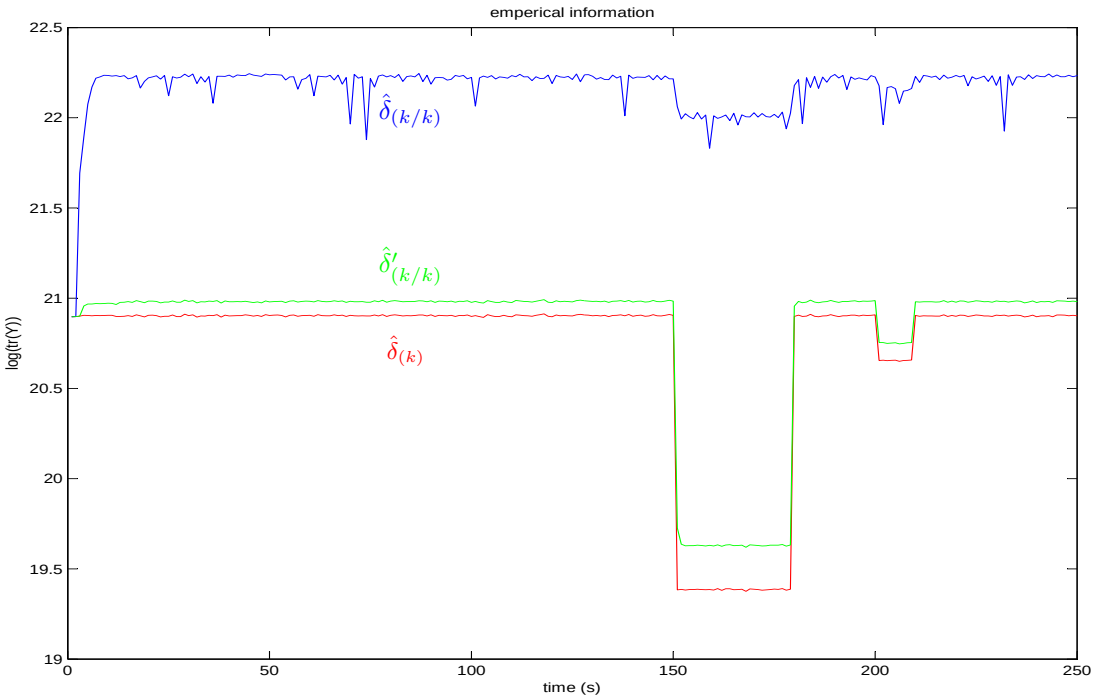


Figure 4.4: Experimental value of $\mathrm{tr}(\mathbf{Y})$ applied to IEEE 14-bus test system: Linear test data, normal load.

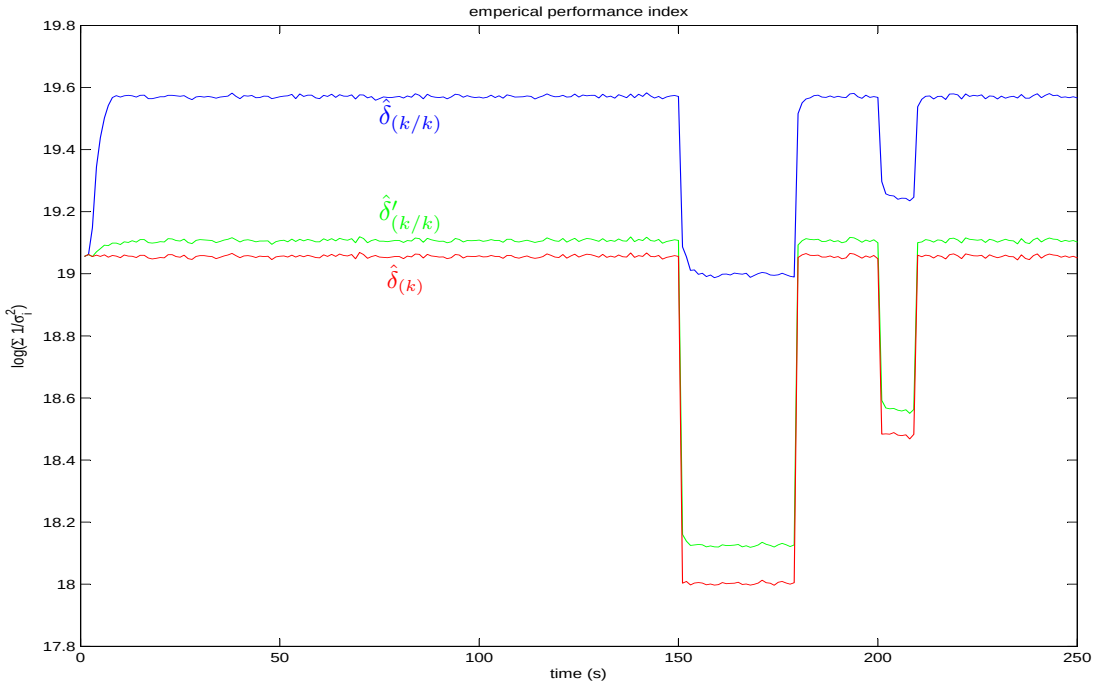


Figure 4.5: Experimental value of $\sum 1/\sigma_i$ applied to IEEE 14-bus test system: Linear test data, normal load.

5. The trace of the information matrix has a larger overall value than the sum of the inverse variances. This makes sense as the trace of the information matrix also incorporates information gathered from the cross correlation of the state variables instead of only the state variable with itself.

6. The sum of inverse variances is more sensitive to reductions in the measurement vector and more accurately indicates if increased errors of individual elements of the state estimate.

4.3.2 Network State Estimation Performance Under Heavy Load : 14-Bus Test System

The network state estimation algorithms were also evaluated under heavier loading conditions. The load transient was increased in magnitude by a factor of ten to give the load transient shown in Fig. 4.6.

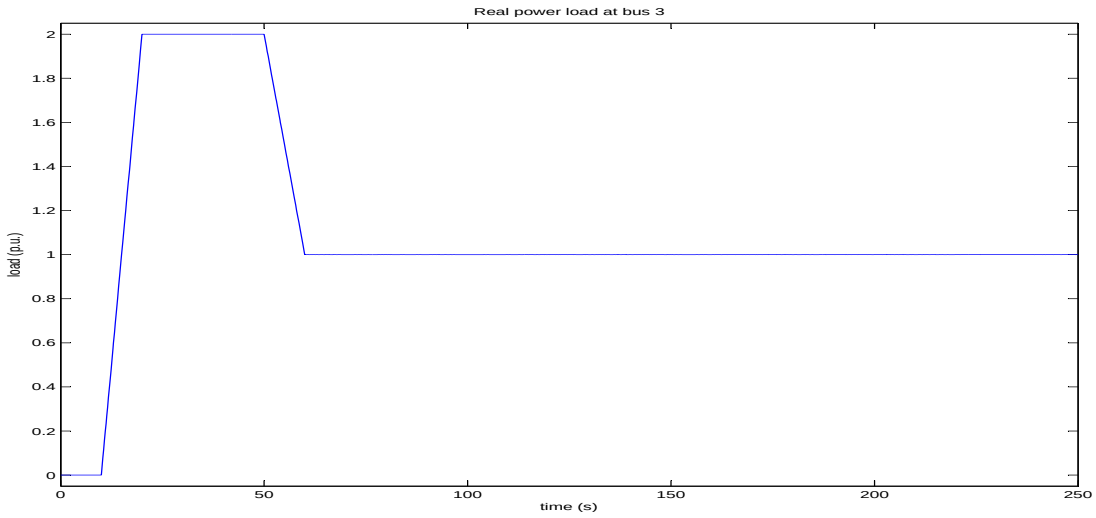


Figure 4.6: Heavy load transient (magnitude increased 10×) for simulation.

Figure 4.7 shows the expected value of tr($\mathbf{Y}$) and Fig. 4.8 shows the expected value of $\sum 1/\sigma_i$ under the heavy loading conditions.

The estimate $\hat{\delta}'_{(k/k)}$ relies on an estimate of the maximum load ramp rate to determine the optimal weighting between the dynamic network state prediction and the static network state estimate. The heavy load scenario has a ramp rate that is ten times that in the normal load scenario. This ramp rate is factored into the Kalman filter as a larger covariance of the load noise. This heavier covariance leads to a lighter weighting of the dynamic network state prediction and a heavier weighting of the static network state estimate. Thus Figs. 4.7 and 4.8 show that the improvements of $\hat{\delta}'_{(k/k)}$ over $\hat{\delta}_{(k)}$ are reduced. Conversely, in situations where the ramp rate is very small, the improvement of $\hat{\delta}'_{(k/k)}$ over $\hat{\delta}_{(k)}$ is expected to be larger.

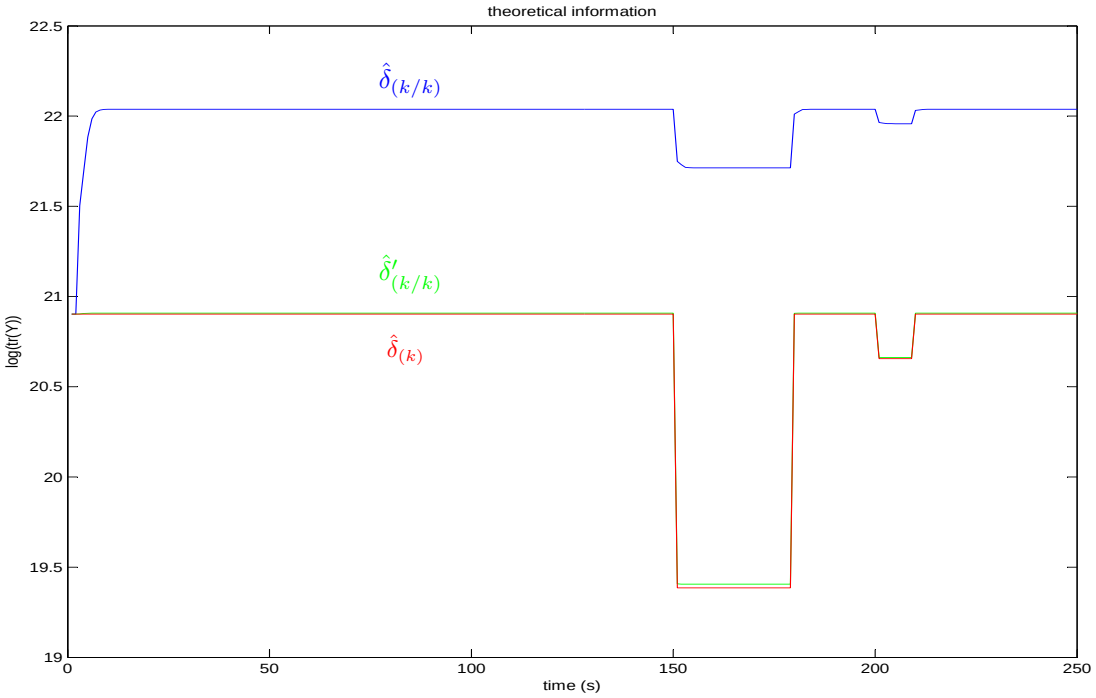


Figure 4.7: Expected value of tr($\mathbf{Y}$) applied to IEEE 14-bus test system: Linear test data, heavy load.

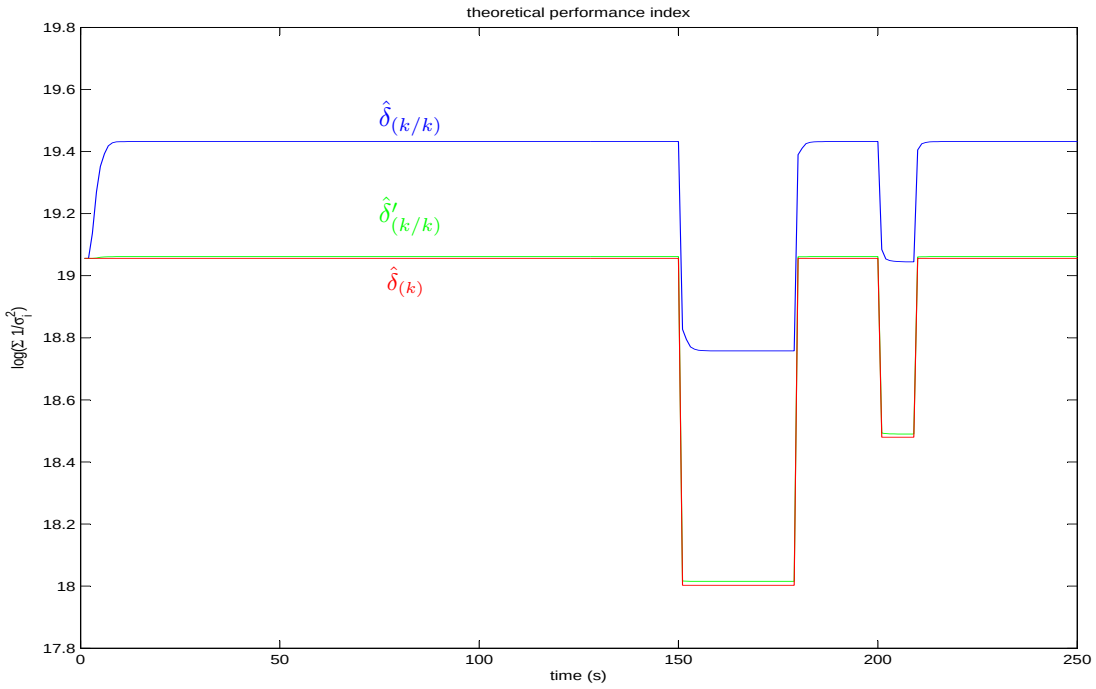


Figure 4.8: Expected value of $\sum 1/\sigma_i$ applied to IEEE 14-bus test system: Linear test data, heavy load.

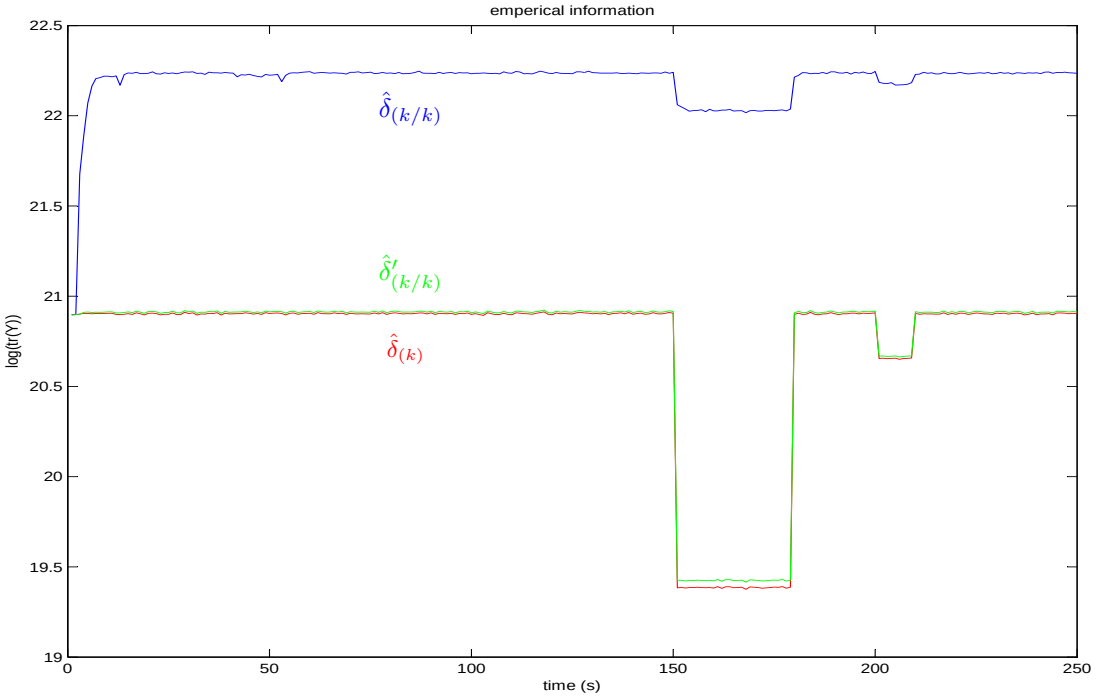


Figure 4.9: Experimental value of tr($\mathbf{Y}$) applied to IEEE 14-bus test system: Linear test data, heavy load.

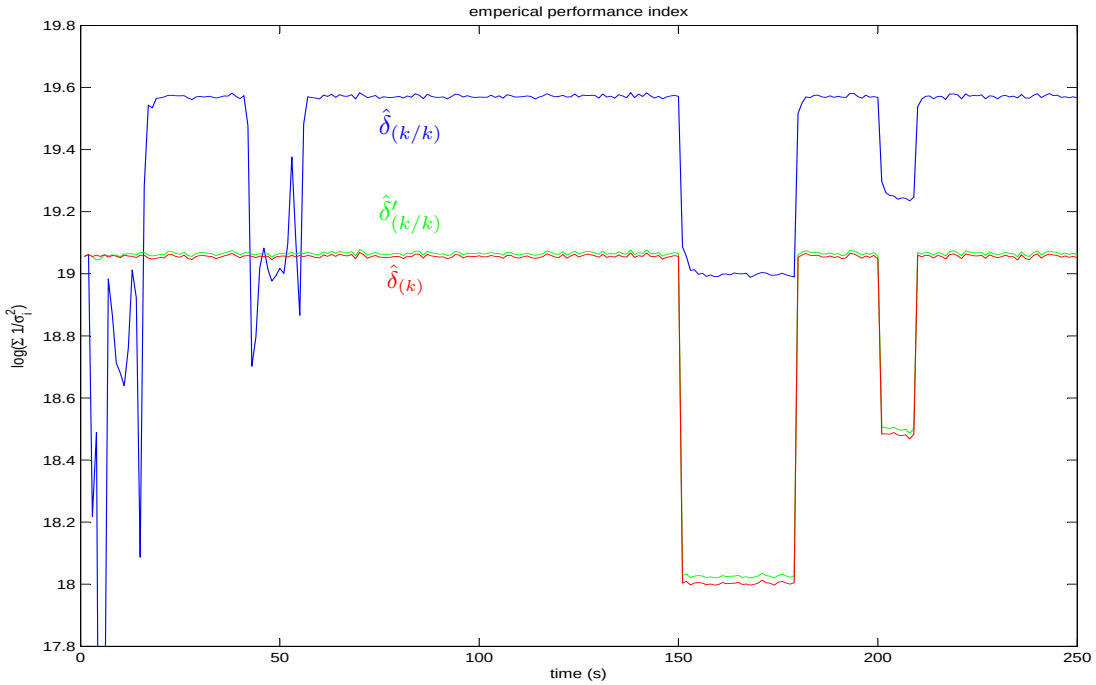


Figure 4.10: Experimental value of $\sum 1/\sigma_i$ applied to IEEE 14-bus test system: Linear test data, heavy load.

Figure 4.9 shows the empirical value of tr($\mathbf{Y}$) and Fig. 4.10 shows the empirical value of $\sum 1/\sigma_i$ under heavy load conditions. Additional error is apparent due to the steeper ramp rate of the load, however we would typically expect $\hat{\delta}_{(k/k)}$ to still track just as well since it is getting measurements of the input. To understand why it is performing poorly we need to look back to Sec. 3.3 to see how the continuous time model is discretized. The discretization of the continuous time dynamic model assumes that the input is constant over the period between samples. As can be seen from the transient in Fig. 4.6 this is not the case.

To see what the performance would be if this assumption were true, the 14-bus network is simulated with the input shown in Fig. 4.11 where a zero-order-hold is applied to the input so that the assumption is valid.

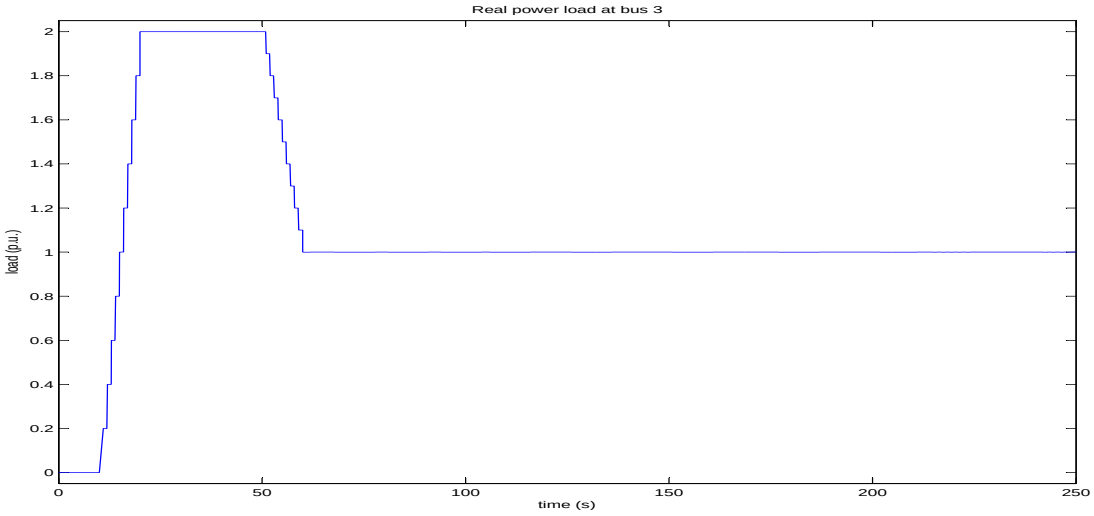


Figure 4.11: Stepwise heavy load transient for simulation.

Figure 4.12 shows the empirical value of tr($\mathbf{Y}$) and Fig. 4.13 shows the empirical value of $\sum 1/\sigma_i$ under heavy stepwise (with zero-order-hold applied) load conditions. Here we can see in Figs. 4.12 and 4.13 that the drop in experimental performance shown in Figs. 4.9 and 4.10 is gone now that the discretization assumptions are true.

Comparatively, if we look at the performance with a transient with one tenth the magnitude we see the following results. Figure 4.14 shows the expected value of tr($\mathbf{Y}$) and Fig. 4.15 shows the expected value of $\sum 1/\sigma_i$ under heavy light load conditions.

Figure 4.16 shows the empirical value of tr($\mathbf{Y}$) and Fig. 4.17 shows the empirical value of $\sum 1/\sigma_i$ under light load conditions. Here we see that the expected and experimental performance of $\hat{\delta}'_{(k/k)}$ shows a more significant improvement over $\hat{\delta}_{(k)}$. This improvement is possible with both the expected and actual ramp rate of the bus loads are small. The performance $\hat{\delta}_{(k/k)}$ shows a small improvement due to the effects of converting the model to discrete time and the performance of $\hat{\delta}_{(k)}$ remain relatively compared to the normal

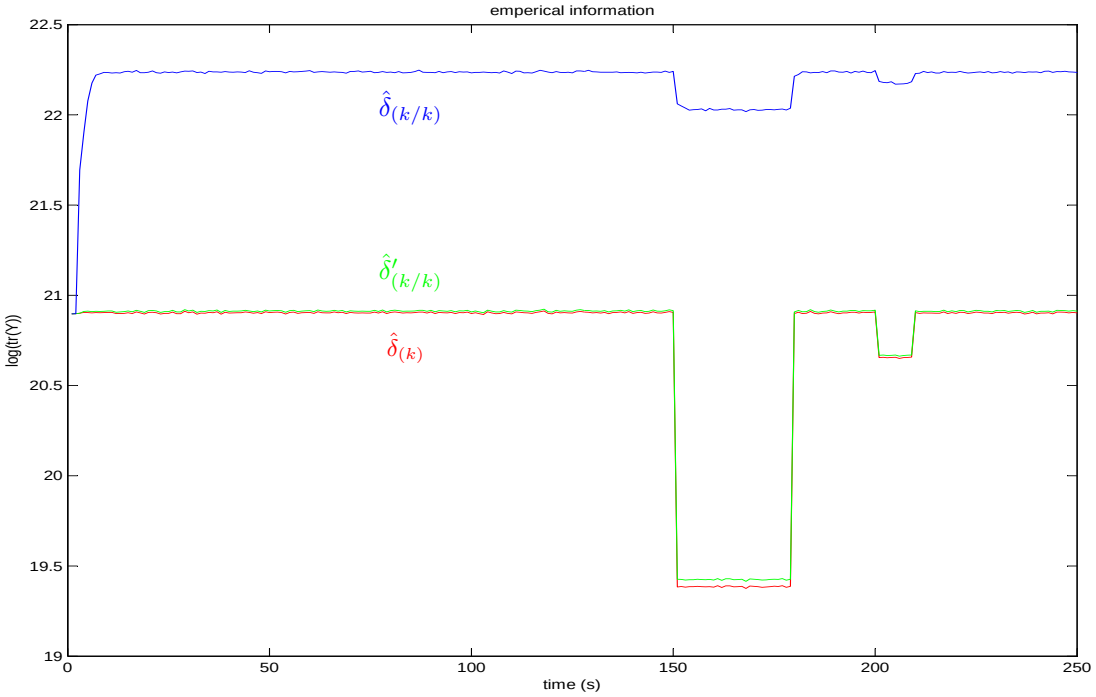


Figure 4.12: Experimental value of tr($\mathbf{Y}$) applied to IEEE 14-bus test system: Linear test data, heavy stepwise load.

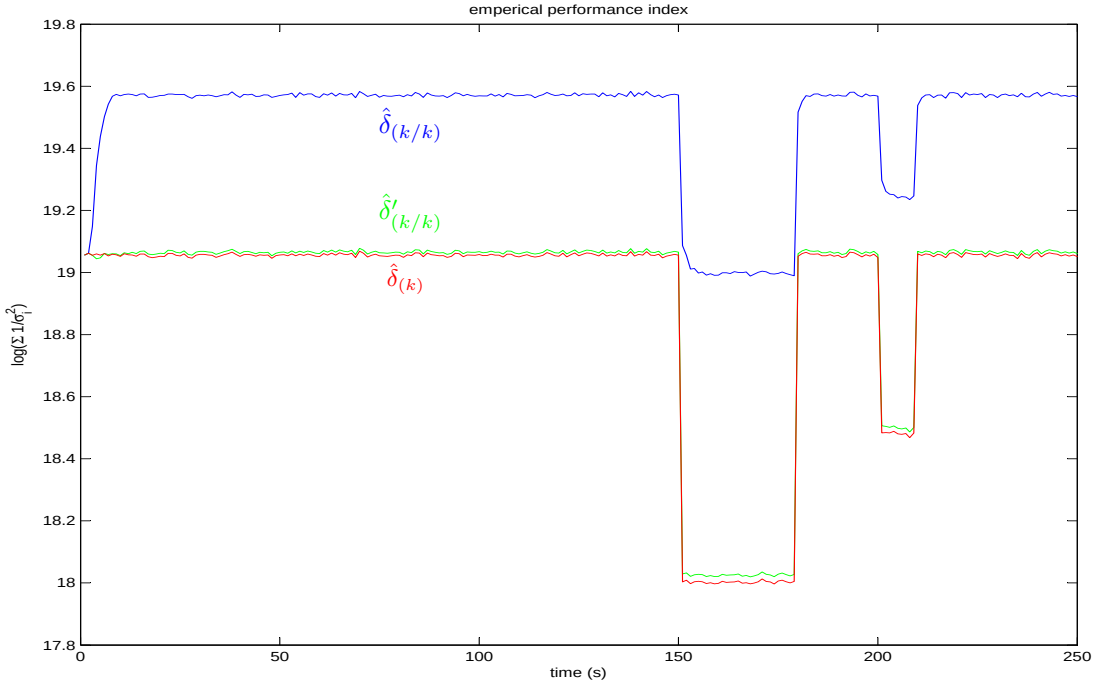


Figure 4.13: Experimental value of $\sum 1/\sigma_i$ applied to IEEE 14-bus test system: Linear test data, heavy stepwise load.

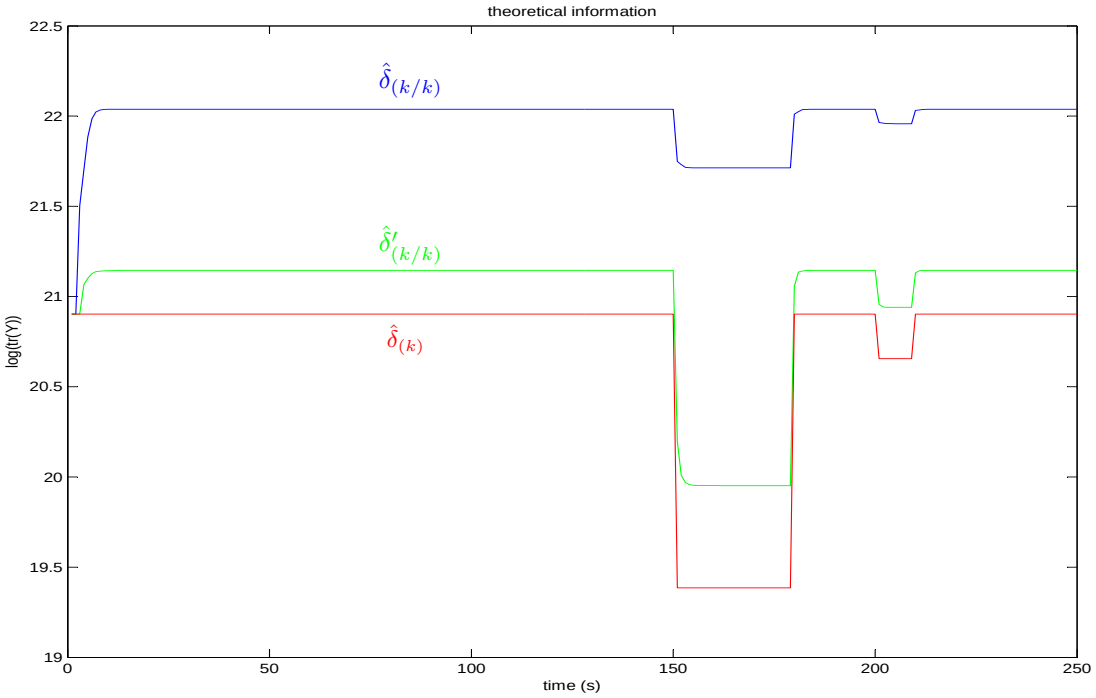


Figure 4.14: Expected value of tr($\mathbf{Y}$) applied to IEEE 14-bus test system: Linear test data, light load.

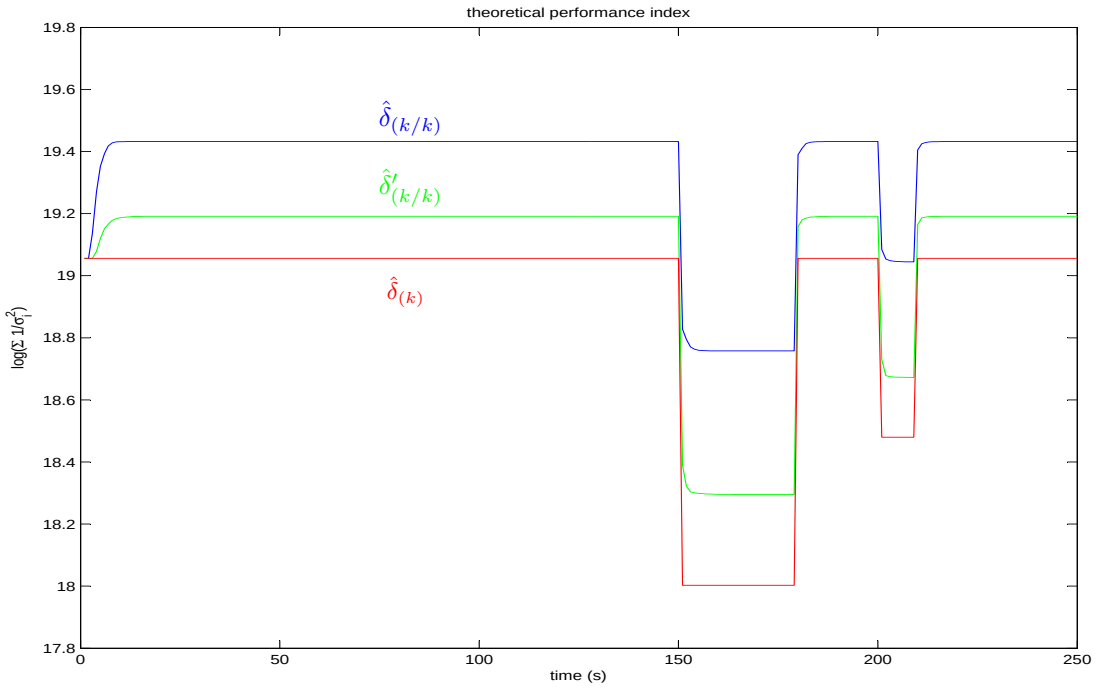


Figure 4.15: Expected value of $\sum 1/\sigma_i$ applied to IEEE 14-bus test system: Linear test data, light load.

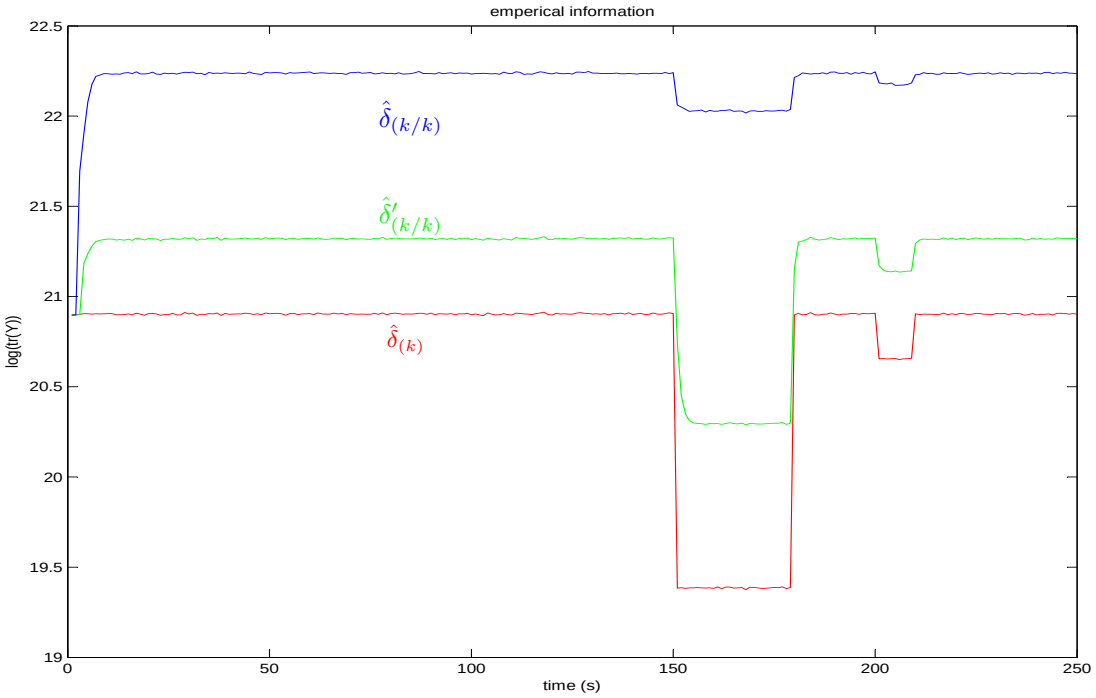


Figure 4.16: Experimental value of tr($\mathbf{Y}$) applied to IEEE 14-bus test system: Linear test data, light load.

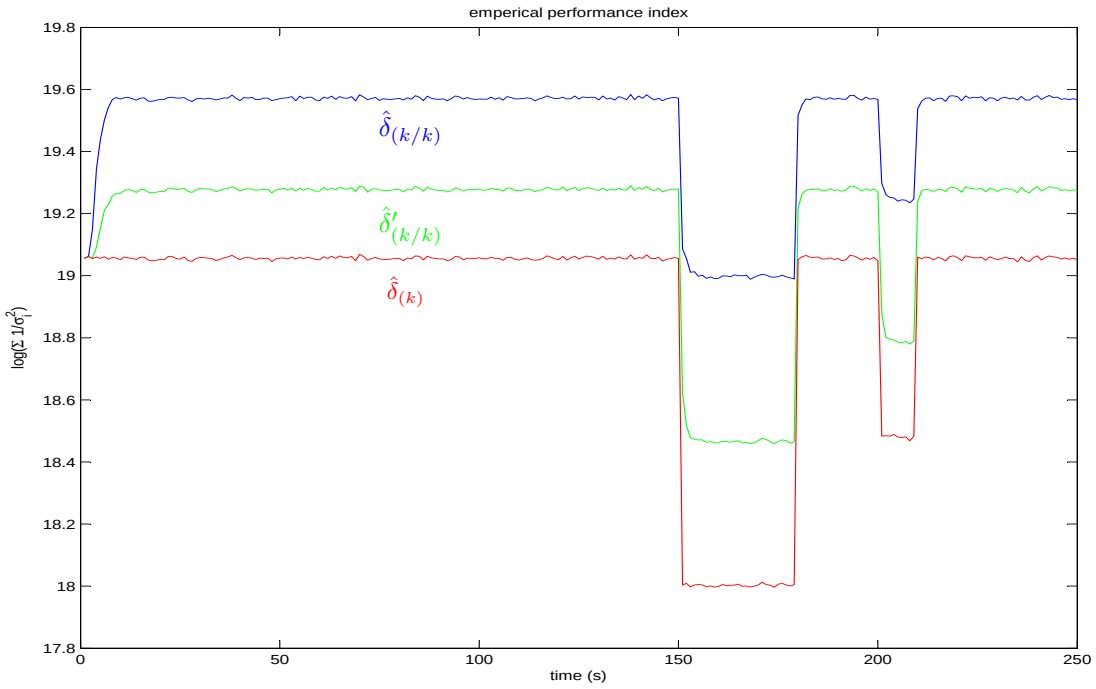


Figure 4.17: Experimental value of $\sum 1/\sigma_i$ applied to IEEE 14-bus test system: Linear test data, light load.

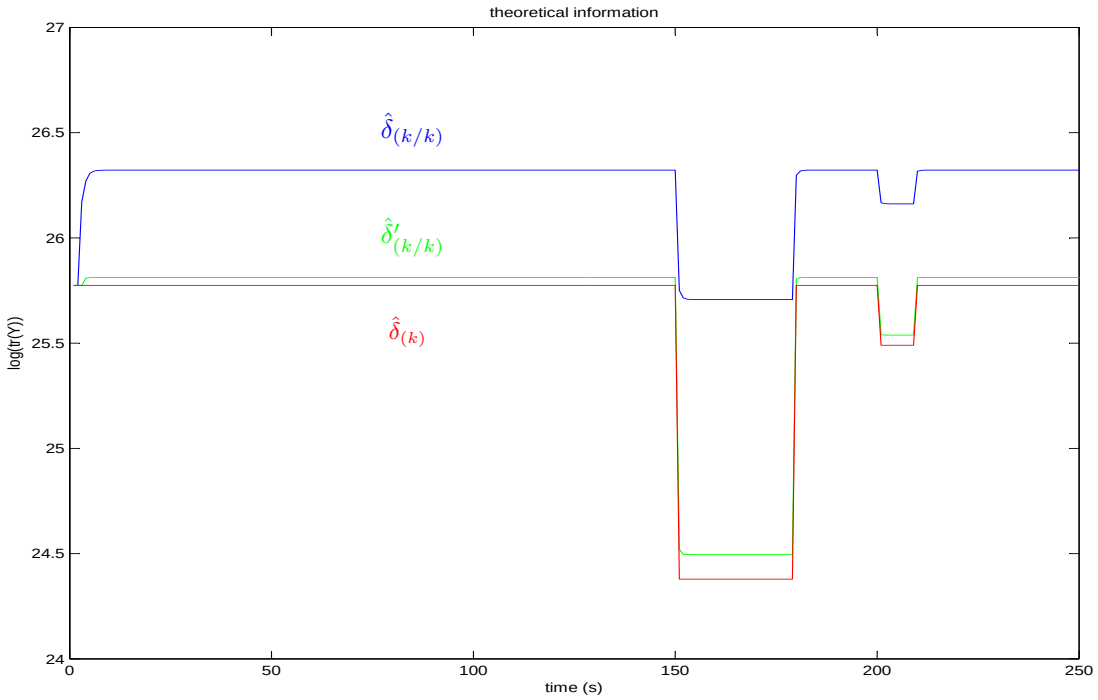


Figure 4.18: Expected value of tr($\mathbf{Y}$) applied to IEEE 118-bus test system: Nonlinear test data, normal load.

loading conditions.

4.3.3 Network State Estimation Performance : 118-Bus Test System

Figure 4.18 shows the expected value of tr($\mathbf{Y}$) and Fig. 4.19 shows the expected value of $\sum 1/\sigma_i$ under normal load conditions for nonlinear test data. Figure 4.20 shows the empirical value of tr($\mathbf{Y}$) and Fig. 4.21 shows the empirical value of $\sum 1/\sigma_i$ based on a Monte Carlo simulation of the IEEE 118-bus test system simulated over 1,000 passes with random noise corrupting the measurements and measurements of the physical bus loads (inputs).

A similar dip in performance of $\sum 1/\sigma_i$ during the load transient is observed in these simulations, although the degradation in $\hat{\delta}_{(k/k)}$ is observed at normal loads which were only significant with heavy loads on the IEEE 14-bus system. As the error in the 14-bus system was due to discretization error, the performance with a stepwise load was also evaluated for normal loads on the 118-bus test system.

Figure 4.22 shows the empirical value of tr($\mathbf{Y}$) and Fig. 4.23 shows the empirical value of $\sum 1/\sigma_i$ with the stepwise instead of the continuous load transient. As with the 14-bus system, the degradation disappears indicating that the degradation was due to the discretization and not the increased size of the test system.

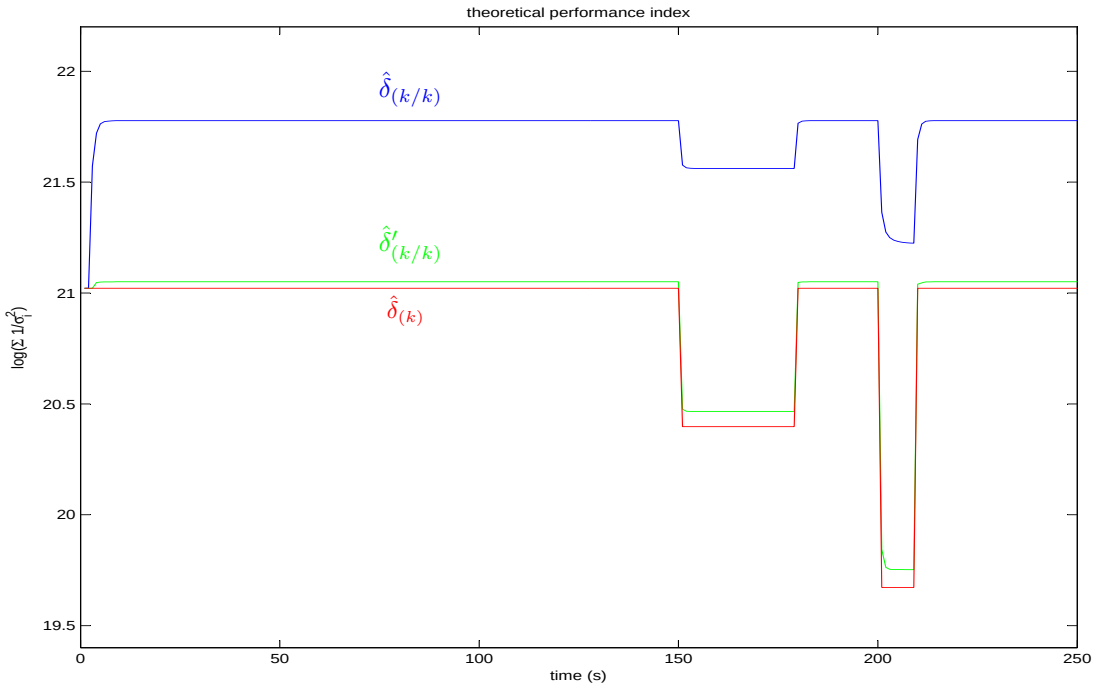


Figure 4.19: Expected value of $\sum 1/\sigma_i$ applied to IEEE 118-bus test system: Nonlinear test data, normal load.

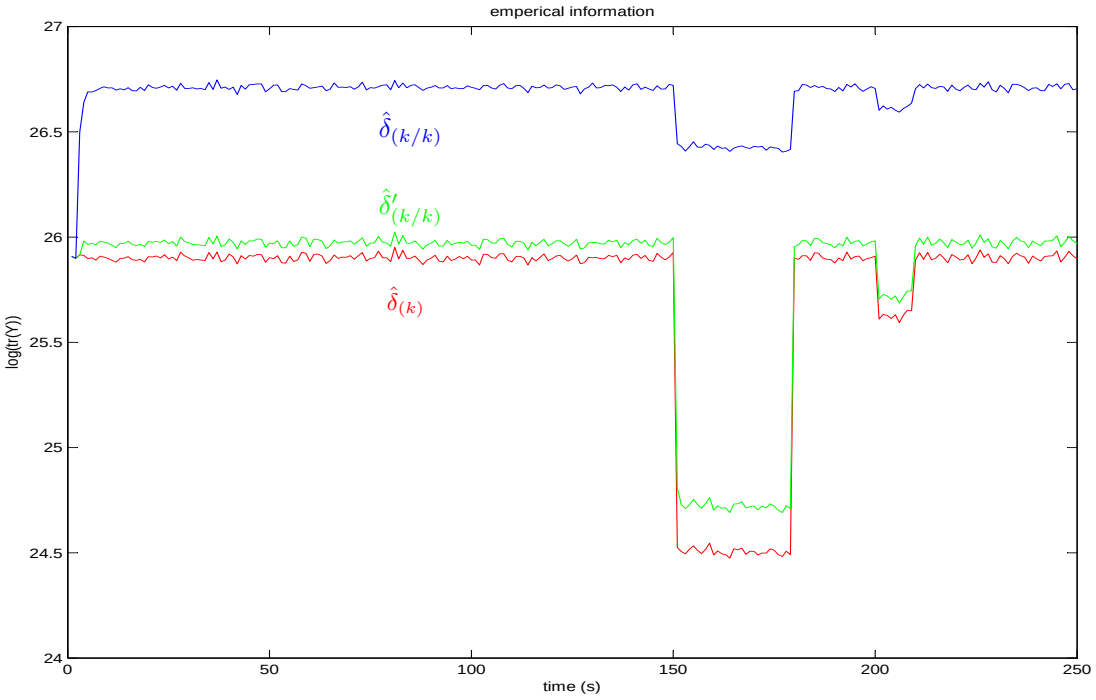


Figure 4.20: Experimental value of tr($\mathbf{Y}$) applied to IEEE 118-bus test system: Linear test data, normal load.

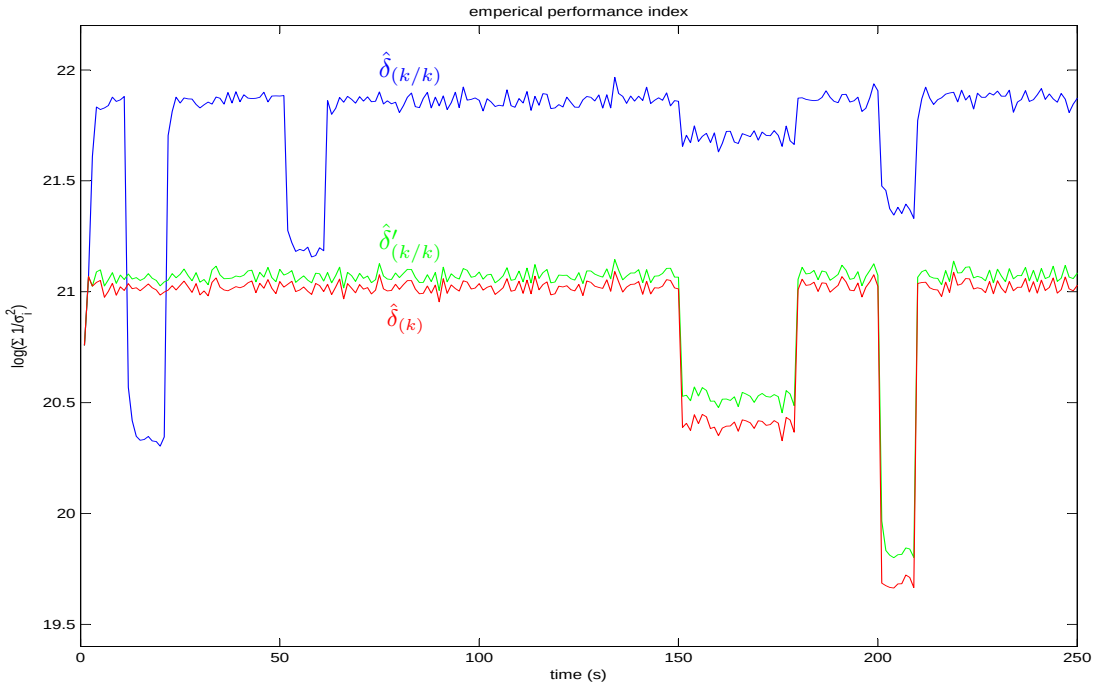


Figure 4.21: Experimental value of $\sum 1/\sigma_i$ applied to IEEE 118-bus test system: Linear test data, normal load.

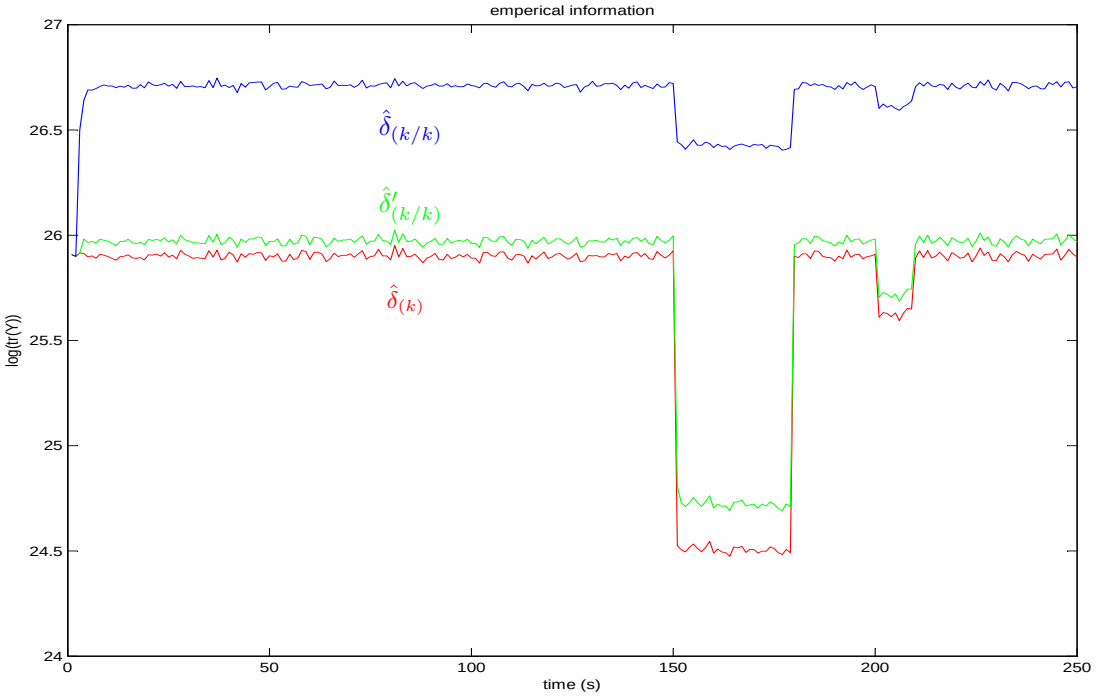


Figure 4.22: Experimental value of tr($\mathbf{Y}$) applied to IEEE 118-bus test system: Linear test data, normal stepwise load.

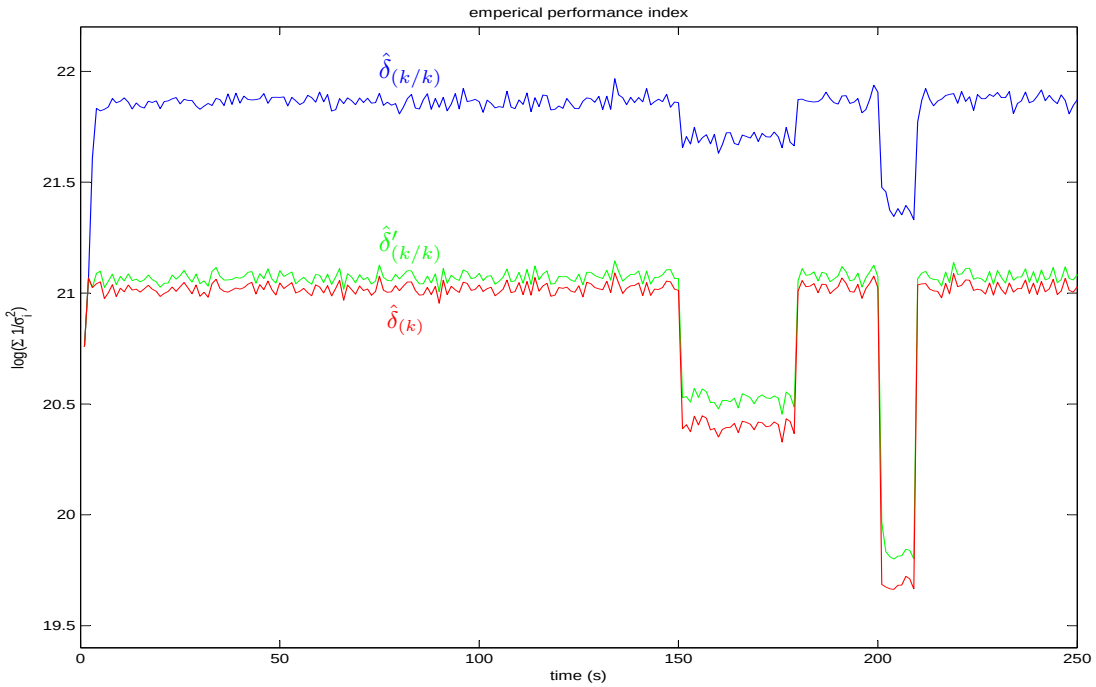


Figure 4.23: Experimental value of $\sum 1/\sigma_i$ applied to IEEE 118-bus test system: Linear test data, normal stepwise load.

4.3.4 Network State Estimation Performance Under Heavy Load : 118-Bus Test System

Figure 4.24 shows the empirical value of tr($\mathbf{Y}$) and Fig. 4.25 shows the empirical value of $\sum 1/\sigma_i$ based on a Monte Carlo simulation of the IEEE 118-bus test under heavy loading conditions. As expected, the degradation in performance of $\sum 1/\sigma_i$ during the transient is observed.

Figure 4.26 shows the empirical value of tr($\mathbf{Y}$) and Fig. 4.27 shows the empirical value of $\sum 1/\sigma_i$ based on a Monte Carlo simulation of the IEEE 118-bus test under heavy stepwise loading conditions. The degradation in performance of $\sum 1/\sigma_i$ during the transient once again disappears when the zero-order-hold assumption of the model discretization process is met.

4.4 Evaluation of Algorithms using Nonlinear Test Data

In this section the same transients analyzed in Sec. are repeated but without using the linearized model to create test data. The assumption of lossless lines is still applied, but the trigonometric small angle approximation as described in Sec. 2.2 was not used. The formulation of the network state estimators is the same so only the empirical data is presented as the expected values of the performance metrics are

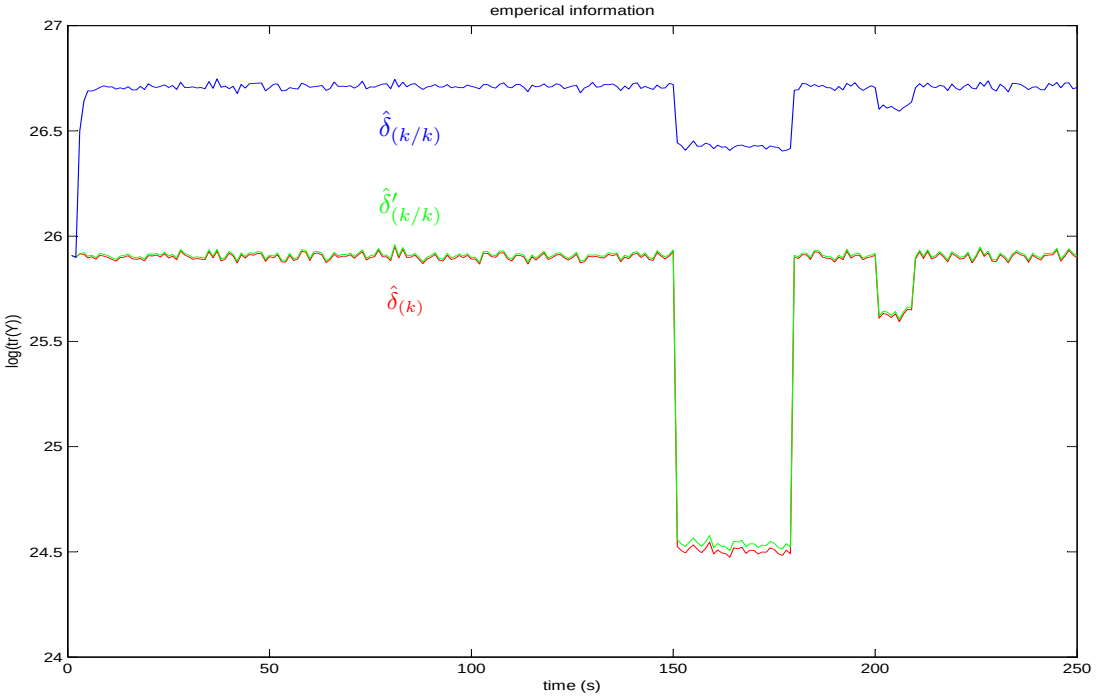


Figure 4.24: Experimental value of tr($\mathbf{Y}$) applied to IEEE 118-bus test system: Linear test data, heavy load.

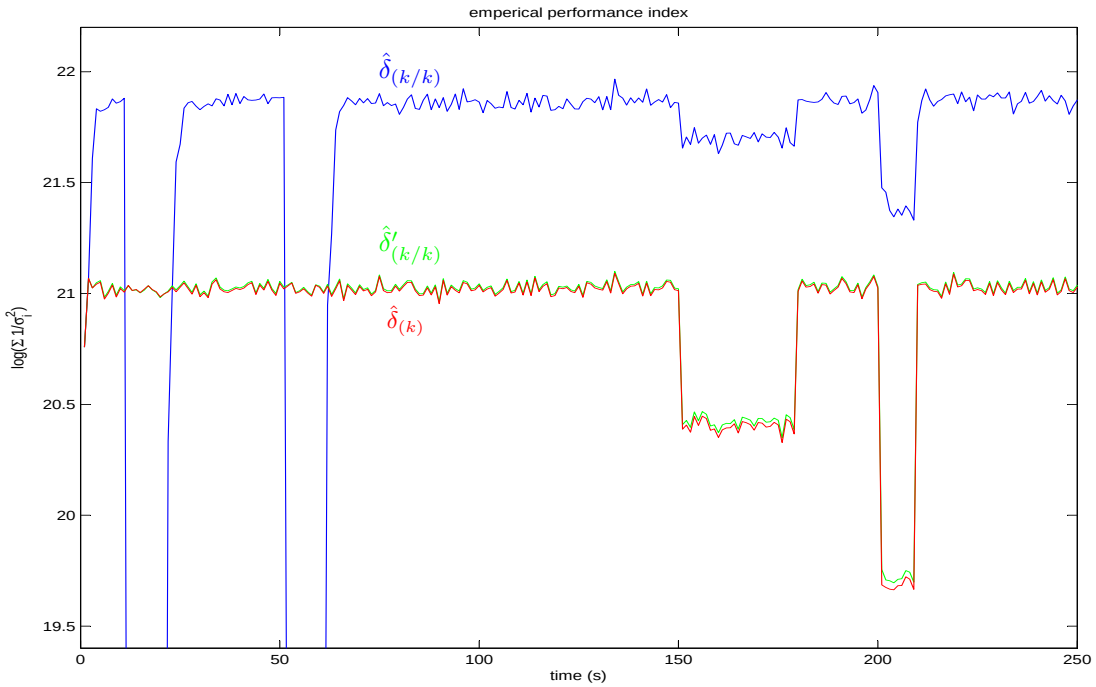


Figure 4.25: Experimental value of $\sum 1/\sigma_i$ applied to IEEE 118-bus test system: Linear test data, heavy load.

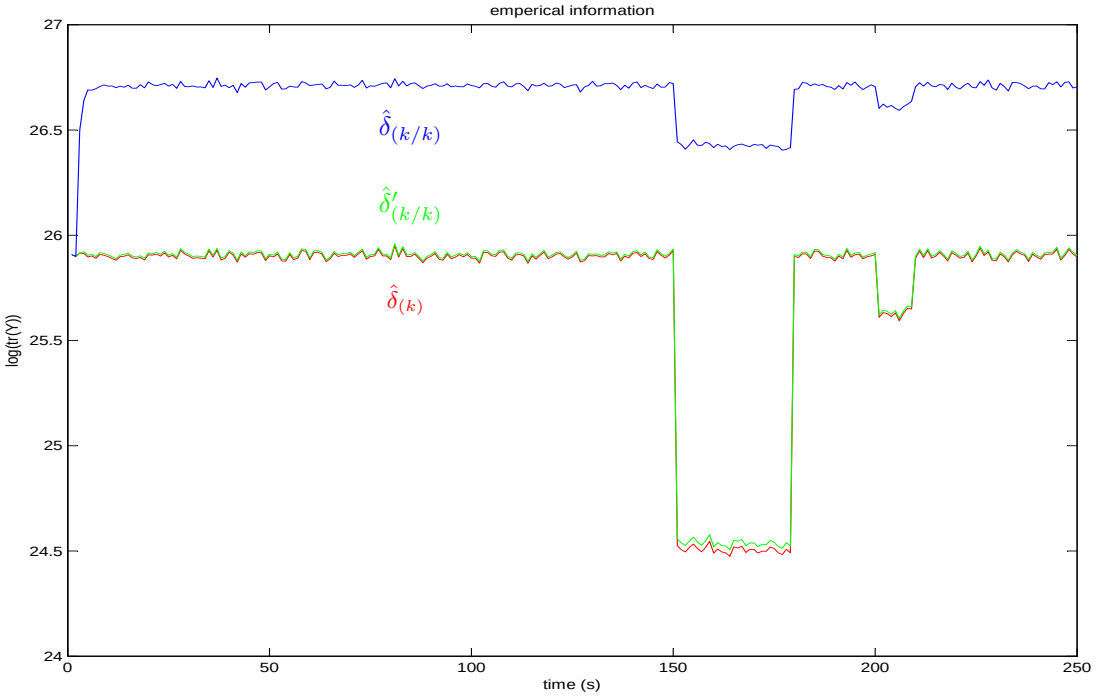


Figure 4.26: Experimental value of tr($\mathbf{Y}$) applied to IEEE 118-bus test system: Linear test data, heavy stepwise load.

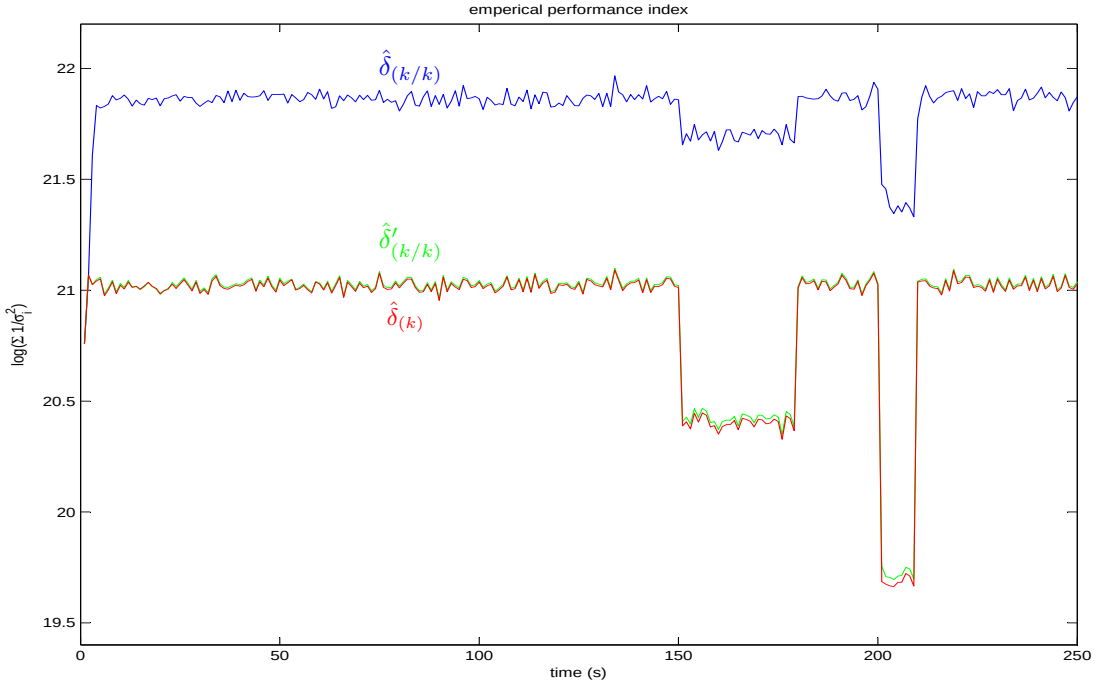


Figure 4.27: Experimental value of $\sum 1/\sigma_i$ applied to IEEE 118-bus test system: Linear test data, heavy stepwise load.

unchanged.

4.4.1 Network State Estimation Performance : 14-Bus Test System

Figure 4.28 shows the empirical value of tr($\mathbf{Y}$) and Fig. 4.29 shows the empirical value of $\sum 1/\sigma_i$ under normal load conditions for nonlinear test data. The same performance trends were observed between the three network state estimators as was observed with linear test data. Some small degradations in tr($\mathbf{Y}$) were noted for $\hat{\delta}_{(k/k)}$, however the performance remained significantly above that of $\hat{\delta}_{(k)}$ or $\hat{\delta}'_{(k/k)}$. No other significant changes in performance as compared to linear test data were observable.

4.4.2 Network State Estimation Performance : 118-Bus Test System

Figure 4.30 shows the empirical value of tr($\mathbf{Y}$) and Fig. 4.31 shows the empirical value of $\sum 1/\sigma_i$ under normal load conditions for nonlinear test data. The same performance trends were observed between the three network state estimators as was observed with linear test data. No significant changes in performance as compared to the linear test data were observable.

4.5 Evaluation of Dynamic Network State Estimation Algorithms Using a Reduced Order Dynamic Model

To better understand the detail to which dynamic modeling must be accomplished in order to achieve accurate dynamic network state prediction, the existing dynamic model for the IEEE 14-bus and 118-bus test systems were simulated under varying degrees of model reduction. To reduce the model, singular value decomposition (MATLAB command `SVD`) was used to reduce the dynamic state vector by a given number of state variables. The IEEE 14-bus test system has 13 elements of the network state and 31 elements of the dynamic state for $\hat{\delta}_{(k/k)}$ and 31+14=45 elements of the dynamic state for $\hat{\delta}'_{(k/k)}$. The IEEE 118-bus test system has 117 elements of the network state and 239 elements of the dynamic state for $\hat{\delta}_{(k/k)}$ and 239+118=357 elements of the dynamic state for $\hat{\delta}'_{(k/k)}$.

Simulations employing the full model without any reductions in the dynamic state vector are shown in Sec. 4.3.1 for the 14-bus system and Sec. 4.4.2 for the 118-bus system. The expected values of the performance metrics tr($\mathbf{Y}$) and $\sum 1/\sigma_i$ respectively are shown in Figs. 4.2 and 4.3 for the 14-bus system and Figs. 4.18 and 4.19 for the 118-bus system. The experimental values of the performance metrics are shown in Figs. 4.4 and 4.5 for the 14-bus system and Figs. 4.20 and 4.21 for the 118-bus system.

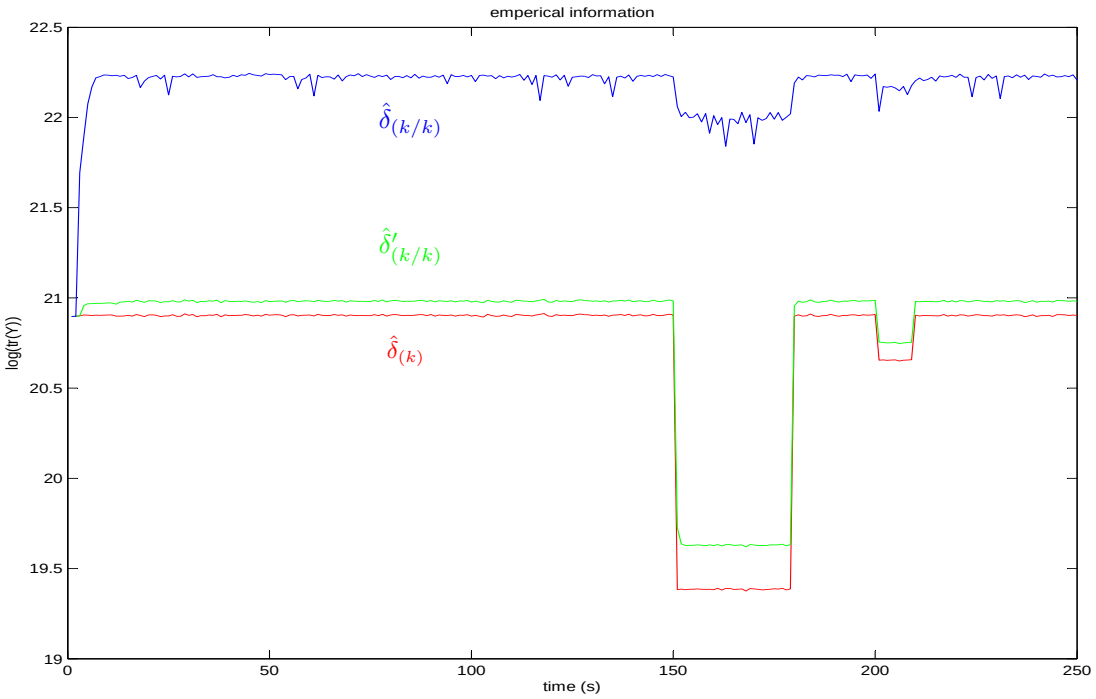


Figure 4.28: Experimental value of $\mathrm{tr}(\mathbf{Y})$ applied to IEEE 14-bus test system: Nonlinear test data, normal load.

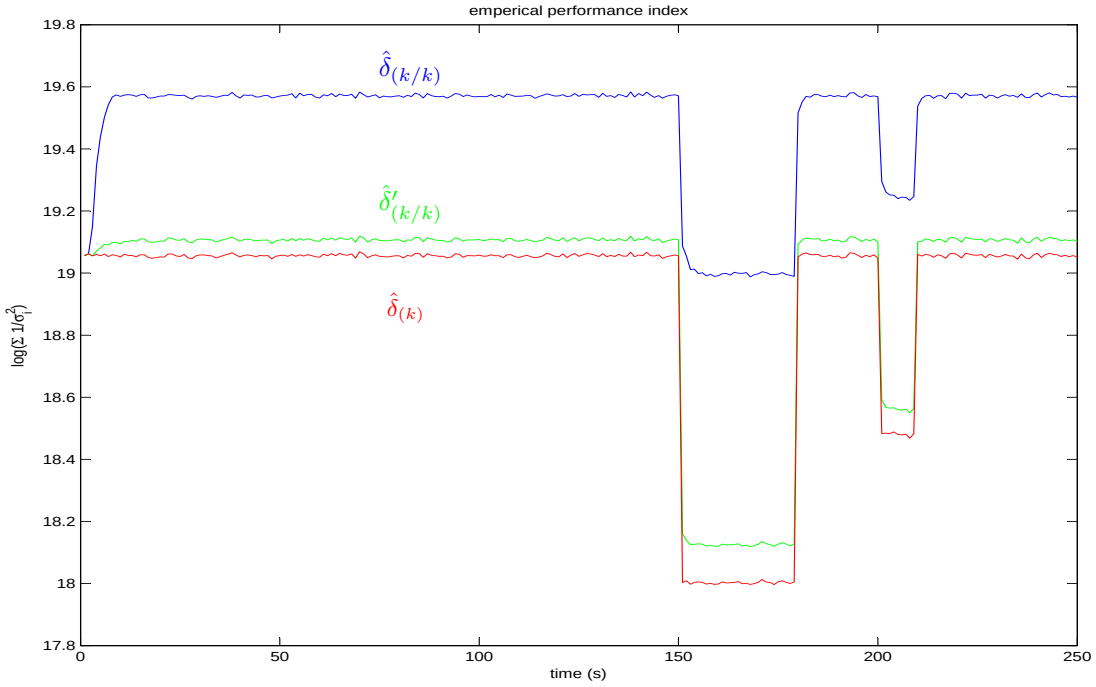


Figure 4.29: Experimental value of $\sum 1/\sigma_i$ applied to IEEE 14-bus test system: Nonlinear test data, normal load.

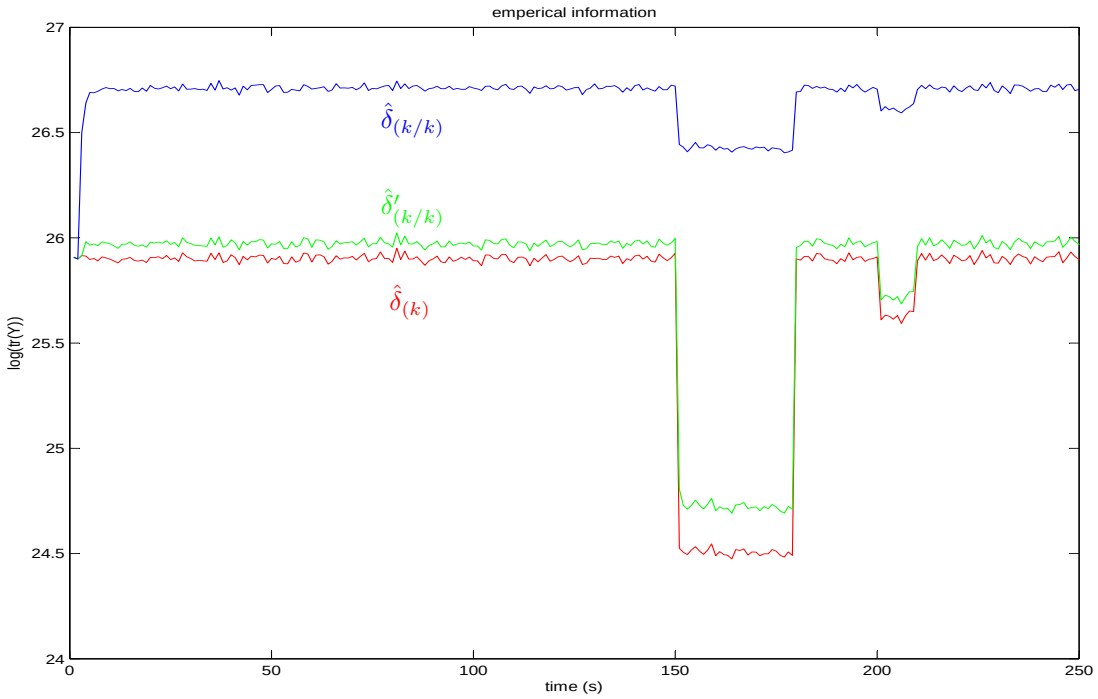


Figure 4.30: Experimental value of $\mathrm{tr}(\mathbf{Y})$ applied to IEEE 118-bus test system: Nonlinear test data, normal load.

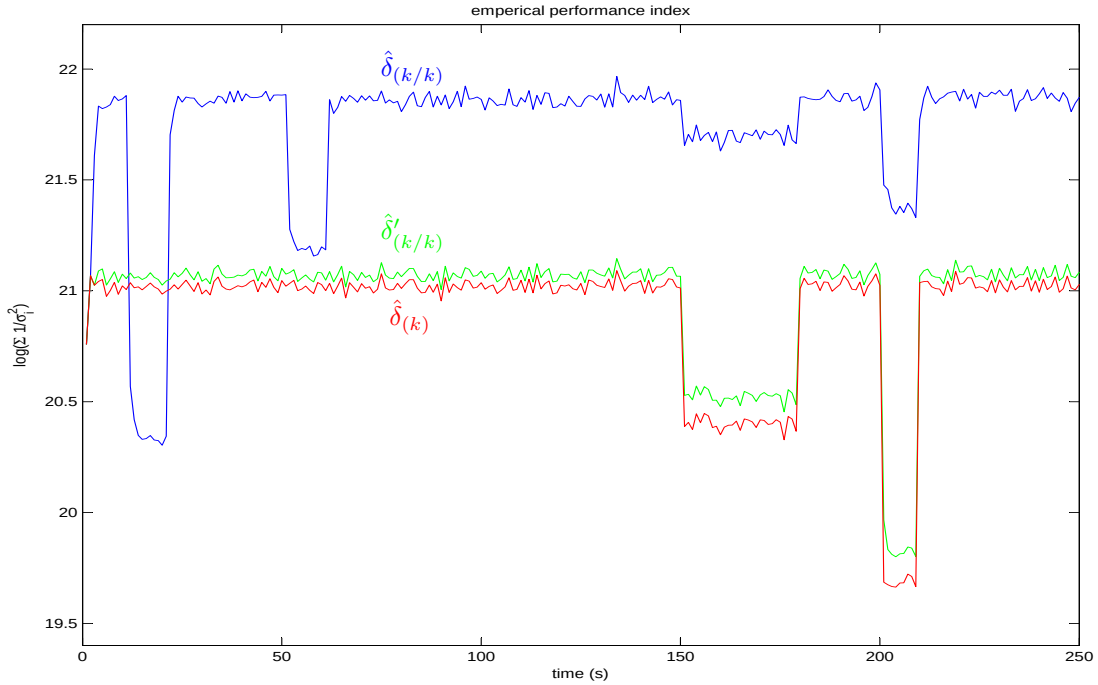


Figure 4.31: Experimental value of $\sum 1/\sigma_i$ applied to IEEE 118-bus test system: Nonlinear test data, normal load.

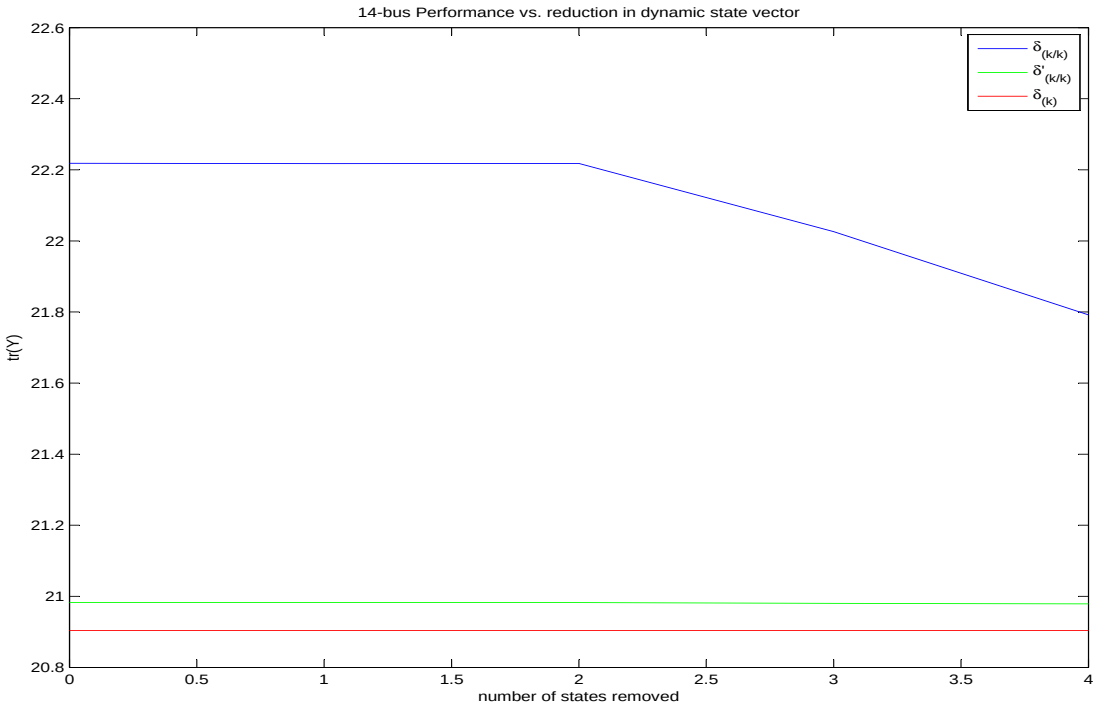


Figure 4.32: Average tr($\mathbf{Y}$) performance with reduced dynamic state modeling : 14-bus

As the expected values do not express the mismatch between reduced and full order modeling, only the experimental results for the model reduction study will be presented. For comparison, we will consider the average performance over the simulated time period from 80 to 150 seconds. During this time period the initial load transient is complete and the full measurement vector is available.

Only minimal changes in the performance of $\hat{\delta}'_{(k/k)}$ are observed and no change in the performance of $\hat{\delta}_{(k)}$ are observed as $\hat{\delta}_{(k)}$ is not affected by the dynamic model. Therefore, the following discussion will focus on the performance of $\hat{\delta}_{(k/k)}$.

Figure 4.32 shows the tr($\mathbf{Y}$) performance metric and Fig. 4.33 shows the $\sum 1/\sigma_i$ performance metric as the state vector is reduced on the 14-bus system. Only minimal degradation in performance is apparent for tr($\mathbf{Y}$) whereas $\sum 1/\sigma_i$ drops significantly with the third dropped state. With a fourth dropped state, the $\sum 1/\sigma_i$ performance drops below that of $\hat{\delta}_{(k)}$.

A potential explanation for this drop is due to the fact that only two generators are modeled, each with four state variables. If we assume that the generator is sufficiently modeled with three state variables, we may shed one state variable from each generator model without incurring significant degradation in performance.

Figure 4.34 shows the tr($\mathbf{Y}$) performance metric and Fig. 4.35 shows the $\sum 1/\sigma_i$ performance metric as the state vector is reduced on the 118-bus system. As expected, the $\sum 1/\sigma_i$ performance metric shows

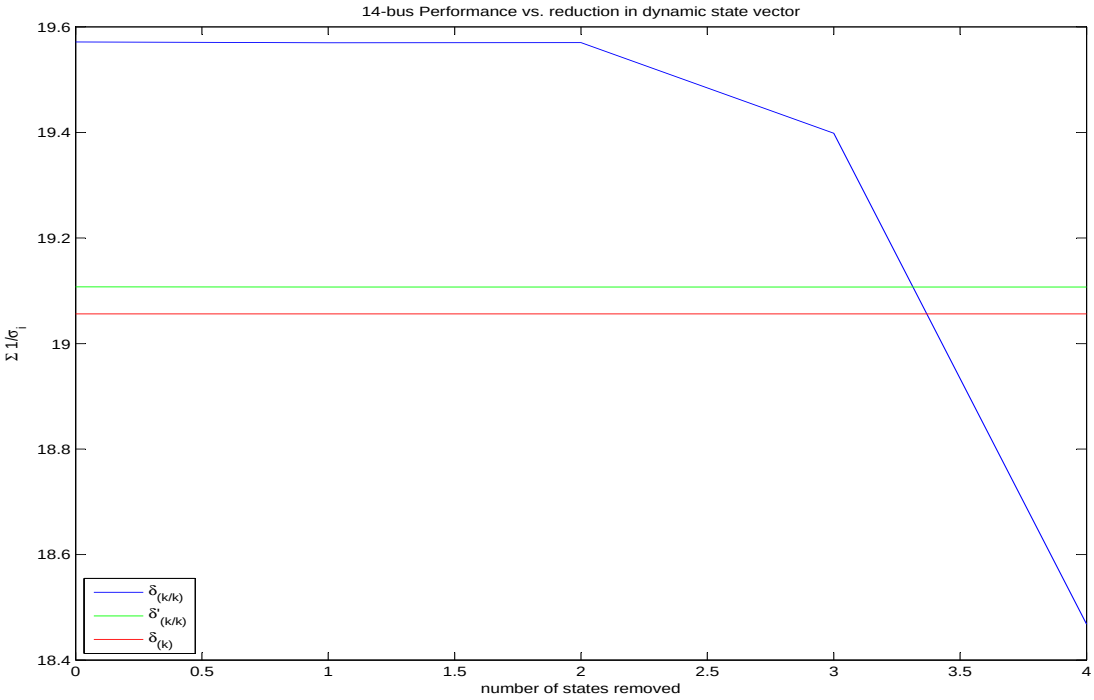


Figure 4.33: Average $\sum 1/\sigma_i$ performance with reduced dynamic state modeling : 14-bus

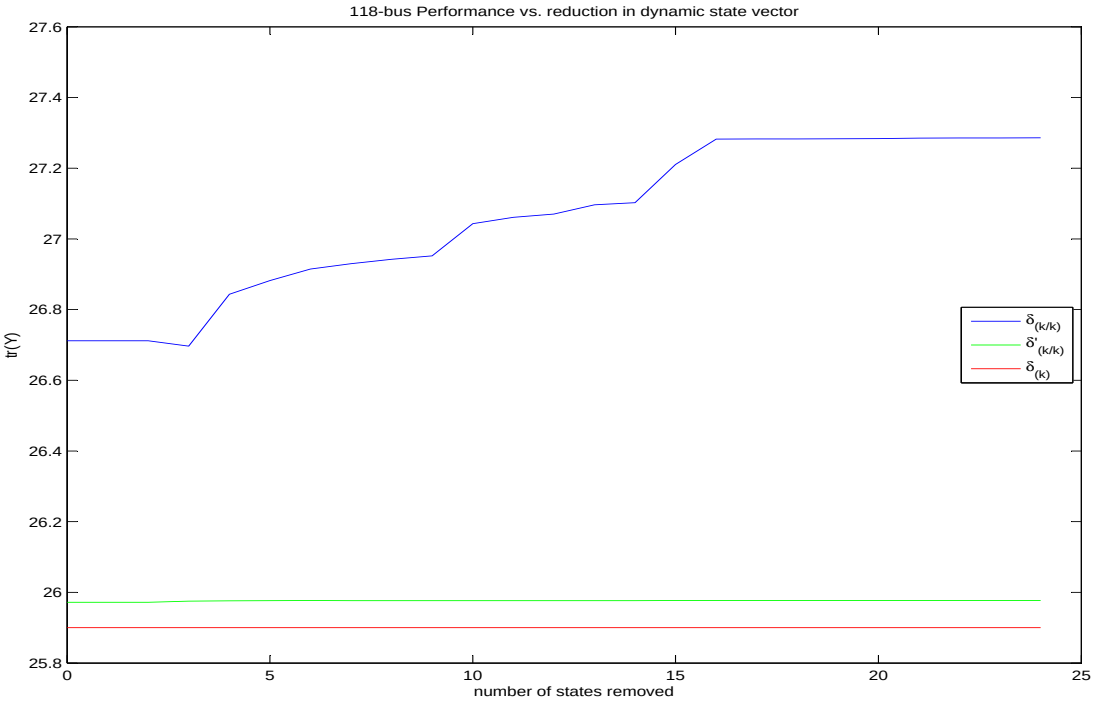


Figure 4.34: Average tr($\mathbf{Y}$) performance with reduced dynamic state modeling : 118-bus

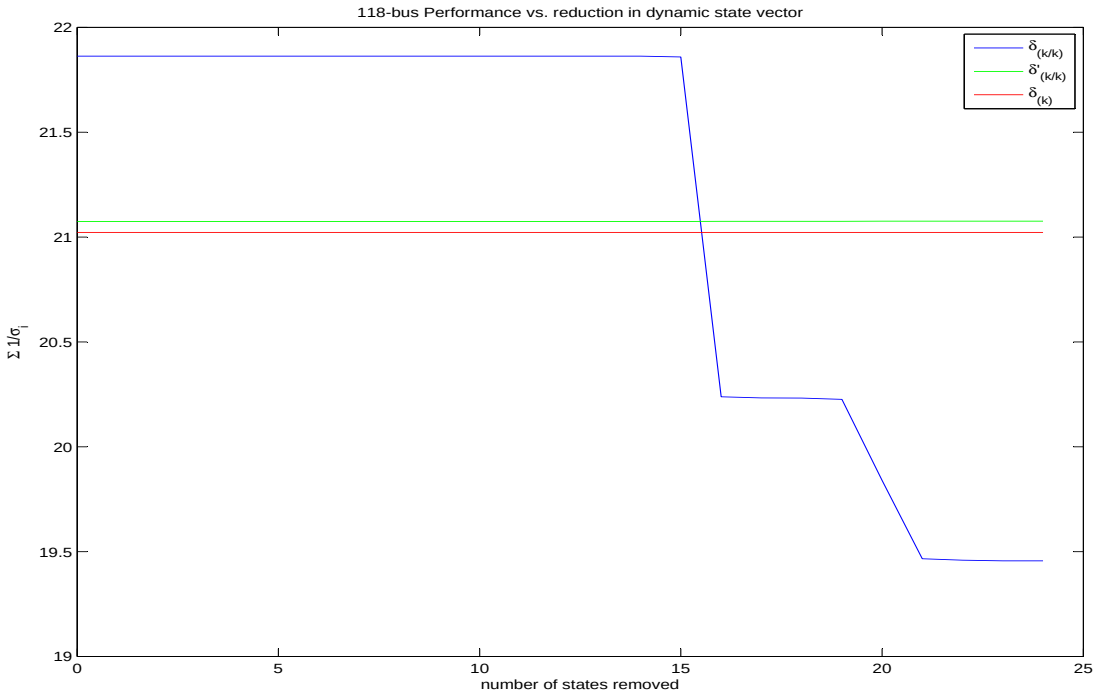


Figure 4.35: Average $\sum 1/\sigma_i$ performance with reduced dynamic state modeling : 118-bus

degradation with increased reduction of the dynamic state vector. The degradation occurs in steps at 16 states and again at 20 and 21.

Similar to the 14-bus system, two generators are modeled. As such, we would normally expect to see significant degradation in performance at the third reduction in state. Also surprisingly, the tr($\mathbf{Y}$) performance metric actually increases with the reductions in the state vector. This is potentially an indication that the network state variables are becoming more highly cross correlated.

4.6 Conclusions

In this chapter we have shown that that significant improvements in network state estimation accuracy can be achieved through an improved measurement model incorporating bus component dynamics. Further improvements can also be acheived by incorporating the additional information of the loads or load forecasts at the buses. Furthermore, use of the complete physics-based model may not be necessary as this chapter has shown that significant improvements in accuracy are still achievable if the the dynamic state of the full model is reduced by several elements.

Chapter 5

Bad Data

Bad data is an unavoidable fact of life when dealing with measurements in power systems applications [4, 50]. The bad data preprocessor is employed to identify and exclude bad data from the set of telemetered measurements prior to processing by the network state estimator. The high likelihood that some data will be missing or erroneous necessitates the use of a bad data preprocessor for a network state estimator to be useful in real applications.

5.1 Introduction

Bad data detection and identification is a tricky process that has caused difficulties in static network state estimator operation since algorithms were first employed in the late 1960's [50, 40, 41]. The results of the bad data preprocessor are used not just to improve the state estimation result, but also to identify locations of malfunctioning equipment, improper maintenance, and other problems [50]. Thus, it is important that the bad data identification be both accurate and robust.

As described in Sec. 2.4, the detection and identification of bad data is both challenging, and very important to the state estimation process. The difficulty is compounded in the use of the static state estimation formulation in that no *a priori* distribution is available to compare against. When trying to identify a bad measurement from a static state estimate, the state estimate is skewed by the presence of the bad data. An *a priori* estimate of the state provides a dataset for comparison that is not skewed by the bad data, and therefore, the bad data should be more easily distinguishable from the remaining measurements.

When employing dynamic state estimation techniques, three estimates of the network state are available

based on the various sources of state information. The static network state estimate, $\hat{\delta}_{(k)}$ is calculated from the measurements, $\tilde{z}_{(k)}$, and the measurement model alone. The dynamic network state prediction,$\hat{\delta}_{(k/k-1)}$ (a sub-vector of the full dynamic state prediction, $\hat{x}_{(k/k-1)}$), is an *a priori* estimate calculated from the previous dynamic state estimate, $\hat{x}_{(k-1/k-1)}$, the dynamic model, and the input. The dynamic network state estimate, $\hat{\delta}_{(k/k)}$ (a sub-vector of the full dynamic state estimate, $\hat{x}_{(k/k)}$), is an *a posteriori* estimate that incorporates the information contained in both $\hat{\delta}_{(k)}$ and $\hat{x}_{(k/k-1)}$.

This chapter presents new methods using the dynamic estimates of the network state (the *dynamic a priori* state estimate $\hat{\delta}_{(k/k-1)}$ and the *dynamic a posteriori* state estimate $\hat{\delta}_{(k/k)}$) and calculates how each may be used to improve the processes of detecting the existence and locations of bad data over existing methods that use the statically estimated network state, $\hat{\delta}_{(k)}$. The performance of the new detection and identification methods are compared to the static methods currently employed in the electric power industry. An analysis of the computational load of each of the algorithms is also included.

In addition to finding and removing bad data from the measurement vector, we note that the removal of measurements can directly affect the static observabilty of the network. An analysis of the static and dynamic observability is also included and the benefits of dynamic state estimation as they pertain to network state observability are discussed.

5.2 Static Bad Data Detection and Identification

Existing methods for bad data detection require that an initial run of the least squares minimization be completed before any tests for bad data can be accomplished. This initial estimate of the static network state is compared to the measurements, via a Chi square (χ^2) test, to determine if the existence of bad data is likely. Ideally, the χ^2 is implemented as,

$$\text{Step 1: estimate } \hat{\delta} \qquad \hat{\delta}_{(k)} = \text{argmin}_{\delta}(\mathbf{h}(\delta) - \tilde{\mathbf{z}}_{(k)})^T \mathbf{V}^{-1} (\mathbf{h}(\delta) - \tilde{\mathbf{z}}_{(k)})$$

$$\text{Step 2: } \chi^2 \text{ test} \qquad (\mathbf{h}(\hat{\delta}_{(k)}) - \tilde{\mathbf{z}}_{(k)})^T \mathbf{V}^{-1} (\mathbf{h}(\hat{\delta}_{(k)}) - \tilde{\mathbf{z}}_{(k)}) \geq \eta_s$$

where η_s is the test threshold for the static bad data detector [1]. As discussed in Section 2.4.1, the Chi square test measures how likely it is that a statistic corresponds to a multivariate normal distribution with $n_Z - n_\delta$ (or $n_Z - (n_B - 1)$) degrees of freedom and noise that is randomly distributed with less than or equal to the assumed variance. The null hypothesis states that the measurement error distribution conforms to

the assumed noise levels; the bad data hypothesis states that this assumption is violated. Therefore, if the test statistic above exceeds η_s, the bad data hypothesis is accepted.

The initial value for the threshold η_s is chosen based on either a desired confidence level or false alarm rate. For example, if the null hypotheses states that the distribution of the measurements corresponds to our measurement model, employing a 95% confidence interval we would only reject the null hypothesis 5% of the time in the absence of bad data (i.e., a false alarm condition). When bad data exists, the distribution is unknown and the miss rate cannot be calculated directly. Therefore, we will base our threshold on the false alarm rate only.

For the 95% confidence interval, the threshold η_s would then be chosen such that the integral of the expected probability distribution function would equal 0.95, or

$$0.95 = 1 - P_{\text{false alarm}} = \int_{-\infty}^{\eta_s} \text{p.d.f.}_{H_0}(x)dx.$$

Since the number of degrees of freedom depends on of the number of measurements [49], the threshold must be recalculated as the size of the measurement vector changes due to bad data, configuration changes, etc., to maintain a constant false alarm rate (CFAR). The corresponding value for η_s can easily be calculated by using the MATLAB command `CHI2INV`.

5.2.1 Static Bad Data Uncertainty

Successful implementation of the bad data detection preprocessor requires an accurate understanding of the estimated measurement error $\tilde{z} - h(\hat{\delta})$. To accurately perform a χ^2 test, the individual measurement errors must be independent identically distributed (i.i.d.) random variables. This is the case with the actual measurement error $\tilde{z} - h(\delta)$, however since the static network state estimate $\hat{\delta}_{(k)}$ is calculated from the measurements $\tilde{z}_{(k)} = h(\delta_{(k)}) + v_{(k)}$ at the same snapshot, the assumption of independence is violated.

In order to force the estimated measurement errors to be uncorrelated, the measurement error can be normalized through use of the estimated measurement error covariance matrix,

$$\hat{\mathbf{V}}_{(k/\bullet)} = E\left[\left(\tilde{z}_{(k)} - h(\hat{\delta}_{(k/\bullet)})\right)\left(\tilde{z}_{(k)} - h(\hat{\delta}_{(k/\bullet)})\right)^T\right] \tag{5.1}$$

The χ^2 test statistic can be represented as

$$\eta_{(k/\bullet)} = \left(\tilde{z}_{(k)} - h(\hat{\delta}_{(k/\bullet)})\right)^T \hat{\mathbf{V}}^{-1}_{(k/\bullet)} \left(\tilde{z}_{(k)} - h(\hat{\delta}_{(k/\bullet)})\right). \tag{5.2}$$

Using the linearized approximation of $h(\delta) = \mathbf{H}\delta$, and recalling that $\hat{\delta}_{(k)} = \Psi_{(k)}\mathbf{H}^T\mathbf{V}^{-1}\tilde{z}_{(k)}$ from (2.1) and $\Psi_{(k)} = (\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H})^{-1}$ from (2.2) , the measurement estimate is

$$\begin{aligned} h(\hat{\delta}) &= \mathbf{H}\hat{\delta} \\ &= \mathbf{H}\Psi_{(k)}\mathbf{H}^T\mathbf{V}^{-1}\tilde{z}_{(k)} \\ &= \mathbf{H}(\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{V}^{-1}\tilde{z}_{(k)} \\ &= \mathbf{K}_{<\mathbf{H},\mathbf{V}>}\tilde{z}_{(k)} \end{aligned} \tag{5.3}$$

where $\mathbf{K}_{<\mathbf{H},\mathbf{V}>} = \mathbf{H}(\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{V}^{-1}$ is the projection operator projecting $\tilde{z}$ onto the weighted subspace $\mathbf{H}, \mathbf{V}$ [1, 49]. The estimated measurement error is therefore

$$\begin{aligned} \tilde{z} - h(\hat{\delta}) &= \tilde{z} - \mathbf{H}\hat{\delta} \\ &= \tilde{z} - \mathbf{K}_{<\mathbf{H},\mathbf{V}>}\tilde{z}_{(k)} \\ &= \left(\mathbf{I} - \mathbf{K}_{<\mathbf{H},\mathbf{V}>}\right)\tilde{z}_{(k)} \end{aligned} \tag{5.4}$$

$(\mathbf{I} - \mathbf{K}_{<\mathbf{H},\mathbf{V}>})$ is known as the residual sensitivity matrix and represents the sensitivity of the measurement residuals to the measurement errors [1]. The $\mathbf{K}_{<\mathbf{H},\mathbf{V}>}$ and $(\mathbf{I} - \mathbf{K}_{<\mathbf{H},\mathbf{V}>})$ matrix exhibits the following properties

$$\begin{aligned} \mathbf{K}_{<\mathbf{H},\mathbf{V}>}\mathbf{K}_{<\mathbf{H},\mathbf{V}>} &= (\mathbf{H}(\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{V}^{-1})(\mathbf{H}(\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{V}^{-1}) \\ &= \mathbf{H}(\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{V}^{-1} \\ &= \mathbf{K}_{<\mathbf{H},\mathbf{V}>} \end{aligned}$$

$$\begin{aligned} (\mathbf{I} - \mathbf{K}_{<\mathbf{H},\mathbf{V}>})(\mathbf{I} - \mathbf{K}_{<\mathbf{H},\mathbf{V}>}) &= \mathbf{I} - 2\mathbf{K}_{<\mathbf{H},\mathbf{V}>} + \mathbf{K}_{<\mathbf{H},\mathbf{V}>}\mathbf{K}_{<\mathbf{H},\mathbf{V}>} \\ &= \mathbf{I} - 2\mathbf{K}_{<\mathbf{H},\mathbf{V}>} + \mathbf{K}_{<\mathbf{H},\mathbf{V}>} \\ &= \mathbf{I} - \mathbf{K}_{<\mathbf{H},\mathbf{V}>} \end{aligned}$$

$$
\begin{aligned}
\mathbf{K}_{<\mathbf{H},\mathbf{V}>}\mathbf{V}\mathbf{K}_{<\mathbf{H},\mathbf{V}>}^{T} &= (\mathbf{H}(\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{V}^{-1})\mathbf{V}(\mathbf{H}(\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{V}^{-1}) \\
&= \mathbf{H}(\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{V}^{-1}\mathbf{V}\mathbf{V}^{-1}\mathbf{H}(\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H})^{-1}\mathbf{H}^T \\
&= \mathbf{H}(\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H}(\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H})^{-1}\mathbf{H}^T \\
&= \mathbf{H}(\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H})^{-1}\mathbf{H}^T \\
&= \left(\mathbf{H}(\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{V}^{-1}\right)\mathbf{V} \\
&= \mathbf{K}_{<\mathbf{H},\mathbf{V}>}\mathbf{V}
\end{aligned}
$$

$$
\begin{aligned}
(\mathbf{I}-\mathbf{K}_{<\mathbf{H},\mathbf{V}>})\mathbf{V}(\mathbf{I}-\mathbf{K}_{<\mathbf{H},\mathbf{V}>})^{T} &= \mathbf{V}-\mathbf{V}\mathbf{K}_{<\mathbf{H},\mathbf{V}>}^{T}-\mathbf{K}_{<\mathbf{H},\mathbf{V}>}\mathbf{V}+\mathbf{K}_{<\mathbf{H},\mathbf{V}>}\mathbf{V}\mathbf{K}_{<\mathbf{H},\mathbf{V}>}^{T} \\
&= \mathbf{V}-\mathbf{K}_{<\mathbf{H},\mathbf{V}>}\mathbf{V}-\mathbf{K}_{<\mathbf{H},\mathbf{V}>}\mathbf{V}+\mathbf{K}_{<\mathbf{H},\mathbf{V}>}\mathbf{V} \\
&= (\mathbf{I}-\mathbf{K}_{<\mathbf{H},\mathbf{V}>})\,\mathbf{V}
\end{aligned}
$$

If $\mathbf{V}$ is diagonal, $\mathbf{K}_{<\mathbf{H},\mathbf{V}>}$ and $(\mathbf{I}-\mathbf{K}_{<\mathbf{H},\mathbf{V}>})$ are symmetric.

Using these properties, the $\hat{\mathbf{V}}$ matrix can be expressed as

$$
\begin{aligned}
\hat{\mathbf{V}}_{(k)} &= E[(\tilde{z}-h(\hat{\delta}_{(k)}))(\tilde{z}-h(\hat{\delta}_{(k)}))^T] \\
&= E[((\mathbf{I}-\mathbf{K}_{<\mathbf{H},\mathbf{V}>})\tilde{z})((\mathbf{I}-\mathbf{K}_{<\mathbf{H},\mathbf{V}>})\tilde{z})^T] \\
&= E[((\mathbf{I}-\mathbf{K}_{<\mathbf{H},\mathbf{V}>})v)((\mathbf{I}-\mathbf{K}_{<\mathbf{H},\mathbf{V}>})v)^T] \\
&= (\mathbf{I}-\mathbf{K}_{<\mathbf{H},\mathbf{V}>})E[vv^T](\mathbf{I}-\mathbf{K}_{<\mathbf{H},\mathbf{V}>})^T \\
&= (\mathbf{I}-\mathbf{K}_{<\mathbf{H},\mathbf{V}>})\mathbf{V}(\mathbf{I}-\mathbf{K}_{<\mathbf{H},\mathbf{V}>})^T \\
&= (\mathbf{I}-\mathbf{K}_{<\mathbf{H},\mathbf{V}>})\mathbf{V}. \qquad (5.5)
\end{aligned}
$$

5.2.2 Static Bad Data Detection Implementation

In order to make the estimated measurement errors, $\tilde{z}_{(k)}-h(\hat{\delta}_{(k/\bullet)})$, uncorrelated for the χ^2 test, we need to be able to invert $\hat{\mathbf{V}}_{(k/\bullet)}$. Inversion of $\hat{\mathbf{V}}_{(k/\bullet)}$ is not always possible as $(\mathbf{I}-K_{<\mathbf{H},\mathbf{V}>})$ is rank deficient unless there are at least twice as many measurements as network states. The standard method of overcoming this difficulty is to settle for an approximation of the χ^2 distribution. A typical bad data preprocessor implementation

approximates a true χ^2 test by assuming that the estimated measurement error is uncorrelated [1]. Further, we assume that the estimated measurement error is distributed identically to the original measurement error. The χ^2 test statistic, η at time (k) would therefore be calculated as

$$\eta = \sum_{i=n_z} \frac{\left(\tilde{z}_{i(k)} - h_i(\hat{\delta}_{(k)})\right)^2}{\sigma_{ii}^2} \tag{5.6}$$

and therefore be available regardless of the number of measurements available [1]. This approximation has the potential to over or under estimate η however, so an additional threshold is employed to prevent the inadvertent identification of good measurements as bad data.

5.2.3 Static Bad Data Identification Implementation

Once bad data is detected, the bad elements of the measurement vector must be removed to eliminate their detrimental effect on the state estimate. To be removed, the bad elements of the measurement vector must first be identified. This identification is typically accomplished through normalized residual analysis. The weighted residuals are the estimated measurement error $\hat{z}_{(k)} - h(\hat{\delta}_{(k/\bullet)})$ normalized by the standard deviation of the expected noise, σ_i. The most likely candidate for the bad datum is the weighted residual with the largest magnitude. That is

$$i = \max_{i \in \mathbf{N}_{bus}} \frac{\tilde{z}_{i(k)} - h_i(\hat{\delta}_{(k/\bullet)})}{\sigma_{ii}} \tag{5.7}$$

will identify an index i corresponding to the measurement that is most likely corrupted. When the measurement model is nonlinear, this cannot always be guaranteed to be effective [49].

The index identified in (5.7) is then checked to see if its residual is larger than the minimum residual for detection. Requiring that the residual be larger than this threshold (typically $3\sigma_i$) reduces the likelihood of a false positive.

Instead of relying solely on the measurement residuals calculated from the static network state estimate $\hat{\delta}_{(k)}$, the following sections propose that the residuals may be computed from information produced in the process of dynamic network state estimation; specifically the *dynamic a priori* network state estimate $\hat{\delta}_{(k/k-1)}$, and the *dynamic a posteriori* network state estimate $\hat{\delta}_{(k/k)}$ to achieve improved network state estimation performance.

5.3 Dynamic *A Priori* Estimate Based Bad Data Detection and Identification

When employing the dynamic state estimation techniques described in Section 3.8, a value for the network state $\hat{\delta}_{(k/k-1)}$ based on the *dynamic a priori* state estimate $\hat{x}_{(k/k-1)}$ is readily available prior to the gathering of the next set of network measurements. This *a priori* estimate can be used to perform an initial check for bad data. Using the *a priori* estimate for bad data detection is advantageous in at least two ways:

1. The *a priori* estimate $\hat{\delta}_{(k/k-1)}$ is not corrupted by noise $v_{(k)}$ applied to the new measurement so that the smearing effect described in Ch. 2 is non-existent for this initial bad data check.
2. The initial bad data check can be performed without first processing the measurements through the static network state estimator. Thus computational effort need not be wasted on the grossly inaccurate data that can be caught by this initial bad data check.

The predicted network state $\hat{\delta}_{(k/k-1)}$ is a subset of the predicted dynamic state $\hat{\mathbf{x}}_{(k/k-1)}$ and can be extracted from the estimate vector by use of a selection matrix $\mathbf{S}$. An initial χ^2 test,

$$\left(\mathbf{h}(\hat{\delta}_{(k/k-1)}) - \tilde{\mathbf{z}}_{(k)}\right)^T \hat{\mathbf{V}}^{-1}_{(k/k-1)} \left(\mathbf{h}(\hat{\delta}_{(k/k-1)}) - \tilde{\mathbf{z}}_{(k)}\right) \geq \eta_d \tag{5.8}$$

can now be performed on the predicted network state to filter out the worst of the bad data with minimal computation. The test threshold, η_d is a separate threshold for this pre-check and may have a different false alarm rate based on the level of model and dynamic input uncertainty. A smaller dynamic false alarm rate may be desirable in order to avoid inadvertently throwing out good data. Missing some instances of bad data with the predictive bad data preprocessor is acceptable because we know that the traditional bad data detector will likely detect the missed bad data if this initial check doesn't.

The same methodology applies to both the dynamic and the static χ^2 thresholds, η_d and η_s respectively. In normal operation, the threshold value may periodically be adjusted to change its sensitivity to bad data (i.e.,increase or decrease the false alarm rate) as the system conditions evolve over time [48, 2].

5.3.1 Dynamic A Priori Bad Data Detection Uncertainty

The variance of the dynamic prediction for the state is can be larger than the variance of the static estimate due to the process noise and lack of a measurement update. The variances of the dynamic predicted measurement error $(\tilde{z}_{(k)} - h(\hat{\delta}_{(k/k-1)}))$ for the dynamic network state prediction $(\hat{V}_{(k/k-1)})$ and static network

state estimate ($\hat{V}_{(k)}$) are given by:

$$\begin{aligned}
\hat{\mathbf{V}}_{(k/k-1)} &= E[(\tilde{z} - h(\hat{\delta}_{(k/k-1)}))(\tilde{z} - h(\hat{\delta}_{(k/k-1)}))^T] \\
&= E[(\mathbf{H}(\delta) + v_{(k)} - \mathbf{H}(\hat{\delta}_{(k/k-1)}))(\mathbf{H}(\delta) + v_{(k)} - \mathbf{H}(\hat{\delta}_{(k/k-1)}))^T] \\
&= E[(\mathbf{H}(\delta - \hat{\delta}_{(k/k-1)}))(\mathbf{H}(\delta - \hat{\delta}_{(k/k-1)}))^T] + 2E[(\mathbf{H}(\delta - \hat{\delta}_{(k/k-1)}))v_{(k)}^T] + E[v_{(k)}v_{(k)}^T] \\
&= \mathbf{H}\Psi_{(k/k-1)}\mathbf{H}^T + \mathbf{V} \qquad (5.9) \\
&= \mathbf{H}\mathbf{P}_{[\delta](k/k-1)}\mathbf{H}^T + \mathbf{V} \\
&= \mathbf{H}(\mathbf{A}\mathbf{P}_{[(k-1/k-1)}\mathbf{A}^T + \mathbf{Q}_{(k-1)})_{[\delta]}\mathbf{H}^T + \mathbf{V} \qquad (5.10)
\end{aligned}$$

Looking at the dynamic measurement error variance, $\hat{\mathbf{V}}_{(k/k-1)}$, we can see in (5.9) that the error covariance is a function of the dynamic network state error covariance $\mathbf{\Psi}_{(k/k-1)}$ and the measurement noise $\mathbf{V}_{(k)}$. Since $\hat{\delta}_{(k/k-1)}$ is calculated from previous measurements $\tilde{z}_{(k-1)}$ and the previous state estimate $\hat{x}_{k-1/k-1}$ but not the new measurements $\tilde{z}_{(k)}$, the dynamically estimated measurement noise $h(\hat{\delta}_{(k/k-1)}) - h(\delta_{(k)})$ and the actual measurement error $\tilde{z}_{(k)} - h(\delta_{(k)})$ are not correlated. The removal of this correlation indicates that smearing will not be a significant problem and identification of bad data in $\tilde{z}_{(k)}$ should be less difficult.

5.3.2 Dynamic A Priori Bad Data Identification

Due to its structure, $\hat{\mathbf{V}}_{(k/k-1)}$, it is typically invertible so that a true χ^2 test (5.8) can be performed to detect bad data. To save on computation, however, the χ^2 approximated test from Sec. 5.2.2 can alternatively be performed. The same analysis of weighted residuals that was used for the static bad data identification, (5.7), can be applied here using the predicted state to estimate what the measurements will likely be $\tilde{z} - h(\hat{\delta}_{(k/k-1)})$. Analysis of these residuals offers the benefit of a network state estimate that is unaffected by smearing. The residual calculated from the static network state estimate can then be used to confirm the bad data identification performed using the dynamic network state estimate.

Identification of the specific measurements in error is accomplished by analyzing the weighted measurement residuals $(\tilde{z}_i - h_i(\hat{\delta}))/\sigma_{ii}$. Specifically, the maximum weighted residual will be flagged as bad data. This weighted residual will then be compared against a identification threshold to verify that the error is sufficiently large to merit a bad data identification. If the error is sufficiently large, that measurement is removed from the measurement vector and the χ^2 test is repeated until no further measurements are flagged as bad data.

5.4 Dynamic *A Posteriori* Estimate Based Bad Data Detection and Identification

Another way that bad data may potentially be detected is to make use of the *dynamic a posteriori* estimate of the static state, $\hat{\delta}_{(k/k)}$. The *dynamic a posteriori* estimate is an optimal combination of the static network state estimate $\hat{\delta}_{(k)}$ and the *dynamic a priori* estimate network state $\hat{\delta}_{(k/k-1)}$. The Kalman gain at time (k) provides the optimal weighting factor $(\mathbf{K}_{(k)})$ so that the *dynamic a posteriori* network state estimate is calculated as

$$\begin{aligned}\hat{\delta}_{(k/k)} &= \hat{\delta}_{(k/k-1)} + \mathbf{K}_{(k)}\left(\hat{\delta}_{(k)} - \hat{\delta}_{(k/k-1)}\right) \\ &= \left(\mathbf{I} - \mathbf{K}_{(k)}\right)\hat{\delta}_{(k/k-1)} + \mathbf{K}_{(k)}\hat{\delta}_{(k)}. \end{aligned} \tag{5.11}$$

This *dynamic a posteriori* estimate for the network state may be used to further improve the bad data detection process. Due to the weighting of the Kalman gain, the smearing effect is reduced so that a check for bad data may be able to detect bad data where the static bad data preprocessor could not. A weighted residual test can once again be used to identify and remove the bad data from the measurement vector. Once done, $\hat{\delta}_{(k)}$ can be recalculated with the reduced measurement vector and re-incorporated into $\hat{\delta}_{(k/k-1)}$ with minimal effort as the $\mathbf{K}_{(k)}$ and $(\mathbf{I} - \mathbf{K}_{(k)})\hat{\delta}_{(k/k-1)}$ terms in (5.11) remain unchanged.

5.4.1 Dynamic *A Posteriori* Bad Data Detection Uncertainty

Again, we need to determine the variance of the *dynamic a posteriori* estimated measurement error, $\tilde{z}_{(k)} - h(\hat{\delta}_{(k/k)}$. First we inspect the error in more detail and note that

$$\begin{aligned}\tilde{z} - h(\hat{\delta}_{(k/k-1)}) &= \mathbf{H}\delta + v_{(k)} - \mathbf{H}(\hat{\delta}_{(k/k)}) \\ &= \mathbf{H}\delta + v_{(k)} - \mathbf{H}((\mathbf{I} - \mathbf{K}_{(k)})\hat{\delta}_{(k/k-1)}) + \mathbf{K}_{(k)}\hat{\delta}_{(k/k)}) \\ &= \mathbf{H}\delta + v_{(k)} - \mathbf{H}(\mathbf{I} - \mathbf{K}_{(k)})\hat{\delta}_{(k/k-1)} - \mathbf{H}\mathbf{K}_{(k)}(\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{V}^{-1}\tilde{z} \\ &= \mathbf{H}\delta + v_{(k)} - \mathbf{H}(\mathbf{I} - \mathbf{K}_{(k)})\hat{\delta}_{(k/k-1)} - \mathbf{H}\mathbf{K}_{(k)}\Psi_{(k)}\mathbf{H}^T\mathbf{V}^{-1}(\mathbf{H}\delta + v_{(k)}) \\ &= \left(\mathbf{I} - \mathbf{H}\mathbf{K}_{(k)}\Psi_{(k)}\mathbf{H}^T\mathbf{V}^{-1}\right)(\mathbf{H}\delta + v_{(k)}) - \mathbf{H}(\mathbf{I} - \mathbf{K}_{(k)})\hat{\delta}_{(k/k-1)} \\ &= \left(\mathbf{I} - \mathbf{H}\mathbf{K}_{(k)}\Psi_{(k)}\mathbf{H}^T\mathbf{V}^{-1}\right)v_{(k)} + \left(\mathbf{H} - \mathbf{H}\mathbf{K}_{(k)}\Psi_{(k)}\mathbf{H}^T\mathbf{V}^{-1}\mathbf{H}\right)\delta - \mathbf{H}(\mathbf{I} - \mathbf{K}_{(k)})\hat{\delta}_{(k/k-1)} \\ &= \left(\mathbf{I} - \mathbf{H}\mathbf{K}_{(k)}\Psi_{(k)}\mathbf{H}^T\mathbf{V}^{-1}\right)v_{(k)} + \mathbf{H}\left(\mathbf{I} - \mathbf{K}_{(k)}\right)\delta - \mathbf{H}(\mathbf{I} - \mathbf{K}_{(k)})\hat{\delta}_{(k/k-1)}\end{aligned}$$

$$= \left(\mathbf{I} - \mathbf{H}\mathbf{K}_{(k)}\Psi_{(k)}\mathbf{H}^T\mathbf{V}^{-1}\right) v_{(k)} + \mathbf{H}\left(\mathbf{I} - \mathbf{K}_{(k)}\right)(\delta - \hat{\delta}_{(k/k-1)}). \quad (5.12)$$

Using the result of (5.12), the *dynamic a posteriori* estimated measurement error covariance matrix $\hat{\mathbf{V}}_{(k/k)}$ is found to be

$$\begin{aligned}
\hat{\mathbf{V}}_{(k/k)} &= E[(\tilde{z} - h(\hat{\delta}_{(k/k-1)}))(\tilde{z} - h(\hat{\delta}_{(k/k)}))^T] \\
&= E[(\left(\mathbf{I} - \mathbf{H}\mathbf{K}_{(k)}\Psi_{(k)}\mathbf{H}^T\mathbf{V}^{-1}\right) v_{(k)} + \mathbf{H}\left(\mathbf{I} - \mathbf{K}_{(k)}\right)(\delta - \hat{\delta}_{(k/k-1)})) \\
&\quad (\left(\mathbf{I} - \mathbf{H}\mathbf{K}_{(k)}\Psi_{(k)}\mathbf{H}^T\mathbf{V}^{-1}\right) v_{(k)} + \mathbf{H}\left(\mathbf{I} - \mathbf{K}_{(k)}\right)(\delta - \hat{\delta}_{(k/k-1)}))^T] \\
&= (\mathbf{I} - \mathbf{H}\mathbf{K}_{(k)}\Psi_{(k)}\mathbf{H}^T\mathbf{V}^{-1})\mathbf{V}(\mathbf{I} - \mathbf{H}\mathbf{K}_{(k)}\Psi_{(k)}\mathbf{H}^T\mathbf{V}^{-1})^T \\
&\quad + \mathbf{H}(\mathbf{I} - \mathbf{K}_{(k)})\Psi_{(k/k-1)}(\mathbf{I} - \mathbf{K}_{(k)})^T\mathbf{H}^T \\
&= \mathbf{V} + \mathbf{H}(\mathbf{I} - \mathbf{K}_{(k)})\Psi_{(k/k-1)}(\mathbf{I} - \mathbf{K}_{(k)})^T\mathbf{H}^T \\
&\quad - \mathbf{H}(\mathbf{K}_{(k)}\Psi_{(k)} + \Psi_{(k)}\mathbf{K}_{(k)}^T)\mathbf{H}^T + \mathbf{H}\mathbf{K}_{(k)}\Psi_{(k)}\mathbf{K}_{(k)}^T\mathbf{H}^T \\
&= \mathbf{V} + \mathbf{H}(\mathbf{I} - \mathbf{K}_{(k)})\Psi_{(k/k-1)}(\mathbf{I} - \mathbf{K}_{(k)})^T\mathbf{H}^T \\
&\quad + \mathbf{H}\left((\mathbf{I} - \mathbf{K}_{(k)})\Psi_{(k)}(\mathbf{I} - \mathbf{K}_{(k)}) - \Psi_{(k)}\right)\mathbf{H}^T. \\
&= \mathbf{V} + \mathbf{H}\left((\mathbf{I} - \mathbf{K}_{(k)})(\Psi_{(k/k-1)} + \Psi_{(k)})(\mathbf{I} - \mathbf{K}_{(k)}) - \Psi_{(k)}\right)\mathbf{H}^T. \quad (5.13)
\end{aligned}$$

We see that this expression looks a lot like (5.9) with some additional terms to account for the added correlation that is incurred when incorporating the static network state estimate.

5.4.2 Dynamic *A Posteriori* Bad Data Identification

Measurement residuals can be calculated by analyzing the difference between the measurements and the measurement estimates as calculated from the dynamic network state estimate, $\tilde{z} - h(\hat{\delta}_{(k/k)})$. Now that we have $\hat{\mathbf{V}}_{(k/k)}$ we can attempt to detect and identify bad data in the measurements using the χ^2 and weighted residual methods described above. As the dynamic network state estimate uses information from the static network state estimate, we are subject to smearing effects but this effect is reduced by also incorporating the dynamic prediction of the network state.

This final dynamic network state estimate $\hat{\delta}_{(k/k)}$ is effectively the optimal linear combination of the static network state estimate $\hat{\delta}_{(k)}$ and the predicted dynamic network state estimate $\hat{\delta}_{(k/k-1)}$. As such, it has the minimum expected variance from the true state $\delta_{(k)}$. Use of this estimate to calculate measurement residuals potentially results in a more sensitive detector of bad data, but like the static network state estimator, may

also be potentially susceptible to smearing. This method is also the most computationally intensive of the bad data identification methods described here.

5.5 Evaluation

Three methods for detecting and identifying bad data have been presented in sections 5.2 through 5.4:

1. **Static estimation**: $\tilde{z} - h(\hat{x}_{(k)})$: Use only the measurements in the present snapshot to detect and identify bad data. This is the existing method used in industry and performs reasonably well but is subject to smearing, which may hide bad data.
2. **Dynamic *a priori* estimation**: $\tilde{z} - h(\hat{x}_{(k/k-1)})$: Use only the predicted network state to screen for bad data. This method relies on having an accurate model of the temporal behavior of the network and attached components to calculate a good prediction and will tend to miss bad data or flag good data as bad if the prediction is inaccurate. Predicting the network state adds additional computational load but is automatically calculated if a dynamic estimator is in use. This method is not subject to smearing.
3. **Dynamic *a posteriori* estimation**: $\tilde{z} - h(\hat{x}_{(k/k)})$: Use information from both the predicted network state and the static network state estimate. This method should be very effective as the dynamic network state estimate error should be the minimum. This is the most computationally intensive method, but suffers minimum effects from smearing.

The bad data detectors are compared as follows. The magnitude of the error contributing to the bad datum is increased. As the error increases, the point at which each algorithm (static and dynamic detection) detect the bad data is recorded. This provides an estimate of both the false alarm rate and miss rate. A state estimator set to monitor an operating network would record this data so that operators could use it to tune η_s and η_d to achieve the desired performance.

When first initialized, only the static bad data detector is employed to look for bad data. The *dynamic a priori* estimate based bad data detector is unavailable as no *a priori* information is available. The The *dynamic a posteriori* estimate based bad data detector only has information from the first set of measurements and therefore offers no advantages over the static bad data detector.

5.5.1 Evaluation Under Specific Bad Data Scenarios

For a single run of the 250-second simulated scenario, the following instances of bad data are explored. In each case an error is added on top of a noisy injection measurement. The leftmost column of tables 5.1 and 5.2. indicates the magnitude of the additional measurement error in standard deviations. For example, for a value of 4 in the leftmost column indicates that an error four times the standard deviation of the expected error is added to the measurement. The remaining columns indicate how many instances of bad data were detected and identified. For example, 4/6 indicates that four instances of bad data were detected and identified of the six instances injected into the measurement vector.

1. **One injection is a bad datum:** For the 14-bus test system, the error is added at bus 4; for the 118-bus test system, the error is added at bus 49.
2. **One flow is a bad datum:** An additional measurement error is added on top of a noisy flow measurement. For the 14-bus test system, the error is added to the line connecting buses 2 and 4. For the 118-bus test system, the error is added to the line connecting buses 50 and 49.
3. **Two flows are bad data:** The flow errors indicated above are introduced. In addition, a second error is added to a second flow affecting the same bus, but with one half the error magnitude. For the 14-bus test system, the second flow error is added to the line connecting buses 3 and 4. For the 118-bus test system, the second flow error is added to the line connecting buses 51 and 49.

Table 5.1 summarizes the comparative performance of the dynamic prediction based bad data filter, the static estimate based bad data filter, and the dynamic estimate based bad data filter. These algorithms are applied to the IEEE 14-bus test system which has 20 flows and 14 injections for measurements.

	Dynamic Prediction			Static Estimate			Dynamic Estimate		
σ	1 injection	1 flow	2 flows	1 injection	1 flow	2 flows	1 injection	1 flow	2 flows
3	0/3	0/3	1/6	0/3	0/3	1/6	1/3	3/3	2/6
4	0/3	1/3	2/6	0/3	1/3	5/6	2/3	3/3	4/6
5	0/3	**3/3**	4/6	0/3	**3/3**	**6/6**	**3/3**	**3/3**	**6/6**
6	0/3		4/6	1/3					
7	1/3		4/6	1/3					
8	2/3		5/6	1/3					
9	**3/3**		**6/6**	2/3					
10				**3/3**					

Table 5.1: Bad data detected by method (95% confidence interval) on 14-bus system

The static estimate based bad data filter performs as well as the others at detecting a single corrupted flow measurement. The dynamic prediction based filter shows improved performance over the static filter at

detecting a single corrupted injection measurement but is less effective at detecting multiple corrupted flow measurements. The dynamic estimate based bad data filter showed improvements over both the dynamic prediction based filter (for one corrupted injection) and the static estimate based filter (for two corrupted flows). The ability of the three bad data filters to detect a single flow error was identical.

Table 5.2 summarizes the comparative performance of the three preprocessor as applied to the IEEE 118-bus test system which has 180 flows and 118 injections for measurements. When applied to a larger test system, the static estimate based bad data filter performs better than the dynamic prediction based filter in all three categories. The dynamic estimation based bad data filter shows improved likelihood of detecting bad data over the static estimate based filter in all three categories.

	Dynamic Prediction			Static Estimate			Dynamic Estimate		
σ	1 injection	1 flow	2 flows	1 injection	1 flow	2 flows	1 injection	1 flow	2 flows
3	0/3	0/3	0/6	0/3	0/3	1/6	0/3	1/3	0/6
4	0/3	0/3	0/6	0/3	0/3	2/6	0/3	2/3	2/6
5	0/3	0/3	0/6	0/3	0/3	2/6	2/3	2/3	3/6
6	0/3	0/3	0/6	0/3	1/3	2/6	2/3	2/3	4/6
7	0/3	0/3	0/6	0/3	1/3	2/6	**3/3**	**3/3**	4/6
8	0/3	0/3	1/6	0/3	1/3	3/6			4/6
9	0/3	0/3	1/6	0/3	1/3	4/6			4/6
10	0/3	0/3	1/6	0/3	**3/3**	4/6			4/6
11	0/3	0/3	2/6	2/3		4/6			4/6
12	0/3	1/3	3/6	2/3		4/6			4/6
13	0/3	1/3	3/6	2/3		4/6			4/6
14	2/3	2/3	3/6	**3/3**		**6/6**			**6/6**
15	2/3	**3/3**	3/6						
16	**3/3**		3/6						
17			4/6						
18			4/6						
19			4/6						
20			4/6						
21			5/6						
22			**6/6**						

Table 5.2: Bad data detected by method (95% confidence interval) on 118-bus system

The performance of a bad data filter using both the dynamic predicted network state and the static estimated network state independently for detecting bad data demonstrated results comparable with the bad data filters working individually when tested on the IEEE 14-bus test network (Table 5.3) and on the IEEE 118-bus test network (Table 5.4). This implies that the dynamic predicted filter and static filter tend to detect the same errors rather than separate instances. More importantly, this indicates that the dynamic estimate based filter detects and identifies instances of bad data that both the dynamic predicting filter and the static estimating filter miss.

	Combined Predicted and Static		
σ	1 injection	1 flow	2 flows
3	0/3	0/3	3/6
4	0/3	1/3	5/6
5	0/3	**3/3**	**6/6**
6	1/3		
7	1/3		
8	2/3		
9	**3/3**		

Table 5.3: Bad data detected by combination of predicted and static (95% confidence interval) on 14-bus system

	Combined Predicted and Static		
σ	1 injection	1 flow	2 flows
4	0/3	0/3	2/6
5	0/3	0/3	2/6
6	0/3	1/3	2/6
7	0/3	1/3	3/6
8	0/3	1/3	3/6
9	0/3	1/3	4/6
10	0/3	**3/3**	4/6
11	2/3		4/6
12	2/3		4/6
13	2/3		4/6
14	**3/3**		**6/6**

Table 5.4: Bad data detected by combination of predicted and static (95% confidence interval) on 118-bus system

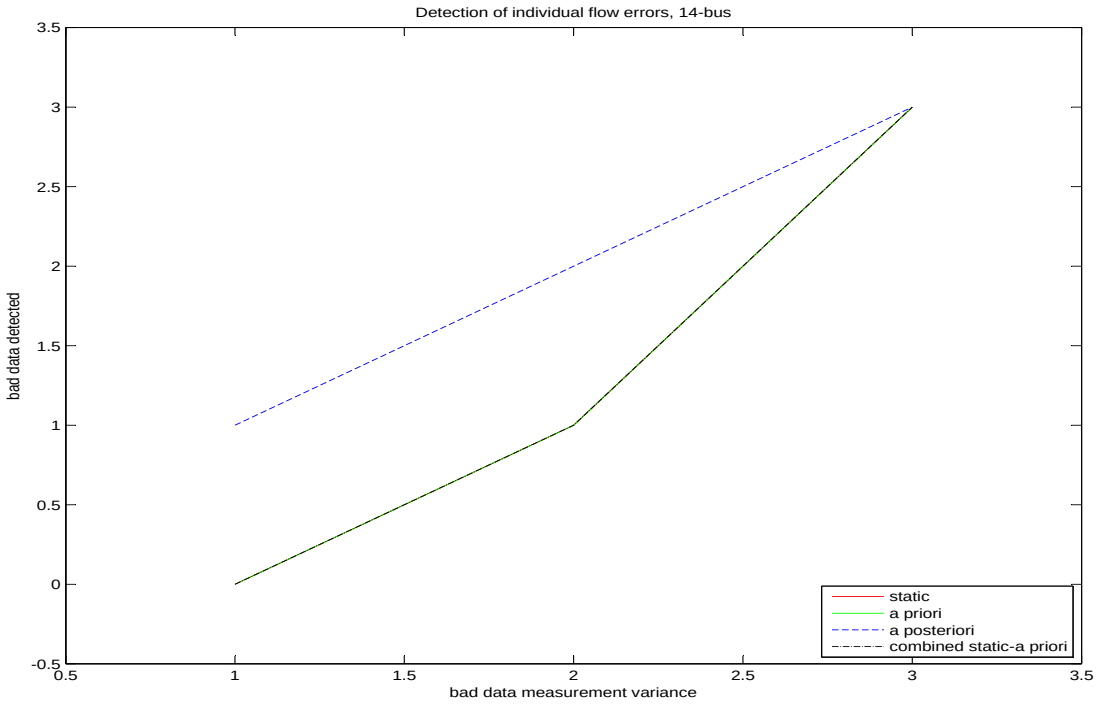


Figure 5.1: Bad data detector performance for individual flow errors on IEEE 14-bus system

The data in tables 5.1 through 5.4 are shown in Figs. 5.1 through 5.6.

5.5.2 Evaluation Under Random Bad Data Scenarios

For a multiple runs of the bad data detection system (50,000 for the 14-bus system, 5,000 for the 118-bus system), a single instances of bad data was injected onto a random measurement. These simulations were run with the normal stepwise loading as described in Sec. 4.3.3 to provide a high accuracy in the a priori network state estimates. The magnitude of the additional error begins at 0 and is increased in increments of 0.1σ up to 10σ, where σ is the expected standard deviation of the measurement noise. The fraction of times that the bad data is detected and identified was recorded. The measurement vector for the 14-bus system has 14 injections and 20 flows, the 118-bus system has 118 injections and 180 flows.

Table 5.5 lists the network state estimates available for use in bad data detection. Figure 5.7 shows the relative detection fraction for the various state estimates when evaluated on the 14-bus system. Figure 5.8 shows the relative detection fraction for the various state estimates when evaluated on the 118-bus system. The red curve on both plots represents the static network state estimation based bad data detector. This is the method used in industry and will be the baseline for comparison. It shows a general trend of higher likelihood of detection as the magnitude of the corrupting bad data increases.

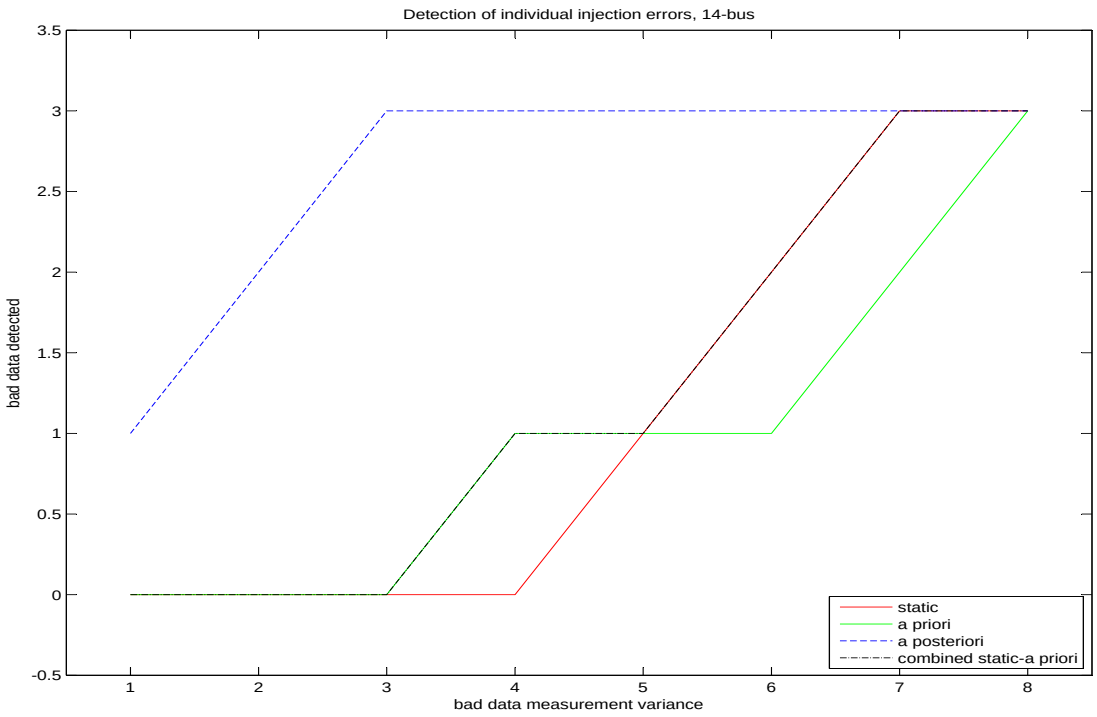


Figure 5.2: Bad data detector performance for individual injection errors on IEEE 14-bus system

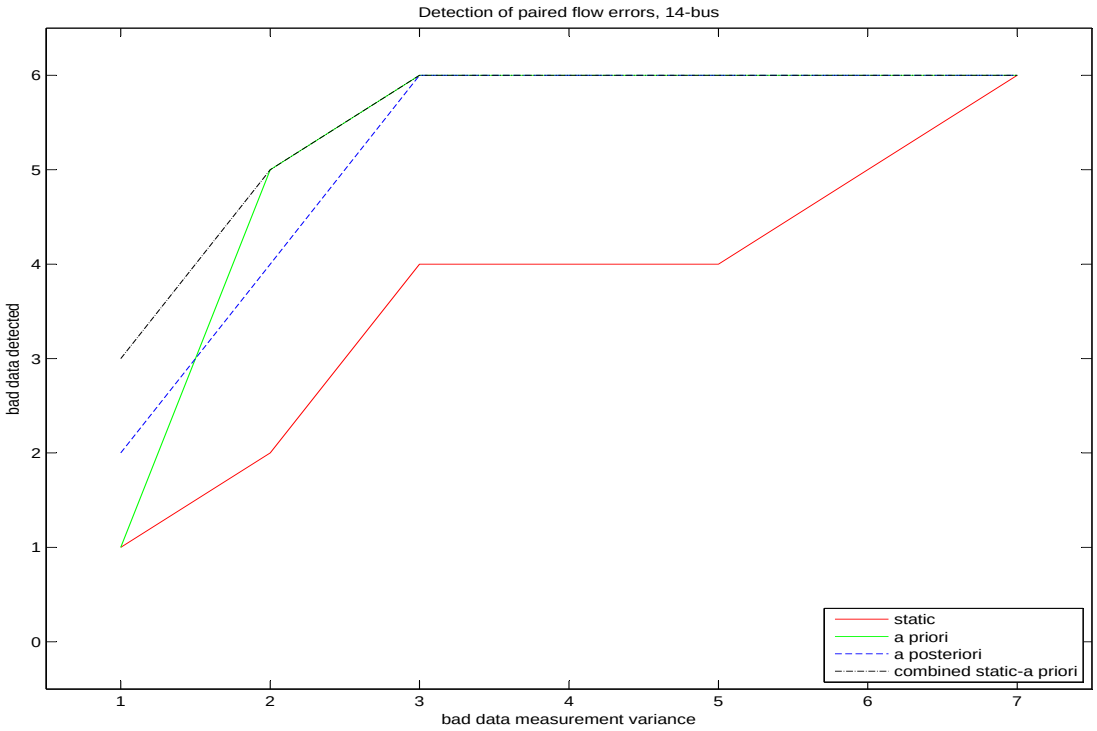


Figure 5.3: Bad data detector performance for pairwise flow errors on IEEE 14-bus system

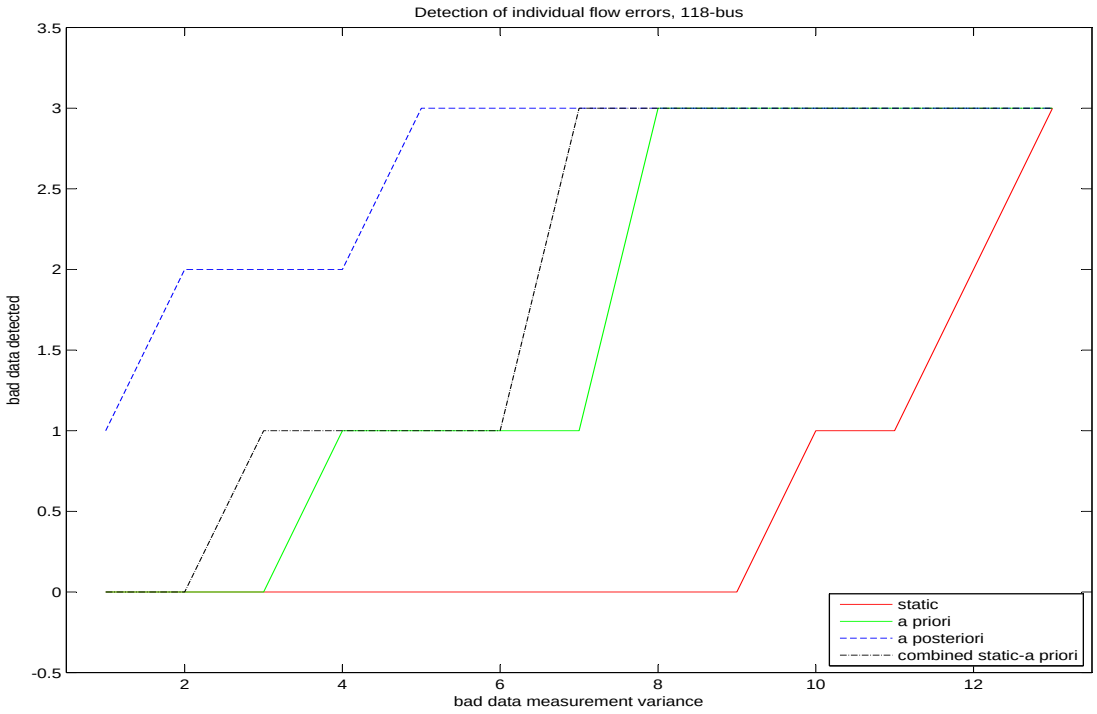


Figure 5.4: Bad data detector performance for individual flow errors on IEEE 118-bus system

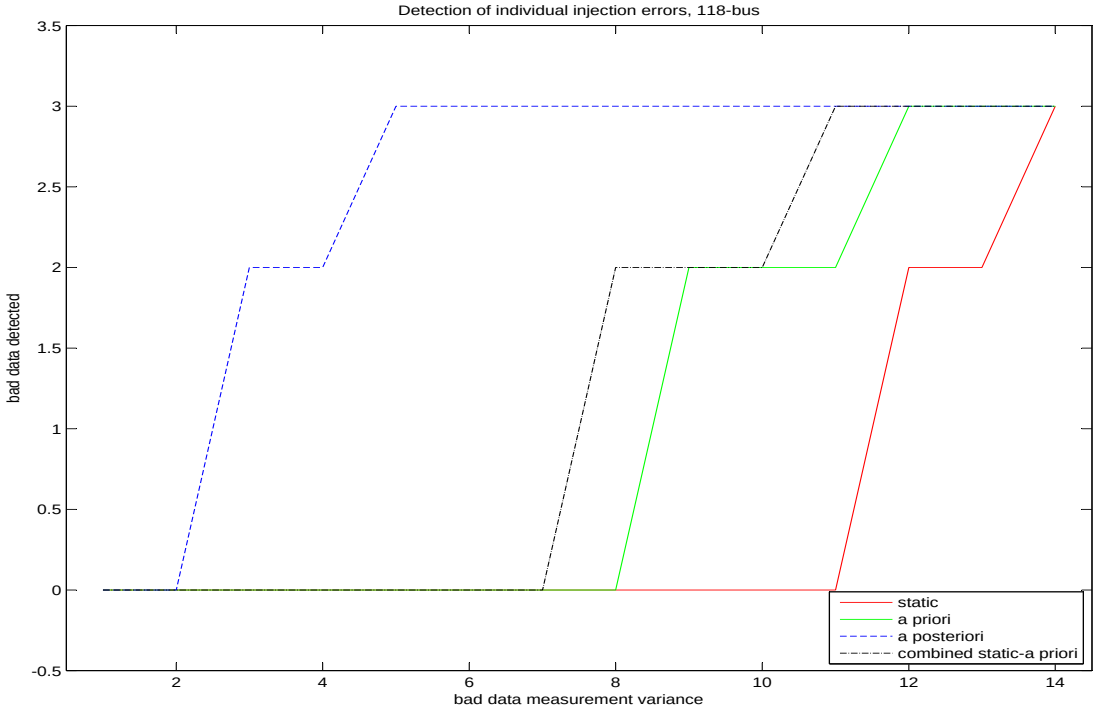


Figure 5.5: Bad data detector performance for individual injection errors on IEEE 118-bus system

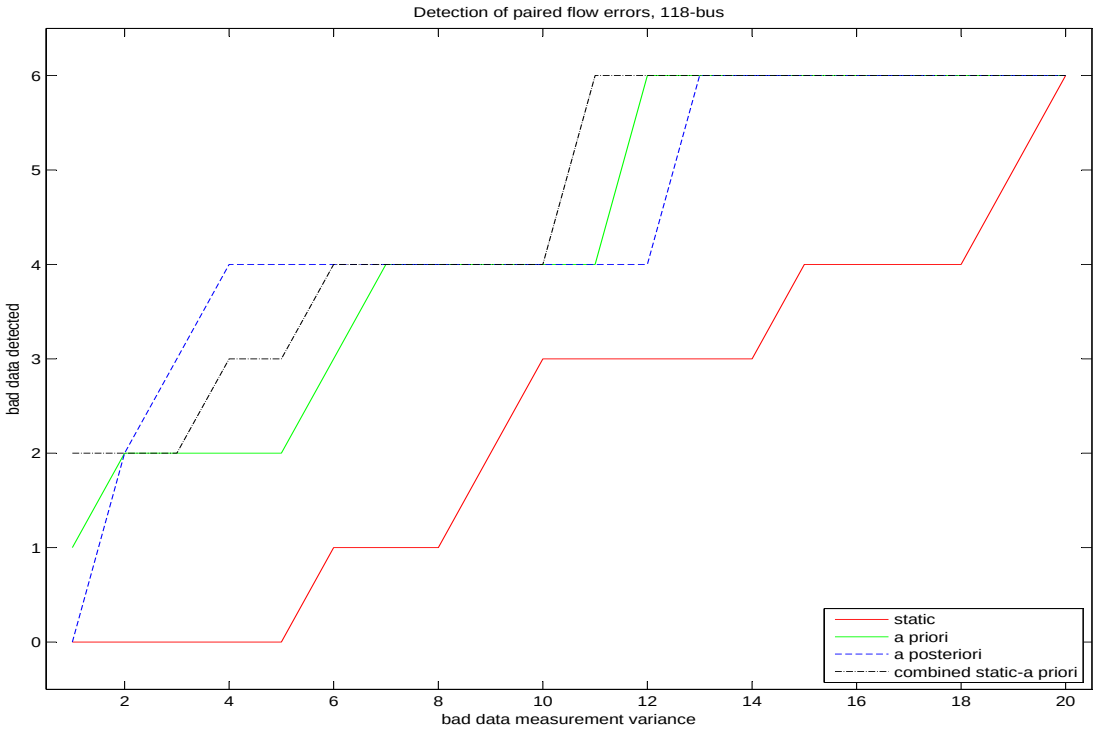


Figure 5.6: Bad data detector performance for pairwise flow errors on IEEE 118-bus system

The augmented network state estimate (both a priori $\hat{\delta}'_{(k/k-1)}$ and a posteriori $\hat{\delta}'_{(k/k)}$) use the same measurement information as the static but makes use of additional component modeling information to perform the estimation dynamically. The a priori estimate rarely detects the bad data on the 14-bus system and doesn't appear to detect it at all on the 118-bus system. As the a priori detection has a relatively low computational cost (effectively just calculation of the χ^2 statistic and a comparison) it may still be worthwhile to use this as a prefilter to catch bad data in order to save computational cost when employing the static bad data detection algorithms later. The a posteriori bad data detector shows a small improvement over the static bad detector but does so at increased computational load.

The standard network state estimate (both a priori $\hat{\delta}_{(k/k-1)}$ and a posteriori $\hat{\delta}_{(k/k)}$) use the same measurement information as the static but in addition factors in information regarding the exogenous inputs at the bus components. These additional measurements plus the component modeling information is used to perform the estimation dynamically. Both the a priori and a posteriori bad data detectors show a significant improvement over the static bad data detector. The a priori bad data detector shows improved detection up to about 6 standard deviation of corrupting noise; the a posteriori bad data detector shows a small improvement in detection over the a priori at values larger than 6 standard deviations. These results indicate that the standard a priori network state estimate $\hat{\delta}_{(k/k-1)}$ based bad data detector provides the best performance

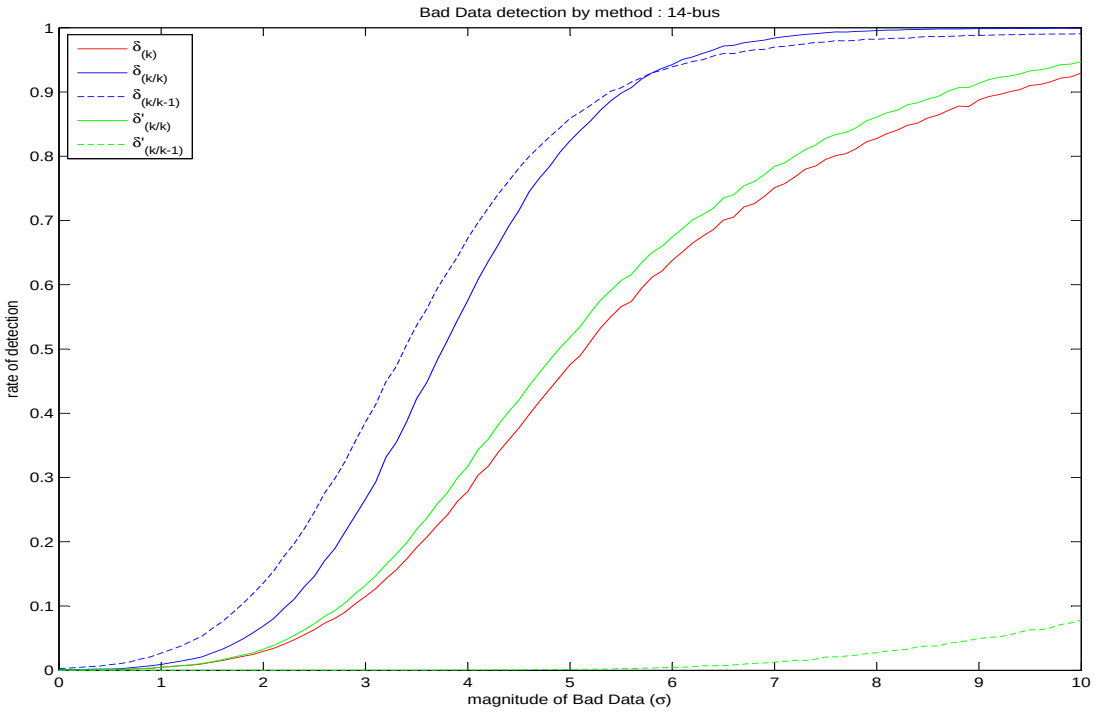


Figure 5.7: Bad data detection fraction vs error magnitude : 14-bus.

when accurate dynamic modeling is possible.

	Estimation method		
Dynamic model	Static	Dynamic a priori	Dynamic a posteriori
Standard	$\hat{\delta}_{(k)}$	$\hat{\delta}'_{(k/k-1)}$	$\hat{\delta}'_{(k/k)}$
Augmented	$\hat{\delta}_{(k)}$	$\hat{\delta}_{(k/k-1)}$	$\hat{\delta}_{(k/k)}$

Table 5.5: Network state estimates used for bad data detection.

5.6 Bad Data and Observability

5.6.1 Static Observability of the Network State

When network state estimation is performed statically, the state estimation result is entirely dependent on a single snapshot of information. This effect is compounded when bad data are detected and removed from the measurement vector [2]. The removal of these bad data may cause portions of the network to become unobservable. In the event that the full network state is not fully observable from this information, some elements of the network state will be unknown and unusable.

An estimator attempting to estimate unobservable states will typically either run into numerical in-

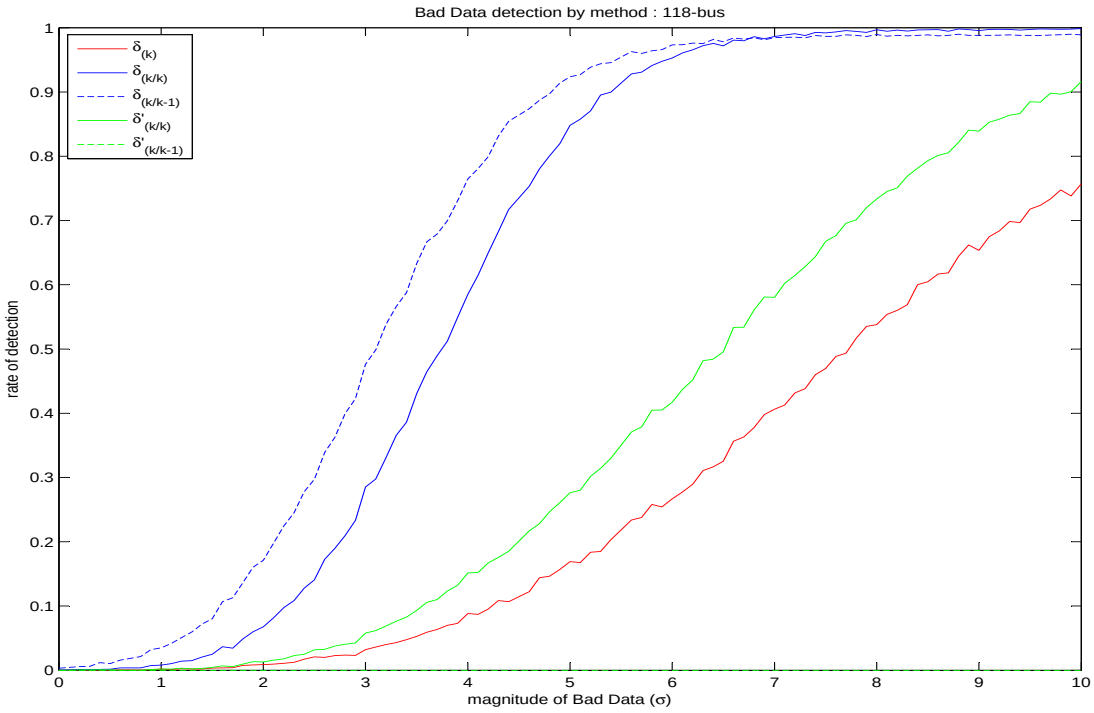


Figure 5.8: Bad data detection fraction vs error magnitude : 118-bus.

stabilities or waste time calculating nonsensical values. Therefore it is important to identify the list of unobservable states and remove them from the estimation problem. Two primary methods are typically employed to identify the list of unobservable states: matrix factorization and nodal analysis [1, 26].

For a linear system, static observability can be determined by calculating the rank of the measurement Jacobian matrix. In a fully observable system, the rank will be equal to the number of states. The matrix factorization method works by stepping through the columns of the measurement Jacobian and only adding states that, when included, increase the rank of the Jacobian. For example, consider the first n columns of the measurement Jacobian. The rank of these columns together have a rank equal to n. When the $n+1$st column is added, if the rank is now equal to $n+1$ then we know this state is observable. If the rank is still n, then the $n+1$st state variable is not observable and should not be included in the state estimation problem [32].

Nonlinear systems for which the factorization method is insufficient may be analyzed using nodal analysis. The nodal analysis method relies on the $P-V$ and $Q-\delta$ decoupling to navigate through a network. An initial observable state is chosen based on the available measurements. Starting from this state, additional states are added to the list of observable states if sufficient measurements exist to determine the relationship between the already identified observable states and the new state. For example, a measurement of real

power flow can define the voltage angle difference between two busses. This process continues until no more states can be added.

In severe situations of unobservability, rather than removing a small subset of unobservable states, it may be necessary to identify two or more observable *islands* in a sea of unobservable states [31, 26].

5.6.2 Dynamic Observability of the Network State

When dynamic state estimation is employed, additional information is available to the state estimator in the form of the state prediction. The state estimator incorporates the information from the predicted state and the measurements (by way of the static state estimate) to calculate an estimate for the state that is more accurate than either independently. In effect, the state prediction is corrected or updated through the incorporation of new information from the measurements. When a state is statically unobservable, the covariances associated with that state are undefined. Normally, incorporating undefined state information would cause difficulties, however the information filter is formulated in such a way that it is able to effectively incorporate this new information even in unobservable conditions.

When the situation arises where portions of the network are statically unobservable, no correction data are available to correct the predicted dynamic state. The weighting factor used to incorporate the new data is zero for the unobservable states so that the dynamic state estimate is just the dynamic predicted state without an update [2]. The information filter automatically keeps track of the information associated with the entire dynamic state and can be used to identify when the variance of the statically unobservable portion become too large to be useful. Thus, instead of losing information as soon as an unobservable condition is present, the information filter exhibits a more *graceful* degradation in the state estimate. Thus, the dynamic state estimator provides more information under more situations than the static network state estimator alone.

Certain requirements need to be met regarding the dynamic model for the bus components for this graceful degradation to be achieved.

1. The bus component dynamic model must be observable from the network injection information associated with that bus.
2. Estimates or direct measurements of the bus injection must be available for sufficient time to fully observe the dynamic bus component state.

The dynamic observability of a dynamic model can be determined by inspection the observability Gramian

matrix. The Gramian matrix,

$$\mathbf{G}(\mathbf{A}_i, \mathbf{C}_i) = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA}^2 \\ \vdots \\ \mathbf{CA}^{(n_{x_i}-1)} \end{bmatrix} \tag{5.14}$$

is composed of the output matrix $\mathbf{C}$ and the state transition matrix $\mathbf{A}$ (3.6) and (3.7). In order to be dynamically observable, the rank of the Gramian must not be less than the number of dynamic states at that bus, n_{x_i}. If the Gramian is rank deficient, only part of the bus component's dynamic state will be observable. Specifically, the number of observable states will be the rank of the $\mathbf{G}(\mathbf{A}_i, \mathbf{C}_i)$.

5.7 Conclusions

This chapter presents two new methods of detecting and identifying bad data in the measurement vector and provided a comparison of those techniques to the existing methods of detecting bad data.

The dynamic network state estimate (both *a priori* and *a posteriori* provided an alternate method to detect bad data which are either free of smearing or have a much reduced effect of smearing respectively.

Also addressed in this chapter are the effects on network state observability achieved by incorporating a dynamic network state estimator. It has been shown that the network state may still be usable when the dynamic network state prediction is calculated even though the network state may not be statically observable. Further study is warranted to determine how many estimate cycles can elapse without measurement updates for a specific element of the network state before the information associated with that state become unusable due to increased error.

Chapter 6

Conclusions

Steady increases in electric power usage have demanded higher and higher performance from the power transmission networks. The capacity of these power transmission network has not increased at a pace to match these demands so that more transmission capacity must be achieved through smarter operation. Smarter operation can only be achieved through providing more accurate and robust information to operators and equipment. This thesis has demonstrated several contributions which may lead the way to the necessary improvements in accuracy and robustness needed to meet our ever increasing energy needs.

6.1 Contributions

This thesis has described and evaluated new algorithms by which the modeling of component dynamics attached to buses in an electric power network can improve the estimation of the network state. These algorithms have been demonstrated on the IEEE 14-bus and IEEE 118-bus test systems through extensive Monte Carlo simulations.

Dynamic Estimation Accuracy. Chapter 4 described and evaluated methods by which the employment of the dynamic network state estimation algorithms developed herein may improve the accuracy of the network state estimate an electric power network. Estimation using the same measurements available to the static estimator but also modeling the bus dynamics ($\hat{\delta}'_{(k/k)}$) offers modest potential gains over static estimation, but maintains those gains even when the model is reduced by several orders. Estimation using the additional information of bus loads or load forecasts along with modeling of the bus dynamics ($\hat{\delta}_{(k/k)}$) offers significant potential increases in the performance of the network state estimator but suffers degradation

when a reduced order dynamic model is employed.

Model Reduction. Chapter 3 described and Ch. 4 evaluated methods by which the model used in the dynamic network state estimation algorithms may be reduced in order to mitigate the additional computational load required to employ the dynamic network state estimation algorithms.

Bad Data Detection and Identification. Chapter 5 described and evaluated methods by which the employment of the dynamic network state estimation algorithms developed herein may improve the detection and identification of bad data on an electric power network.

Observability. Chapter 5 also described the effects on network observability resulting from the employment of the dynamic network state estimation algorithms developed herein may improve the accuracy of the network state estimate an electric power network.

6.2 Further Research

Observable degradation in the performance of the dynamic network state estimator $\hat{\delta}_{(k/k)}$ was observed due to continuous time changes in bus loads but discrete time assumptions regarding the modeling of components at the buses. In essence, the model discretization process assumed a zero-order-hold is applied to the loads. Significant improvements to the performance of the dynamic network state estimator may be achieved through improved methods of discretization of the continuous time models which do not rely on this zero-order-hold assumption. Some possibilities include: alternate methods of continuous time model discretization, incorporating both the load and its rate of change to approximate a first-order-hold, or using load forecasting information to predict the future state of the bus load to enable higher order modeling.

The dynamic a priori network state estimation based bad data filter did not demonstrate improvements over in bad data detectability over the static network state estimation base bad data filter when using the default parameters. Careful adjustment of the detection threshold (χ^2 confidence interval) and the identification threshold (minimum weighted residual for identification) for both the IEEE 14-bus and the IEEE 118-bus test systems allowed the dynamic network state prediction based bad data filter to achieve comparable performance to the static network state estimate based bad data filter. An algorithm could potentially be developed to analyze a network and determine the optimal values for these two thresholds.

The dynamic a posteriori network state estimation based bad data filter demonstrated consistent improvements in bad data detectability over the static network state estimation based bad data filter. While computationally more intense than the existing method, proper caching of data can minimize the additional computation load to a small number of matrix multiplications.

The degree to which a network dynamic model may be reduced before significant degradation in performance was determined experimentally at significant computational cost. The network periodically changes due to various lines and other equipment being brough into and out of service. This analysis would need to be performed for each likely network configuration to make the dynamic model reduction techniques useful when applied to a real system. The development of a method to determine the state reduction threshold via model analysis would be necessary.

Bibliography

[1] A. Abur and A.G. Exposito. *Power System State Estimation: Theory And Implementation*. CRC Press, 2004.

[2] A. Abur, A. Keyhani, and H. Bakhtiari. Autoregressive filters for the identification and replacement of bad data in power system state estimation. *IEEE Transactions on Power Systems*, PWRS-2(3), August 1987.

[3] M.M. Adibi, K.A. Clements, R.J. Kafka, and J.P. Stovall. Integration of remote measurement calibration with state estimation-a feasibility study. *IEEE Transactions on Power Systems*, 7:1164 –1172, Aug 1992.

[4] J. Allemong. State estimation fundamentals for successful deployment. *Proceedings of the 2005 IEEE Power Engineering Society General Meeting*, June 2005.

[5] J. Bay. *Fundamentals of Linear State Space Systems*. McGraw-Hill, 1998.

[6] B.M. Bell and F.W. Cathey. The iterated Kalman filter update as a gauss-newton method. *IEEE Transactions on Automatic Control*, 38(2):294–297, February 1993.

[7] E. A. Blood, B. H. Krogh, J. Ilić, and M. D. Ilić. A Kalman filter approach to quasi-static state estimation in electric power systems. *Proceedings of the 38th North American Power Symposium (NAPS) Carbondale IL*, September 2006.

[8] E. A. Blood, B. H. Krogh, and M. D. Ilić. Electric power system static state estimation through Kalman filtering and load forecasting. *Proceedings of the 2008 IEEE Power Engineering Society General Meeting*, July 2008.

[9] J. Bolz, I. Farmer, E. Grinspun, and P. Schröoder. Sparse matrix solvers on the gpu: conjugate gradients and multigrid. *ACM Trans. Graph.*, 22:917–924, July 2003.

[10] T. A. Davis. *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics,

Philadelphia, 2006.

[11] A.S. Debs and R.E. Larson. A dynamic estimator for tracking the state of a power system. *IEEE Transactions on Power Apparatus and Systems*, PAS-89, Sept/Oct 1970.

[12] ERCOT. Electrical buses and outage scheduling. *http://nodal.ercot.com/*, August 2006.

[13] A. Garcia, A. Monticelli, and P. Abreu. Fast decoupled state estimation and bad data processing. *Power Apparatus and Systems, IEEE Transactions on*, PAS-98(5):1645 –1652, sept. 1979.

[14] Arthur Gelb. *Applied Optimal Estimation*. MIT Press, May 1974.

[15] J.D. Glover, M.S. Sarma, and T.J. Overbye. *Power Systems Analysis and Design*. Thomson Learning, 2008.

[16] K.E. Hillesland, S. Molinov, and R. Grzeszczuk. Nonlinear optimization framework for image-based modeling on programmable graphics hardware. *ACM Trans. Graph.*, 22:925–934, July 2003.

[17] IIT Power Group. One-line diagram of ieee 118-bus test system. *http://www.geocities.jp/ ps_dictionary/ simulation/ IEEE118bus_figure.pdf*, 2003.

[18] J. Ilić. Interactive power systems simulator users manual. *http://www.ece.cmu.edu/∼nsf-education/software.html*, December 2007.

[19] Marija Ilić and John Zaborsky. *Dynamics and Control of Large Electric Power Systems*. Wiley-IEEE Press, March 2000.

[20] M.D. Ilić, L. Xie, U.A. Khan, and J.M.F. Moura. Modeling future cyber-physical energy systems. *2008 IEEE Power Engineering Society General Meeting, Pittsburgh, PA.*, July 2008.

[21] A. Jain and N.R. Shivakumar. Power system tracking and dynamic state estimation. *Power Systems Conference and Exposition, 2009. PSCE '09. IEEE/PES*, March 2009.

[22] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME Journal of Basic Engineering, Ser D*, 82:31–35, 1960.

[23] K. Kato and H.R. Fudeh. Performance simulation of distributed energy management systems. *Power Systems, IEEE Transactions on*, 7(2):820–827, May 1992.

[24] A. Keyhani and A. Abur. Comparative study of polar and cartesian co-ordinate algorithms for power-system state-estimation problems. *IEE Proceedings on Generation, Transmission and Distribution*, 132:132–138, May 1985.

[25] J. Krüger and R. Westermann. Linear algebra operators for gpu implementation of numerical algorithms. *ACM Trans. Graph.*, 22:908–916, July 2003.

[26] G. R. Krumpolz, K. A. Clements, and P. W. Davis. Power system observability analysis: A practical algorithm using network topology. *IEEE Transactions on Power Apparatus and Systems*, PPAS-90, July/Aug 1980.

[27] P. Kundur. *Power System Stability and Control.* McGraw-Hill, Inc., 1994.

[28] A.M. Leite da Silva, M.B. Do Coutto Filho, and J.M.C. Cantera. An efficient state estimation algorithm including bad data processing. *IEEE Transactions on Power Systems*, PWRS-2(4), November 1987.

[29] R. D. Masiello and F. C. Schweppe. A tracking state estimator. *IEEE Transactions on Power Apparatus and Systems*, PAS-90, May/June 1971.

[30] A. Monticelli. *State estimation in electric power systems: A Generalized approach.* Springer, 1991.

[31] A. Monticelli and F.F. Wu. Network observability: Identification of observable islands and measurement placement. *IEEE Transactions on Power Apparatus and Systems*, PAS-104, May 1985.

[32] A. Monticelli and F.F. Wu. Network observability: Theory. *IEEE Transactions on Power Apparatus and Systems*, PAS-104, May 1985.

[33] A. Papoulis and S. U. Pillai. *Probability, Random Variables, and Stochastic Processes.* Addison-Welsley Publishing Company Inc., Reading, Massachusetts, December 1991.

[34] PJM Interconnection. PJM statistics. *http://www.pjm.com*, February 2011.

[35] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C, Second Edition.* Cambridge University Press, 1992.

[36] J. K. Reid. *Large Sparse Sets of Linear Equations.* Academic Press: London and New York, 1971.

[37] P. Rosseaux, T. Van Cutsem, and T. E. Dy Liacco. Whither dynamic state estimation. *Intl Journal of Electric Power & Energy Systems*, 12, April 1990.

[38] R. Rudenberg. *Transient Poerforance of Electric Power Systems.* McGraw-Hill, 1950.

[39] L.L. Scharf. *Statistical Signal Processing, Detection, Estimation, and Time Series Analysis, 4th ed.* McGraw-Hill Science/Engineering/Math, 2001.

[40] F.C. Schweppe. Power system static-state estimation, part iii: Implementation. *IEEE Transactions on Power Apparatus and Systems*, PAS-89:130–135, January 1970.

[41] F.C. Schweppe and E.J. Handschin. Static state estimation in electric power systems. *IEEE Proceedings*, 62, July 1974.

[42] F.C. Schweppe and J. Wildes. Power system static-state estimation, part i: Exact model. *IEEE Transactions on Power Apparatus and Systems*, PAS-89:120–125, January 1970.

[43] Z. Shan and A. Abur. Auto tuning of measurement weights in wls state estimation. *IEEE Transactions on Power Systems*, 19, Nov 2004.

[44] Z. Shan and A. Abur. Combined state estimation and measurement calibration. *IEEE Transactions on Power Systems*, 20, Feb 2005.

[45] K-R Shih and S-J Huang. Application of a robust algorithm for dynamic state estimation of a power system. *IEEE Transactions on Power Systems*, 17(1), Feburary 2002.

[46] I.W. Slutsker, S. Mokhtari, and K.A. Clements. Real time recursive parameter estimation in energy management systems. *IEEE Transactions on Power Systems*, 11:1393 – 1399, Aug 1996.

[47] B. Stott and O. Alsac. Fast decoupled load flow. *Power Apparatus and Systems, IEEE Transactions on*, PAS-93(3):859 –869, may 1974.

[48] Z.G. Stoumbos, Jr. M.R. Reynolds, T.P. Ryan, and W.H. Woodall. The state of statistical process control as we proceed into the 21st century. *Journal of the American Statistical Association*, 95(451):992–998, September 2000.

[49] H.L. Van Trees. *Detection, Estimation, and Modulation Theory, Part I.* John Wiley and Sons, Inc., 1968.

[50] L.S. VanSlyck and J.J. Allemong. Operating experience with the AEP state estimator. *IEEE Transactions on Power Systems*, 3, May 1988.

[51] H. Xue, Q. Jia, N. Wang, Z. Bo, H. Wang, and H. Ma. A dynamic state estimation method with pmu and scada measurement for power systems. *International Power Engineering Conference, 2007. IPEC 2007*, Dec 2007.

Appendices

Appendix A

Terminology and Notation

In the respective fields of *electric power* and *control*, the meaning of the word *state* has a very important difference. For electric power, the system in question is assumed to be static and state refers to the vector of complex phasor voltages at each bus. In this thesis, I consider only the phasor angle and refer to it as the network state, δ. For control, the system in question is dynamic by definition and state refers to the set of variables that describe the present condition of the dynamics. In this thesis, I refer to the dynamic state as x which includes the network state δ among its state variables. The following additional notation is defined and are used throughout this dissertation.

A.1 Network Notation

Bus Numbers

n_B: Number of busses

$\mathbf{N}_B$: List of busses, $\mathbf{N}_B = \{\{1\}, ..., \{n_B\}\}$

n_L: Number of lines

$\mathbf{N}_L$: List of bus connectivity, $\mathbf{N}_L = \{\{i_1, j_1\}, ..., \{i_{n_L}, j_{n_L}\}\}$

$\mathbf{N}_{L_i}$: Subset of List $\mathbf{N}_L$ including only lines connecting to bus i

n_Z: Number of measurements

n_U: Number of inputs

Power

$\mathbf{P}_E$: Vector of injections at all busses

$\mathbf{P}_{E_i}$: Injection at bus i

$\mathbf{P}_{E_{Bi}}$: Injection at bus i, $i \in N_B$

$\mathbf{P}_{E_{ij}}$: Flow on a line from bus i to bus j

$\mathbf{P}_{E_{Lk}}$: Flow on line $k \in N_L$

Phasor Voltage

$\vec{V}_i$: Complex phasor voltage at bus i, $\vec{V}_i = V_i \angle \delta_i = V_i(\cos(\delta_i) + j\sin(\delta_i))$

V_i : Voltage magnitude at bus i

δ_i : Voltage angle difference between bus i and the reference bus (typically bus 1), $\delta_i = \delta_i^{\text{abs}} - \delta_{\text{ref}}^{\text{abs}}$

δ : Vector of all voltage angles δ_i, excluding the reference bus

δ_{ij} : Voltage angle difference between bus i and bus j, $\delta_{ij} = \delta_i - \delta_j$

Admittance

y_{ij} : Complex admittance on the line from bus i to bus j, $i \neq j$. Reciprocal of impedance z_{ij}

$y_{ij} = y_{ji} = g_{ij} + jb_{ij}$,

b_{ij}: Susceptance, component of admittance, inverse of reactance, x_{ij}

g_{ij}: Conductance, component of admittance, inverse of resistance r_{ij}

$\mathbf{Y}$: Complex admittance matrix for a network, $\mathbf{Y} = \mathbf{G} + j\mathbf{B}$

$\mathbf{Y}_i$: i^{th} row of $\mathbf{Y}$

$\mathbf{Y}_{ij}$: Element at position $\{i, j\}$ of $\mathbf{Y}$

$\mathbf{Y}_{ii} = \sum_{\{i,j\} \in N_L} -y_{ij}$

$\mathbf{Y}_{ij} = y_{ij} \forall \{i, j\} \in N_L, 0$ otherwise.

$\mathbf{Y}_{ij} = \mathbf{G}_{ij} + j\mathbf{B}_{ij}$.

A.2 Estimation Notation

Modifiers

$\bullet_{(k)}$: Sub k : Value at time k

$\bullet_{(k/l)}$: Sub k, l : Value at time k given information available up to and including time l

$\bullet_{(k/\bullet)}$: Sub k, bullet : Value at time k, independent of method (static, dynamic a priori, dynamic a posteriori) of calculation.

$\bullet^{(i)}$: Super i : Value after the ith iteration

$\bullet$: No modifier : True value

$\hat{\bullet}$: Hat : Estimated value, also $\hat{\bullet}_{(k/k)}$ for dynamic and $\hat{\bullet}_{(k)}$ for static

$\bar{\bullet}$: Bar : Predicted value, also $\hat{\bullet}_{(k/k-1)}$

$\tilde{\bullet}$: Tilde : Value corrupted with additive white Gaussian noise (AWGN).

E.g., $\tilde{\mathbf{z}} = \mathbf{z} + \mathbf{v}$ where $\mathbf{v}$ is AWGN.

$\bullet_{[v]}$: Sub bracket v : Subset of vector or matrix corresponding to vector v

State

$\mathbf{x}_i$: Bus dynamic state: Vector of the state of the dynamic system at bus i

$\mathbf{x}$: Dynamic state: Concatenation of all bus state vectors, $[\mathbf{x}_1^T, \mathbf{x}_2^T, ..., \mathbf{x}_n^T]^T$

δ : The network state represented as a vector of all voltage angles δ_i, excluding the reference bus. Subset of the dynamic state.

θ : The non-network state. The subset of the dynamic state that does not comprise the network state.

n_x: Number of dynamic state variables

n_{x_i}: Number of dynamic state variables at a bus

n_δ: Number of angles (i.e., network states). Typically $n_B - 1$.

Network State Estimate

$\hat{\delta}_{(k)}$: Static network state estimate. The classic method of processing a single snapshot of data to estimate the network state in a maximum likelihood manner.

$\hat{\delta}'_{(k/k)}$: Augmented dynamic network state estimate. This method estimates the state using dynamic modeling of the components attached to the buses to provide additional information while using the same measurements available to the static network state estimate.

$\hat{\delta}_{(k/k)}$: Standard dynamic network state estimate. This method estimates the state using dynamic modeling of the components attached to the buses and measurements of the mechanical loads applied to these components.

$\hat{\delta}_{(k/\bullet)}$: Network state estimate independent of calculation method.

State error covariance

$\mathbf{P}$: Covariance of $\mathbf{x} - \hat{\mathbf{x}}$

$\mathbf{Y}$: Information matrix, $\mathbf{Y} = \mathbf{P}^{-1}$

σ_k: Variance of x_k. Also $\mathbf{P}_{kk}$

$\mathbf{Q}$: Covariance of process noise

$\mathbf{\Psi}$: Covariance of $\delta - \hat{\delta}$

$\mathbf{\Psi}_{(k)}$ is the same static state error covariance matrix $(\mathbf{H}\mathbf{V}^{-1}\mathbf{H}^T)^{-1}$

$\mathbf{\Psi}_{(k/k)}$ is the δ portion of the $P_{(k/k)}$ matrix or $P_{[\delta](k/k)}$

Measurements / Inputs

z_k : Measurement number k

$\mathbf{z}$: Vector of all measurements

$\mathbf{h}(\delta)$: Measurement as a function of δ

$\tilde{\mathbf{z}}'$: Raw measurement vector containing bad data

$\mathbf{u}_i$: Vector of input at bus i

$\mathbf{u}$: Vector of all inputs

Γ : Vector of quasi-static outside power injections

Measurement / Input Noise

$\mathbf{v}$: Vector of White Gaussian Noise affecting the measurements

$\mathbf{V}$: Covariance matrix of $\mathbf{v}$

$\mathbf{w}$: Vector of White Gaussian Noise affecting the inputs

$\mathbf{W}$: Covariance matrix of $\mathbf{w}$

Appendix B

Test Networks

The following standard IEEE test systems were used to evaluate the algorithms described in this dissertation. The IEEE 14-bus test system (Fig. B.1) has 14 buses and 20 branches. The IEEE 118-bus test system (Fig. B.2) has 118 buses and 186 branches. The IEEE 118-bus test system was modified to consolidate parallel lines so that the system used in simulations has 180 instead of 186 branches.

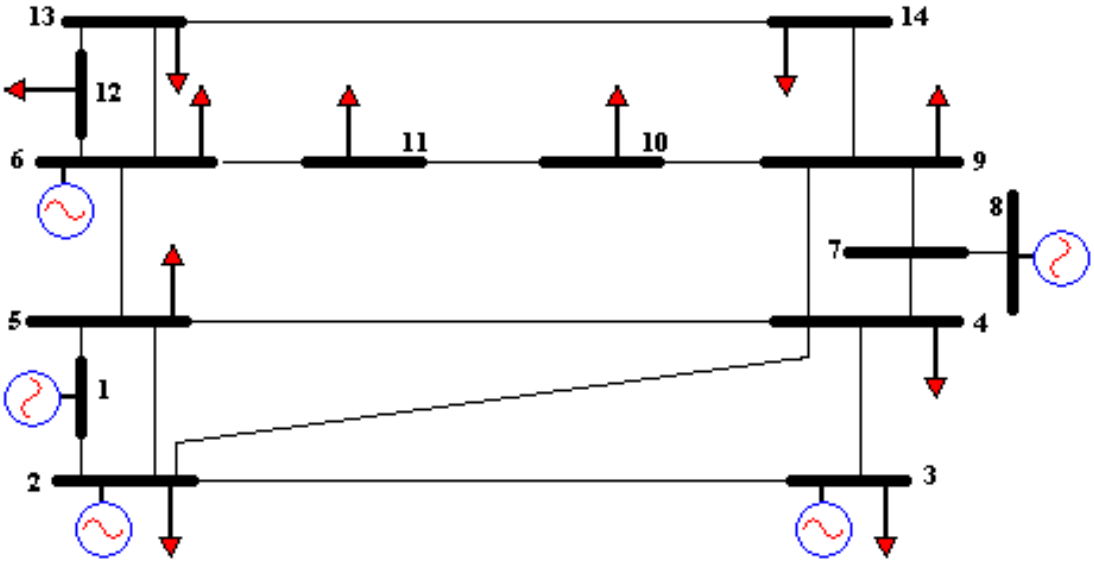


Figure B.1: IEEE 14-bus test system

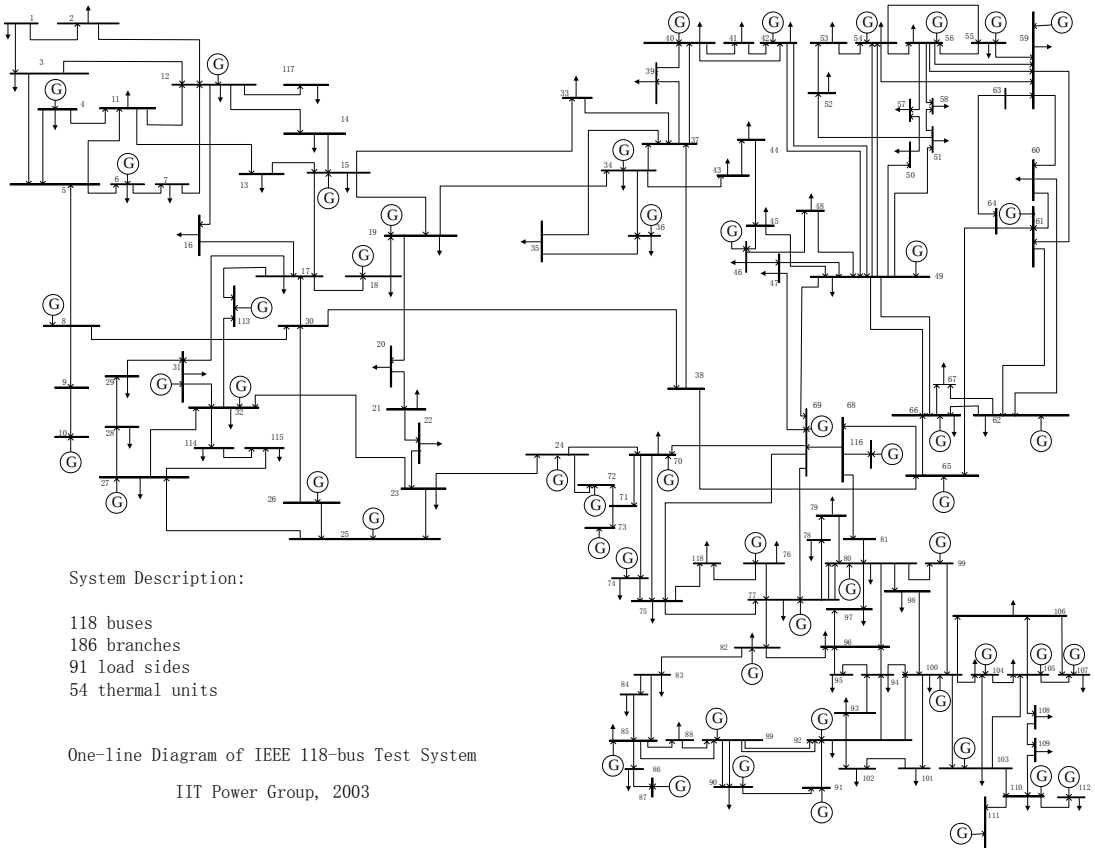


Figure B.2: IEEE 118-bus test system

Appendix C

Matlab Code Listing - Test Data Generation

C.1 fourteen_bus_vars.m

```
%function [ output_args ] = threebus_vars( input_args )
%THREEBUS_VARS Summary of this function goes here
%   Detailed explanation goes here

% Pe isi positive real power injected into
% the network at a bus

% 24 April: correct problem with bus2δ

net=pl_pti23Net('g14bus.raw')
for i=1:length([net.branch.R])
    net.branch(i).R=0;  % make it lossless
end
swing=net.swing;
n_branch=length([net.branch]);
n_bus=length([net.bus]);
susceptance=1./[net.branch.X]';
connectivity=full(sparse([1:n_branch,1:n_branch]',[net.branch.i, net.branch.j]',...
```

```
                          [-ones(n_branch,1);ones(n_branch,1)],n_branch,n_bus));
bus2δ=full(sparse([2:2:2*n_bus]',[1:n_bus]',ones(n_bus,1),2*n_bus,n_bus));
bus2Dδ=(bus2δ*connectivity')';
δminusref=...
[ [eye(swing-1) , -ones(swing-1,1),zeros(swing-1, n_bus-swing)];...
%  [zeros(1,swing-1), -1, zeros(1,n_bus-swing-1)];...
  [zeros(n_bus-swing,swing-1), -ones(n_bus-swing,1),eye(n_bus-swing)]]
bus2omega=full(sparse([2:2:2*n_bus]'-1,[1:n_bus]',ones(n_bus,1),2*n_bus,n_bus));


%Sbase (MVA)
Sb=100;

%-------- Generator 1 ---
%Load response to frequency 1% to 1.5% change in load for 1% change in
%frequency
D=1.5;edit
%Turbine time constant 0.2 to 0.3 sec
T_ch1=0.2;
%Droop characteristic 4% to 5%
R1=0.05;
%Rate Limits  0.1 pu/s opening,   -1.0 pu/s closing
Lco=0.1;
Lcc=-1.0;
%Generator inertia constant 10sec
M1=10
%Governor: Tg=0.2  K=1/(R*Tg)
K1=1/(0.2*R1);
% x=[DeltaA, DeltaPm, DeltaOmega, δ]^T
%BUS 1
A1=[ -K1*R1,      0,  K1,    0 ;
      T_ch1,  -T_ch1,   0,    0 ;
      0    ,   -1/M1,  -D/M1, 0;
      0    ,      0,    1,    0];
B1=[ 0     0  ;
      1     0  ;
      0 -1/M1  ;
      0     0  ];
C1=[zeros(2),eye(2)];
```

```
58 D1=zeros(2);
59
60
61 %———————— Generator 2 ———
62 %BUS 2
63 T_ch2=0.3;
64 R2=0.04;
65 M2=5
66 K2=1/(0.2*R2);
67 A2=[ -K2*R2,     0 ,    K2, 0 ;
68       T_ch2, -T_ch2,      0, 0 ;
69       0    ,  -1/M2  ,-D/M2, 0;
70       0    ,      0   ,   1, 0];
71 B2=[ 0     0  ;
72       1     0  ;
73       0 -1/M2  ;
74       0     0  ];
75 C2=[zeros(2),eye(2)];
76 D2=zeros(2);
77
78 %———————— Synchronous Condenser  ———
79
80 %BUSES 3,6,8
81 Msc=5
82 Asc=[ -D/Msc 0 ;
83       1 0];
84 Bsc= [ -1/Msc, -1/Msc;
85          0  , 0   ];
86 Csc=eye(2);
87 Dsc=zeros(2);
88
89 %———————— Load  ———
90 %BUSES 4,5,7,9,10,11,12,13,14
91 %x3=[omega3, δ3]^T
92
93 M3=1
94 A3=[ -D/M3 0 ;
95       1 0];
96 B3= [ -1/M3, -1/M3;
```

```
97          0 , 0 ];
98  C3=eye(2);
99  D3=zeros(2);
100
101 %------- Combined system -------
102
103 n_δ=14;   %number of busses and therefore number of δs
104
105 n_x=0;        %number of dynamic states so far
106 A_combined=[];  % SE A matrix
107 B_combined=[];  % set B matrix
108 A_sim=[];       % simulink A matrix
109 B_sim=[];       % simulink B matrix
110 C_sim=[];       % simulink C matrix
111 M_δ=[];     % mask of δs
112 M_omega_small=[]; %mask of omegas prior to adding δs
113
114 for(i=1:14)
115     switch(i)
116         case{1}
117             %Generator 1
118             Aaa=A1(1:3,1:3);  % A matrix minus δ
119             Bbb=B1(1:3,:);    % B matrix minus δ
120             Aaaa=A1;
121             Bbbb=B1;
122             Cccc=[0 0 1 0; 0 0 0 1];
123             M_omega_small=[M_omega_small, 0 0 1 ];
124             n_x=n_x+3;
125         case{2}
126             %Generator 2
127             Aaa=A2(1:3,1:3);
128             Bbb=B2(1:3,:);
129             Aaaa=A2;
130             Bbbb=B2;
131             Cccc=[0 0 1 0; 0 0 0 1];
132             M_omega_small=[M_omega_small, 0 0 1 ];
133             n_x=n_x+3;
134         case{3,6,8}
135             %SC 1 (bus 3, 6, 8)
```

```
136             Aaa=Asc(1,1);  % synchronous condenser, only has intertia
137             Bbb=Bsc(1,:);
138             Aaaa=Asc;
139             Bbbb=Bsc;
140             Cccc=[ 1 0; 0 1];
141             M_omega_small=[M_omega_small, 1 ];
142             n_x=n_x+1;
143         otherwise % load (4, 5, 7, 9 - 14)
144             Aaa=A3(1,1);
145             Bbb=B3(1,:);
146             Aaaa=A3;
147             Bbbb=B3;
148             Cccc=[ 1 0; 0 1];
149             M_omega_small=[M_omega_small, 1 ];
150             n_x=n_x+1;
151     end
152     % combined is the matrix that is used for KF
153     A_combined=[A_combined, zeros(size(A_combined,1),size(Aaa,2)); ...
154                         zeros(size(Aaa,1),size(A_combined,2)), Aaa];
155     B_combined=[B_combined, zeros(size(B_combined,1),size(Bbb,2)); ...
156                        zeros(size(Bbb,1),size(B_combined,2)), Bbb];
157     % combined is the matrix that is used for simulink simulation
158     A_sim=[A_sim, zeros(size(A_sim,1),size(Aaaa,2));zeros(size(Aaaa,1),size(A_sim,2)), Aaaa];
159     B_sim=[B_sim, zeros(size(B_sim,1),size(Bbbb,2));zeros(size(Bbbb,1),size(B_sim,2)), Bbbb];
160     C_sim=[C_sim, zeros(size(C_sim,1),size(Cccc,2));zeros(size(Cccc,1),size(C_sim,2)), Cccc];
161 end
162
163 M_omega=[M_omega_small zeros(1,n_bus-1)]; %mask of omega locations
164 [i,j,s]=find(sparse(M_omega));
165 [m,n]=size(M_omega);
166 M_omega2_small=full(sparse(1:length(i),j,s));
167 M_omega2=full(sparse(1:length(i),j,s,n_bus,n));  %omega selection matrix from dynamic state
168
169 M_δ=[zeros(size(M_omega_small)),ones(1,n_bus-1)];
170 [i,j,s]=find(sparse(M_δ));
171 M_δ2=full(sparse(1:length(i),j,s)); %δ selection matrix from dynamic state
172
173 M_δ_not=ones(size(M_δ))-M_δ;  %mask of all but δ
174 [i,j,s]=find(sparse(M_δ_not));
```

```
175 M_δ_not2=full(sparse(1:length(i),j,s,n-n_bus+1,n));%selection matrix of everything but δ
176
177 A_combined_δ=δminusref*M_omega2_small;
178 A_combined=[A_combined;A_combined_δ];
179 A_combined=[A_combined,zeros(size(A_combined,1),n_bus-1)];  % A matrix, uses δ(n-1) vice δ(n)
180
181 B_combined=[B_combined; zeros(n_bus-1,size(B_combined,2))];
182 n_x=n_x+(n_bus-1);
183
184 Pe_multiplex=   full(sparse([1:n_bus]*2,[1:n_bus],ones(1,n_bus),2*n_bus,n_bus));
185 input_multiplex=full(sparse([1:n_bus]*2-1,[1:n_bus],ones(1,n_bus),2*n_bus,n_bus));
```

C.2 oneoneeight_bus_vars.m

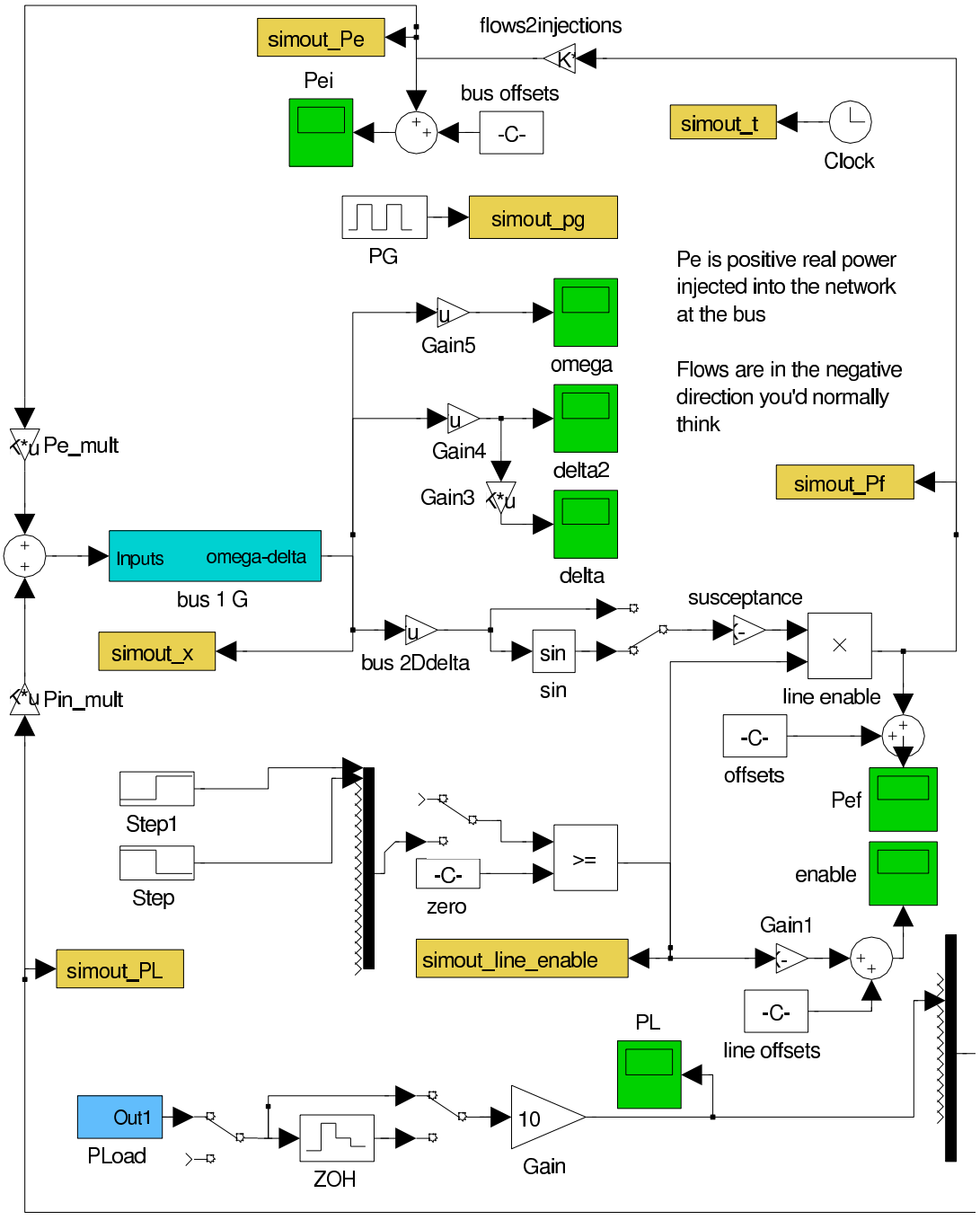
```
%function [ output_args ] = oneoneeight_vars( input_args )
%
% This function puts together the data structures necessary to run the
% Simulink simulation of the 118 bus test system.

% Pe isi positive real power injected into
% the network at a bus

% 20 Feb 2011: convert from 14 bus to 118 bus

net=pl_pti23Net('118BUSnoTXnoParallelQlimhigh.23')
for i=1:length([net.branch.R])
    net.branch(i).R=0;  % make it lossless
end
% force the swing bus to be bus 1
    net.swing=1;
swing=net.swing;
n_branch=length([net.branch]);
n_bus=length([net.bus]);
susceptance=1./[net.branch.X]';
connectivity=full(sparse([1:n_branch,1:n_branch]',[net.branch.i, net.branch.j]', ...
                    [-ones(n_branch,1);ones(n_branch,1)],n_branch,n_bus));
bus2δ=full(sparse([2:2:2*n_bus]',[1:n_bus]',ones(n_bus,1),2*n_bus,n_bus));
bus2Dδ=(bus2δ*connectivity')';
δminusref=...
[ [eye(swing-1) , -ones(swing-1,1),zeros(swing-1, n_bus-swing)];...
%  [zeros(1,swing-1), -1, zeros(1,n_bus-swing-1)];...
  [zeros(n_bus-swing,swing-1), -ones(n_bus-swing,1),eye(n_bus-swing)]]
bus2omega=full(sparse([2:2:2*n_bus]'-1,[1:n_bus]',ones(n_bus,1),2*n_bus,n_bus));


%Sbase (MVA)
Sb=100;

%-------- Generator 1 --- (swing bus, #69)
gen1idx=[69];
```

```
%Load response to frequency 1% to 1.5% change in load for 1% change in
%frequency
D=1.5;
%Turbine time constant 0.2 to 0.3 sec
T_ch1=0.2;
%Droop characteristic 4% to 5%
R1=0.05;
%Rate Limits  0.1 pu/s opening,   -1.0 pu/s closing
Lco=0.1;
Lcc=-1.0;
%Generator inertia constant 10sec
M1=10
%Governor: Tg=0.2  K=1/(R*Tg)
K1=1/(0.2*R1);
% x=[DeltaA, DeltaPm, DeltaOmega, δ]^T
%BUS 1
A1=[ -K1*R1,      0,  K1,    0 ;
      T_ch1,  -T_ch1,   0,    0 ;
      0    ,   -1/M1,  -D/M1, 0;
      0    ,      0,    1,    0];
B1=[ 0     0  ;
      1     0  ;
      0 -1/M1  ;
      0     0  ];
C1=[zeros(2),eye(2)];
D1=zeros(2);


%-------- Generator 2 ---- (bus 89)
gen2idx=[89];
%BUS 2
T_ch2=0.3;
R2=0.04;
M2=5
K2=1/(0.2*R2);
A2=[ -K2*R2,     0 ,    K2, 0 ;
      T_ch2, -T_ch2,      0, 0 ;
      0    ,  -1/M2  ,-D/M2, 0;
      0    ,      0   ,   1, 0];
```

```
B2=[ 0     0  ;
     1     0  ;
     0 -1/M2  ;
     0     0  ];
C2=[zeros(2),eye(2)];
D2=zeros(2);

%-------- Synchronous Condenser (remaining generator busses) ---
scidx=setxor([gen1idx,gen2idx],[net.gen.I]);

%BUSES 3,6,8
Msc=5
Asc=[ -D/Msc 0 ;
      1 0];
Bsc= [ -1/Msc, -1/Msc;
         0  , 0   ];
Csc=eye(2);
Dsc=zeros(2);

%-------- Load  ---
loadidx=setxor([1:118],[net.gen.I]);
%x3=[omega3, δ3]^T

M3=1
A3=[ -D/M3 0 ;
      1 0];
B3= [ -1/M3, -1/M3;
         0 , 0 ];
C3=eye(2);
D3=zeros(2);

%------- Combined system  -------

n_δ=118;   %number of busses and therefore number of δs

n_x=0;        %number of dynamic states so far
A_combined=[];  % SE A matrix
B_combined=[];  % set B matrix
A_sim=[];       % simulink A matrix
```

```
115 B_sim=[];       % simulink B matrix
116 C_sim=[];       % simulink C matrix
117 M_δ=[];     % mask of δs
118 M_omega_small=[]; %mask of omegas prior to adding δs
119
120 for(i=1:n_δ)
121     switch(i)
122         case{gen1idx}
123             %Generator 1
124             Aaa=A1(1:3,1:3);  % A matrix minus δ
125             Bbb=B1(1:3,:);    % B matrix minus δ
126             Aaaa=A1;
127             Bbbb=B1;
128             Cccc=[0 0 1 0; 0 0 0 1];
129             M_omega_small=[M_omega_small, 0 0 1 ];
130             n_x=n_x+3;
131         case{gen2idx}
132             %Generator 2
133             Aaa=A2(1:3,1:3);
134             Bbb=B2(1:3,:);
135             Aaaa=A2;
136             Bbbb=B2;
137             Cccc=[0 0 1 0; 0 0 0 1];
138             M_omega_small=[M_omega_small, 0 0 1 ];
139             n_x=n_x+3;
140         case{scidx}
141             %SC
142             Aaa=Asc(1,1);  % synchronous condenser, only has intertia
143             Bbb=Bsc(1,:);
144             Aaaa=Asc;
145             Bbbb=Bsc;
146             Cccc=[ 1 0; 0 1];
147             M_omega_small=[M_omega_small, 1 ];
148             n_x=n_x+1;
149         otherwise % load
150             Aaa=A3(1,1);
151             Bbb=B3(1,:);
152             Aaaa=A3;
153             Bbbb=B3;
```

```
154             Cccc=[ 1 0; 0 1];
155             M_omega_small=[M_omega_small, 1 ];
156             n_x=n_x+1;
157     end
158     % combined is the matrix that is used for KF
159     A_combined=[A_combined, zeros(size(A_combined,1),size(Aaa,2)); ...
160                         zeros(size(Aaa,1),size(A_combined,2)), Aaa];
161     B_combined=[B_combined, zeros(size(B_combined,1),size(Bbb,2)); ...
162                         zeros(size(Bbb,1),size(B_combined,2)), Bbb];
163     % combined is the matrix that is used for simulink simulation
164     A_sim=[A_sim, zeros(size(A_sim,1),size(Aaaa,2));zeros(size(Aaaa,1),size(A_sim,2)), Aaaa];
165     B_sim=[B_sim, zeros(size(B_sim,1),size(Bbbb,2));zeros(size(Bbbb,1),size(B_sim,2)), Bbbb];
166     C_sim=[C_sim, zeros(size(C_sim,1),size(Cccc,2));zeros(size(Cccc,1),size(C_sim,2)), Cccc];
167 end
168
169 M_omega=[M_omega_small zeros(1,n_bus-1)]; %mask of omega locations
170 [i,j,s]=find(sparse(M_omega));
171 [m,n]=size(M_omega);
172 M_omega2_small=full(sparse(1:length(i),j,s));
173 M_omega2=full(sparse(1:length(i),j,s,n_bus,n));  %omega selection matrix from dynamic state
174
175 M_δ=[zeros(size(M_omega_small)),ones(1,n_bus-1)];
176 [i,j,s]=find(sparse(M_δ));
177 M_δ2=full(sparse(1:length(i),j,s)); %δ selection matrix from dynamic state
178
179 M_δ_not=ones(size(M_δ))-M_δ;  %mask of all but δ
180 [i,j,s]=find(sparse(M_δ_not));
181 M_δ_not2=full(sparse(1:length(i),j,s,n-n_bus+1,n));%selection matrix of everything but δ
182
183 A_combined_δ=δminusref*M_omega2_small;
184 A_combined=[A_combined;A_combined_δ];
185 A_combined=[A_combined,zeros(size(A_combined,1),n_bus-1)];  % A matrix, uses δ(n-1) vice δ(n)
186
187 B_combined=[B_combined; zeros(n_bus-1,size(B_combined,2))];
188 n_x=n_x+(n_bus-1);
189
190 Pe_multiplex=   full(sparse([1:n_bus]*2,[1:n_bus],ones(1,n_bus),2*n_bus,n_bus));
191 input_multiplex=full(sparse([1:n_bus]*2-1,[1:n_bus],ones(1,n_bus),2*n_bus,n_bus));
```


simout_Pe
flows2injections
K
Pei
bus offsets
-C-
simout_t
Clock
simout_pg
PG
Pe is positive real power injected into the network at the bus
Flows are in the negative direction you'd normally think
u
Gain5
omega
u
Gain4
delta2
Gain3
u
delta
simout_Pf
u Pe_mult
Inputs
omega-delta
bus 1 G
susceptance
simout_x
u
bus 2Ddelta
sin
sin
line enable
u Pin_mult
-C-
offsets
Pef
Step1
Step
-C-
zero
>=
enable
Gain1
simout_PL
simout_line_enable
-C-
line offsets
PL
Out1
PLoad
ZOH
10
Gain

C.4 fourteen_bus_grab_data.m

```
%% threebus reduced-order SE
%
% This MATLAB script uses information generated by the
%  simulnk threebus2.mdl system and threebus_vars.m script
%  file to perform state estimation on a simulated electric
%  power system.
%
% The simulink model uses a decoupled real-power only solution
%  (assumes voltage magnitudes==1) and that real power flow is
%  P_12=G sin(d1-d2) (note, this assumption makes current and power
%  effectively interchangeable).
%
% The state estimator uses the same assumptions except that
%  the small angle assumption is used to linearize the trig
%  functions in the power flow equation.
%
% Data from the simulink model is in the following form:
%
% time: nx1 array
% simout_t.signals.values
%
% state (th1, th2, th3, d12, d13): nx5 array
% simout_x.signals.values
%
% input (PL): nx1 array
% simout_PL.signals.values
%
% measurements (Pi): n x n_bus array
%  Pe is positive real power injected into the network at the bus
% simout_Pe.signals.values
%
% transmission lines in service: nx1 array
% enable  x ≥ 0, disable x<0 (generally +/- 1)
% simout_line_enable.signals.values

time=simout_t.signals.values;
```

```
enable=simout_line_enable.signals.values;
Pe=simout_Pe.signals.values;
Pl=simout_PL.signals.values;
Pf=simout_Pf.signals.values;
x_true=simout_x.signals.values;
%simout_x.signals.values

%sample the data

clear data

Ts=1;
k_max=max(time);

data.time(1)=0;
k=1;
data.time(1)=time(1);
data.enable(1,:)=enable(1,:);
data.Pe(1,:)=Pe(1,:);
data.Pl(1,:)=Pl(1,:);
data.x_true(1,:)=x_true(1,:);

for index=1:length(time);
    if(time(index) ≥ floor(data.time(k))+Ts)
        k=k+1;
        data.time(k)=time(index);
        data.enable(k,:)=enable(index,:);
        data.Pe(k,:)=Pe(index,:);
        data.Pl(k,:)=Pl(index,:);
        data.Pf(k,:)=Pf(index,:);
        data.x_true(k,:)=x_true(index,:);
    end
end


save SE14data_10_5_5_1_(10 linear nos yesi).mat data
save SE14params_10_5_5_1.mat A_combined B_combined M_omega M_δ ...
               M_omega2 M_δ2 M_δ_not2 bus2Dδ bus2δ ...
               δminusref bus2omega
```

```
76  save SE14network.mat net
```

Appendix D

Matlab Code Listing - Data Analysis

D.1 oneoneeight_SE7_info_matrix.m

```
%% oneoneeight reduced-order SE
%
% This MATLAB script uses information generated by the
%  simulnk oneoneeightbus.mdl system and threebus_vars.m script
%  file to perform state estimation on a simulated electric
%  power system.
%
% The simulink model uses a decoupled real-power only solution
%  (assumes voltage magnitudes==1) and that real power flow is
%  P_12=G sin(d1-d2) (note, this assumption makes current and power
%  effectively interchangeable).
%
% The state estimator uses the same assumptions except that
%  the small angle assumption is used to linearize the trig
%  functions in the power flow equation.
%
% Data from the simulink model is in the following form:
%
% time: nx1 array
% state (th1, th2, th3, d12, d13): nx5 array
% input (PL3): nx1 array
```

```
% measurements (Pe1, Pe2, Pe3): nx3 array
%  Pe is positive real power injected into the network at the bus
% enable  x ≥ 0, disable x<0 (generally +/− 1)
%  transmission lines in service: nx1 array

% Use the scripts threebus_grab.dat to create the data in the files
% SEdata.mat  SEparams.mat, and SEnetwork.mat, then use
% make_params.m function to convert the SEnetwork data for use.
%  load "data" (time, enable, Pe, Pl, x_true)
%  SEnetwork.mat
%
% This file differs from threebus_SE in that the dynamic estimation uses
% the result from the static estimation as a linear measurement of the sate
% and therefore does not require iteration to incorporate these
% measurements (the iteration was already taken care of in the static
% estimation of the power flows).
%


format compact
clear

%% choose which data to analyze
switch(15)
    case 1
        %data.(time, enable,Pe,Pl,x_true)
        load simulation_data/SE118data_10_5_5_1(10 linear nos yesi)
        %10(gen1) 5(gen2) 5(sc) 1(load) inertial constants (seconds)
        load simulation_data/SE118params_10_5_5_1
        load simulation_data/SE118network   %net
        SEtext='118 lin ns yi 10 5 5 1 (10%)';
        ramp_rate=10;
        nonlinear_SE=false;
    case 2
        load simulation_data/SE118data_10_5_5_1(10 nonlinear nos yesi)
        load simulation_data/SE118params_10_5_5_1
        load simulation_data/SE118network;   %net
        SEtext='118 nonlin(data) lin(est) ns yi 10 5 5 1 (10%)';
        ramp_rate=10;
```

```
61          nonlinear_SE=false;
62      case 3
63          load simulation_data/SE118data_10_5_5_1(10 nonlinear nos yesi)
64          load simulation_data/SE118params_10_5_5_1
65          load simulation_data/SE118network;   %net
66          SEtext='118 nonlin ns yi 10 5 5 1 (10%)';
67          ramp_rate=10;
68          nonlinear_SE=true;
69      case 4
70          load simulation_data/SE118data_10_5_5_1(100 linear nos yesi)
71          load simulation_data/SE118params_10_5_5_1
72          load simulation_data/SE118network;   %net
73          SEtext='118 lin ns yi 10 5 5 1 (100%)';
74          ramp_rate=100;
75          nonlinear_SE=false;
76      case 5
77          load simulation_data/SE118data_10_5_5_1(100 nonlinear nos yesi)
78          load simulation_data/SE118params_10_5_5_1
79          load simulation_data/SE118network;   %net
80          SEtext='118 nonlin(data) lin(est) ns yi 10 5 5 1 (100%)';
81          ramp_rate=100;
82          nonlinear_SE=false;
83      case 6
84          load simulation_data/SE118data_10_5_5_1(100 nonlinear nos yesi)
85          load simulation_data/SE118params_10_5_5_1
86          load simulation_data/SE118network;   %net
87          SEtext='118 nonlin ys yi 10 5 5 1 (100%)';
88          ramp_rate=100;
89          nonlinear_SE=true;
90      case 7
91          load simulation_data/SE14data_10_5_5_1_(10 linear nos yesi)
92          load simulation_data/SE14params_10_5_5_1
93          load simulation_data/SE14network;   %net
94          SEtext='14 lin(data) lin(est) ns yi 10 5 5 1 (10%)';
95          ramp_rate=10;
96          nonlinear_SE=false;
97      case 8
98          load simulation_data/SE14data_10_5_5_1_(10 nonlinear nos yesi)
99          load simulation_data/SE14params_10_5_5_1
```

```
100         load simulation_data/SE14network;   %net
101         SEtext='14 nonlin(data) lin(est) ns yi 10 5 5 1 (10%)';
102         ramp_rate=10;
103         nonlinear_SE=false;
104     case 9
105         load simulation_data/SE14data_10_5_5_1_(10 linear nos yesi)
106         load simulation_data/SE14params_10_5_5_1
107         load simulation_data/SE14network;   %net
108         SEtext='14 nonlin(data) nonlin(est) ns yi 10 5 5 1 (10%)';
109         ramp_rate=10;
110         nonlinear_SE=true;
111     case 10
112         load simulation_data/SE14data_10_5_5_1_(100 linear nos yesi)
113         load simulation_data/SE14params_10_5_5_1
114         load simulation_data/SE14network;   %net
115         SEtext='14 lin(data) lin(est) ns yi 10 5 5 1 (10%)';
116         ramp_rate=100;
117         nonlinear_SE=false;
118     case 11
119         load simulation_data/SE14data_10_5_5_1_(100 nonlinear nos yesi)
120         load simulation_data/SE14params_10_5_5_1
121         load simulation_data/SE14network;   %net
122         SEtext='14 nonlin(data) lin(est) ns yi 10 5 5 1 (10%)';
123         ramp_rate=100;
124         nonlinear_SE=false;
125     case 12
126         load simulation_data/SE14data_10_5_5_1_(100 linear nos yesi)
127         load simulation_data/SE14params_10_5_5_1
128         load simulation_data/SE14network;   %net
129         SEtext='14 nonlin(data) nonlin(est) ns yi 10 5 5 1 (10%)';
130         ramp_rate=100;
131         nonlinear_SE=true;
132     case 13
133         load simulation_data/SE14data_10_5_5_1_(100 linear nos yesi zoh)
134         load simulation_data/SE14params_10_5_5_1
135         load simulation_data/SE14network;   %net
136         SEtext='14 nonlin(data) nonlin(est) ns yi 10 5 5 1 (10%)';
137         ramp_rate=100;
138         nonlinear_SE=false;
```

```
139      case 14
140          load simulation_data/SE14data_10_5_5_1_(100 nonlinear nos yesi zoh)
141          load simulation_data/SE14params_10_5_5_1
142          load simulation_data/SE14network;   %net
143          SEtext='14 nonlin(data) nonlin(est) ns yi 10 5 5 1 (10%)';
144          ramp_rate=100;
145          nonlinear_SE=false;
146     case 15
147          load simulation_data/SE14data_10_5_5_1_(1 linear nos yesi)
148          load simulation_data/SE14params_10_5_5_1
149          load simulation_data/SE14network;   %net
150          SEtext='14 lin(data) lin(est) ns yi 10 5 5 1 (1%)';
151          ramp_rate=1;
152          nonlinear_SE=true;
153      case 16
154          load simulation_data/SE14data_10_5_5_1_(1 nonlinear nos yesi)
155          load simulation_data/SE14params_10_5_5_1
156          load simulation_data/SE14network;   %net
157          SEtext='14 nonlin(data) lin(est) ns yi 10 5 5 1 (1%)';
158          ramp_rate=1;
159          nonlinear_SE=false;
160   end
161
162
163  %%% ----------------------------------------
164
165
166
167
168  %% Set various options
169
170
171  %how many samples sets to do of the simulation (more is more accurate)
172  num_iter=50000;
173
174  % include noise in the measurements and input for estimation
175  noise_on=true;
176  % used for the outer iteration loop.  What range of variances will we
177  % simulate this over (if only one point, then just loop on that one value)
```

```
178 measurement_variance_setpoint=[1e-5];
179
180 % do we want to spend the processor cycles to calculate the trace of the
181 % information matrix emperically?
182 get_trace_info=true;
183
184 % insert bad data into the simulations
185 insert_bad_data=false;
186 inserted_bad_data=9;
187
188 % detect and remove bad data if it exceeds a 0.95% chi square threshold
189 detect_bad_data=false;
190 bad_data_threshold=0.95;
191 identify_threshold=1.62;
192
193 % filter for bad data from predicted δs
194 detect_bad_data_dynamic=false;
195 bad_data_threshold_dynamic=0.95;
196 identify_threshold_d=1.8;%1.6;  % use 1.8 for 118-bus, 1.6 for 14-bus
197 detect_bad_data_dynamicu=false ;
198
199 % filter for bad data from dynamic estiamted δs
200 dynamic_est_BDfilter=false;
201 bad_data_threshold_dynamic_est=0.95;
202 identify_threshold_dh=3;%2.75;
203
204 %plot individual bus error plots
205 bus_error_plots=false;
206
207 %reduce the measurement vector for a few time steps
208 option_few_meas=true;
209 %have bus 3 be unobservable at t=200sec
210 option_unobservable=false;
211
212 %reduce the order of the dynamic model
213 %how many variables to reduce by
214 reduced=0;
215
216
```

```
%% Convert state space representation to reduced order rep
% consolidating the δs to the end
% x= [ [a1 p1 o1] [a2 p2 o2] [ o3] d12 d13]
% xu=[ [a1 p1 o1] [a2 p2 o2] [ o3] d12 d13 [p3] ]

params=make_params( net , data.enable(1,:) );   %make a compressed parameter list
n_bus=params.n_bus; n_branch=params.n_branch;
nswngindx=params.nswngindx;

n_x=length(M_δ);
n_x2=n_x+n_bus;
Mu_δ=[M_δ,zeros(1,n_bus)];
Mu_δ2=[M_δ2,zeros(size(M_δ2,1),n_bus)];
Mu_δ_not=ones(size(Mu_δ))-Mu_δ;
[i,j,s]=find(sparse(Mu_δ_not));
Mu_δ_not2=full(sparse(1:length(i),j,s,n_x2-n_bus+1,n_x2));
Mu_omega2=[M_omega2,zeros(size(M_omega2,1),n_bus)];
M_input=[zeros(n_bus,n_x),eye(n_bus)];

Pe_in=sparse((1:n_bus)'*2,(1:n_bus)', ones(1,n_bus), 2*n_bus, n_bus);
Pe_L=sparse((1:n_bus)'*2-1,(1:n_bus)', ones(1,n_bus), 2*n_bus, n_bus);

dt= bus2δ'*data.x_true';
δ_true=dt-ones(n_bus,1)*dt(net.swing,:);

 %initialize variables
δ_hat(:,1)=zeros(n_bus,1);
x_hat(:,1)=zeros(n_x,1);
x_bar(:,1)=zeros(n_x,1);
xu_hat(:,1)=zeros(n_x2,1);
xu_bar(:,1)=zeros(n_x2,1);

A_2=A_combined;

%specify measurements
meas.mPF=[[net.branch.i]',[net.branch.j]'];
meas.iPF=(1:n_branch)';
meas.mPI=([net.bus.i])';
meas.iPF=(1:n_bus)';
```

```
n_samples=length(data.time);
ps=zeros(1,n_samples); pd=zeros(1,n_samples); pdu=zeros(1,n_samples);
pys=zeros(1,n_samples); pyd=zeros(1,n_samples); pydu=zeros(1,n_samples);
ob=zeros(1,n_bus);
dofs=zeros(1,n_samples); dofd=zeros(1,n_samples);
dof_dh=zeros(1,n_samples); dof_dhu=zeros(1,n_samples);
c_s=zeros(1,n_samples);c_d=zeros(1,n_samples);
c_dh=zeros(1,n_samples);c_dhu=zeros(1,n_samples);

%% Simulate processing measurements corrupted with random noise

% run with random noise
for measurement_variance=measurement_variance_setpoint;


input_variance=measurement_variance/4;

W=input_variance*eye(1);
W2=0.01/5*ramp_rate/100;  %governed by input ramp rate

xs_error=zeros(n_bus-1,n_samples);  %sum of static error
xs2_error=zeros(n_bus-1,n_samples); %sum of static e^2
xd_error=zeros(n_bus-1,n_samples);  %sum of dynamic e
xd2_error=zeros(n_bus-1,n_samples); %e^2
xdu_error=zeros(n_bus-1,n_samples); %e (with input estimation)
xdu2_error=zeros(n_bus-1,n_samples);%e^2
Ps_emperical=zeros(n_bus-1,n_bus-1,n_samples);
Pd_emperical=zeros(n_bus-1,n_bus-1,n_samples);
Pdu_emperical=zeros(n_bus-1,n_bus-1,n_samples);

sprintf('Start var=%d, %s',input_variance,datestr(now))

%% iterate over the measurements

for iter=1:num_iter
    if(mod(iter,100)==0)
        disp([datestr(now), sprintf(' %i/%i', iter,num_iter)]);
    end
    i=1;
```

```
295      randn('state',iter);
296      measurement_noise_inject=random('norm',0,sqrt(measurement_variance),n_samples,n_bus);
297      measurement_noise_flow=random('norm',0,sqrt(measurement_variance),n_samples,n_branch);
298      input_noise=random('norm',0,sqrt(input_variance),n_samples,n_bus);
299
300       if noise_on
301          Pi=data.Pe+measurement_noise_inject;
302          Pf=-data.Pf+measurement_noise_flow;
303          Pl=data.Pl+input_noise;
304       else
305          Pi=data.Pe;
306          Pf=-data.Pf;
307          Pl=data.Pl;
308       end
309
310
311      %% perform initial static estimation
312      meas.mPI= [net.bus.i]';
313      meas.iPI=[net.bus.i]';
314      meas.mPF= [net.branch.i; net.branch.j]';
315      meas.iPF=(1:n_branch)';
316      z=[Pi(1,:)';Pf(1,:)'];
317      params=make_params(net,data.enable(1,:));
318      %% ------- perform static estimation and filter as  necessary ------
319
320     [δ_hat(:,i),infoIs,obs,iis,dofs(i),c_s(i)]=est_det_ident_static( ...
321          z, meas,params, net, measurement_variance, ...
322          bad_data_threshold, identify_threshold, ...
323          nonlinear_SE,detect_bad_data);
324      ps(i)=iis;
325      if(get_trace_info)
326           pys(i)=trace(infoIs(obs,obs));
327      end
328
329  %make state transition matrix for dynamic model
330      Y2=imag(make_Y(net, data.enable(i,:)) );
331      Atc=A_2-B_combined*Pe_in*Y2(:,nswngindx)*M_δ2;
332      % compute reduced order state transition matrix
333      At=expm(Atc);
```

```
334     if(reduced ~=0 )
335         [u,s,v]=svds(At,size(At,1)-reduced,'L');
336         At=u*s*v';
337     end
338     B_cd2=Atc\(At-eye(size(At)))*B_combined*Pe_L;
339
340     %%initalize dynamic initialy to static state
341     x_hat(:,1)=M_d2'*d_hat(nswngindx,i);
342     Y_hat=M_d2'*infoIs(nswngindx,nswngindx)*M_d2...
343         +(eye(n_x)-M_d2'*M_d2);
344     P_hat=inv(Y_hat);
345
346 %    pd_old(1)=trace(M_d2*Y_hat*M_d2');
347     if(get_trace_info)
348        pyd(i)=trace(M_d2*Y_hat*(eye(n_x)-...
349             M_d_not2'/(M_d_not2*Y_hat*M_d_not2')*...
350             M_d_not2*Y_hat)*M_d2');
351     end
352     pd(i)=sum(1 ./diag(P_hat(n_x-(n_bus-1)+1:n_x, n_x-(n_bus-1)+1:n_x)));
353
354 %make state transition matrix for dynamic-integrating input model
355     Atu=[[At, B_cd2];[zeros(n_bus,size(At,2)),eye(n_bus)]];
356     Bu_cd2=[B_cd2;zeros(n_bus,size(B_cd2,2))];
357
358     %%initalize dynamic state to static state
359     xu_hat(:,i)=Mu_d2'*d_hat(nswngindx,i);
360     Yu_hat=Mu_d2'*infoIs(nswngindx,nswngindx)*Mu_d2 ...
361         +(eye(n_x2)-Mu_d2'*Mu_d2);
362     Pu_hat=inv(Yu_hat);
363     if(get_trace_info)
364         pydu(i)=trace(Mu_d2*Yu_hat*(eye(n_x2)-...
365             Mu_d_not2'/(Mu_d_not2*Yu_hat*Mu_d_not2')*...
366             Mu_d_not2*Yu_hat)*Mu_d2');
367     end
368     pdu(i)=sum(1 ./diag(Pu_hat(n_x-(n_bus-1)+1:n_x, n_x-(n_bus-1)+1:n_x)));
369
370
371     %% iterate on the measurements
372     for i= 2:n_samples
```

```
373
374  %% ------update params if necessary (i.e., topology change)
375          if(i>1)
376               if( sum(data.enable(i,:) ~= data.enable(i-1,:))  )
377                    params=make_params(net,data.enable(i,:));
378               end
379          end
380
381           if(i>2) %update dynamic model if necessary (measured u)
382              if( sum( data.enable(i-1,:) ~= data.enable(i-2,:))  )
383                  Y2=imag(make_Y(net, data.enable(1,:)) );
384                  Atc=A_2-B_combined*Pe_in*Y2(:,nswngindx)*M_d2;
385                  % compute reduced order state transition matrix
386                  At=expm(Atc);
387                  if(reduced ~=0)
388                      [u,s,v]=svds(At,size(At,1)-reduced,'L');
389                      At=u*s*v;
390                  end
391                  B_cd2=Atc\(At-eye(size(At)))*B_combined*Pe_L;
392                  %update dynamic model if necessary (estimated u)
393                  Atu=[[At, B_cd2];[zeros(n_bus,size(At,2)),eye(n_bus)]];
394                  Bu_cd2=[B_cd2;zeros(n_bus,size(B_cd2,2))];
395               end
396           end
397
398          %% predict dynamic estimates ---------------------------
399          %------------- measured u ---------------
400
401          %predict
402          x_bar(:,i)=At*x_hat(:,i-1)+B_cd2*Pl(i-1,:)' ;
403
404          %don't need to reset swing bus to zero because d portion of
405          % x already subtracts it
406
407          P_bar=At*P_hat*At'+B_cd2*W*B_cd2'+(2.5e-7)*M_omega2'*M_omega2;
408          Y_bar=inv(P_bar);
409          y_bar=Y_bar*x_bar(:,i);
410
411          %------------- estimated u ---------------
```

```
412
413         %predict
414         xu_bar(:,i)=Atu*xu_hat(:,i-1);
415         Pu_bar=Atu*Pu_hat*Atu'+W2*(M_input'*M_input);
416         Yu_bar=inv(Pu_bar);
417         yu_bar=Yu_bar*xu_bar(:,i);
418
419         %% gather new measurements and correct the predictions
420
421         if(i>150 && i<180 && option_few_meas)
422         % flows only
423             meas.mPI=[];
424             meas.iPI=[]';
425             meas.mPF= [net.branch.i; net.branch.j]';
426             meas.iPF=(1:n_branch)';
427             z=Pf(i,:)'.*data.enable(i,:)'; %disregard flows if a line is off
428         elseif(i>=160 && i<165 && n_bus==118 && option_unobservable)
429             %bus 30/118 unobservable (flows only)
430             meas.mPI=[];
431             meas.iPI=[];
432             meas.mPF= [net.branch.i; net.branch.j]';
433             meas.iPF=(1:n_branch)';
434             %flows identically zero if a line is off
435             z=Pf(i,:)'.*data.enable(i,:)';
436
437             blah=1:n_branch;  % set the measurement vector to make bus 3
438             blah(50)=[];         %  statically unobservable
439             blah(44)=[];
440             blah(28)=[];
441             blah(14)=[];
442
443             meas.mPF=meas.mPF(blah,:);
444             meas.iPF=meas.iPF(blah,:);
445             z=z(blah,:);
446         elseif(i>=160 && i<165 && n_bus==14 && option_unobservable)
447             %bus 3/14 unobservable (flows only)
448             meas.mPI=[];
449             meas.iPI=[];
450             meas.mPF= [net.branch.i; net.branch.j]';
```

```
451             meas.iPF=(1:n_branch)';
452             %flows identically zero if a line is off
453             z=Pf(i,:)'.*data.enable(i,:)';
454
455             blah=1:n_branch;  % set the measurement vector to make bus 3
456             blah(6)=[];         %  statically unobservable
457             blah(3)=[];
458
459             meas.mPF=meas.mPF(blah,:);
460             meas.iPF=meas.iPF(blah,:);
461             z=z(blah,:);
462         elseif(i>200 && i<210 && option_few_meas)
463             %missing more measurements (barely observable)
464         % injections only
465             meas.mPI= [net.bus.i]';  %injections only
466             meas.iPI= [net.bus.i]';
467             meas.mPF= [];
468             meas.iPF= [];
469             z=Pi(i,:)';
470         else %full measurment vector
471             meas.mPI= [net.bus.i]';
472             meas.iPI= [net.bus.i]';
473             meas.mPF= [net.branch.i; net.branch.j]';
474             meas.iPF= (1:n_branch)';
475             z=[Pi(i,:)';Pf(i,:)'.*data.enable(i,:)'];
476         end
477
478         %% ------------------ insert bad data --------------------
479         if(insert_bad_data)
480             if(i==50 || i==55 || i==60 ) %injection
481                 if(n_bus==118)
482                     %bad injection at bus 49
483                     z(49)=z(49)+inserted_bad_data*sqrt(measurement_variance);
484                 elseif(n_bus==14)
485                     %bad injection at bus 4
486                     z(4)=z(4)+inserted_bad_data*sqrt(measurement_variance);
487                 end
488             end
489             if( i==70 || i==75 || i==80) %injection
```

```
490                 if(n_bus==118)
491                     %bad flow between 49 and 50
492                     z(118+79)=z(118+79)+inserted_bad_data*sqrt(measurement_variance);
493                 elseif(n_bus==14)
494                     %bad flow between 2 and 4
495                     z(14+4)=z(14+4)+inserted_bad_data*sqrt(measurement_variance);
496                 end
497             end
498             if( i==90 || i==95 || i==100) %injection
499                 if(n_bus==118)
500                     %bad flow between 49 and 50
501                     z(118+79)=z(118+79)+inserted_bad_data*sqrt(measurement_variance);
502                     z(118+80)=z(118+80)+inserted_bad_data/2*sqrt(measurement_variance);
503                 elseif(n_bus==14)
504                     %bad flow between 2 and 4
505                     z(14+4)=z(14+4)+inserted_bad_data*sqrt(measurement_variance);
506                     z(14+6)=z(14+6)+inserted_bad_data/2*sqrt(measurement_variance);
507                 end
508             end
509         end
510
511          %% ------------- filter for bad data (dynamic) ---------------------
512         δ_bar=zeros(n_bus,1);
513         if(detect_bad_data_dynamicu==false)
514             δ_bar(nswngindx,:)= M_δ2*x_bar(:,i);
515             Psi=M_δ2*P_bar*M_δ2';
516         else
517             δ_bar(nswngindx,:)= Mu_δ2*xu_bar(:,i);
518             Psi=Mu_δ2*Pu_bar*Mu_δ2';
519         end
520
521         [z,meas,dofd(i),c_d(i)]=det_ident_dynamic(z,meas,δ_bar,Psi, ...
522             params, net, measurement_variance, ...
523             bad_data_threshold_dynamic, identify_threshold_d, ...
524             nonlinear_SE,detect_bad_data_dynamic);
525
526
527         %% ------- perform static estimation and filter as  necessary ------
528
```

```
529         [δ_hat(:,i),infoIs,obs,iis,dofs(i),c_s(i)]=est_det_ident_static( ...
530             z, meas,params, net, measurement_variance, ...
531             bad_data_threshold, identify_threshold, ...
532             nonlinear_SE,detect_bad_data);
533
534         ob(i)=length(obs);  %how many observable states?
535
536
537          %% ———— perform dynamic estimation ——————————————————————
538
539          %————————————— measured u———————————————————
540          [x_hat(:,i),P_hat, Y_hat,infoIs,obs,iis,dof_dh(i), c_dh(i)]=...
541              est_det_ident_dynamic( infoIs, δ_hat(:,i), P_bar, Y_bar, y_bar,...
542              z, meas,params, net, measurement_variance, ...
543              bad_data_threshold_dynamic_est, identify_threshold_dh, ...
544              nonlinear_SE,dynamic_est_BDfilter, obs, iis, M_δ2);
545
546          %————————————— estimated u———————————————————
547          [xu_hat(:,i),Pu_hat, Yu_hat,infoIs,obs,iis,dof_dhu(i), c_dhu(i)]=...
548              est_det_ident_dynamic( infoIs, δ_hat(:,i),Pu_bar,Yu_bar,yu_bar,...
549              z, meas,params, net, measurement_variance, ...
550              bad_data_threshold_dynamic_est, identify_threshold_dh, ...
551              nonlinear_SE,dynamic_est_BDfilter, obs, iis, Mu_δ2);
552
553
554          % gather data for performance metric
555          ps(i)=iis;
556          pd(i)=sum(1 ./diag(P_hat(n_x-(n_bus-1)+1:n_x, n_x-(n_bus-1)+1:n_x)));
557          pdu(i)=sum(1 ./diag(Pu_hat(n_x-(n_bus-1)+1:n_x, n_x-(n_bus-1)+1:n_x)));
558          if(get_trace_info)
559              pys(i)=trace(infoIs(obs,obs));
560              pyd(i)=trace(M_δ2*Y_hat*(eye(n_x)-M_δ_not2'/...
561               (M_δ_not2*Y_hat*M_δ_not2')*M_δ_not2*Y_hat)...
562               *M_δ2');
563              pydu(i)=trace(Mu_δ2*Yu_hat*(eye(n_x2)-Mu_δ_not2'/...
564               (Mu_δ_not2*Yu_hat*Mu_δ_not2')*Mu_δ_not2*Yu_hat)...
565               *Mu_δ2');
566          end
567
```

```
568
569     end   % for 1=2:n_samples
570
571     %% collect state variance information
572     xs_error=xs_error+(1/num_iter).*(δ_hat(nswngindx,:)-δ_true(nswngindx,:));
573     xs2_error=xs2_error+(1/num_iter).*(δ_hat(nswngindx,:)-δ_true(nswngindx,:)).^2;
574
575     xd_error=xd_error+(1/num_iter).*(M_δ2*x_hat-δ_true(nswngindx,:));
576     xd2_error=xd2_error+(1/num_iter).*(M_δ2*x_hat-δ_true(nswngindx,:)).^2;
577
578     xdu_error=xdu_error+(1/num_iter).* (Mu_δ2*xu_hat-δ_true(nswngindx,:));
579     xdu2_error=xdu2_error+(1/num_iter).* (Mu_δ2*xu_hat-δ_true(nswngindx,:)).^2;
580
581
582     %% collect information matrix trace information
583     if(get_trace_info)
584         for newloop=1:n_samples
585             Ps_emperical(:,:,newloop)=Ps_emperical(:,:,newloop)+(1/num_iter)* ...
586             (δ_hat(nswngindx,newloop)-δ_true(nswngindx,newloop))* ...
587             (δ_hat(nswngindx,newloop)-δ_true(nswngindx,newloop))';
588         end
589
590         newδ=M_δ2*x_hat;
591         for newloop=1:n_samples
592             Pd_emperical(:,:,newloop)=Pd_emperical(:,:,newloop)+(1/num_iter)* ...
593             (newδ(:,newloop)-δ_true(nswngindx,newloop))* ...
594             (newδ(:,newloop)-δ_true(nswngindx,newloop))';
595         end
596         newδ=Mu_δ2*xu_hat;
597         for newloop=1:n_samples
598             Pdu_emperical(:,:,newloop)=Pdu_emperical(:,:,newloop)+(1/num_iter)* ...
599             (newδ(:,newloop)-δ_true(nswngindx,newloop))* ...
600             (newδ(:,newloop)-δ_true(nswngindx,newloop))';
601         end
602
603     end
604
605
606 end
```

```
607
608 % generate various plots
609 [ys,yd,ydu]=make_plots(measurement_variance, n_bus,n_x, ...
610     n_samples, SEtext,  num_iter, ...
611         xs2_error, xd2_error, xdu2_error, ...
612     bus_error_plots, δ_hat, δ_true, x_hat, xu_hat,  ...
613         pd, pdu, ps, ...
614     get_trace_info, Ps_emperical, Pd_emperical, Pdu_emperical, ...
615         pyd, pydu, pys,...
616     detect_bad_data_dynamic,detect_bad_data,dynamic_est_BDfilter,...
617     dof_dh,dof_dhu,dofd,dofs...
618     );
619
620 datestr(now)
621
622
623 end
624
625
626
627 % store data from simulation
628
629    description=['SE14data_10_5_5_1(1 Data linEst nos yesi), 50000 iterations,'...
630                 ' var=0.00001 BDinsNremN redMeas(N)'];
631    disp(description);
632     save SE14_simulation(2011-03-22).mat description SEtext ...
633          xs_error xs2_error xd_error xd2_error ...
634          xdu_error xdu2_error pd  pdu ps ...
635          δ_hat δ_true x_hat x_bar xu_hat xu_bar ;
636
637  if(get_trace_info)
638      save SE14_simulation(2011-03-22)YYY.mat description SEtext ...
639              Ps_emperical Pd_emperical Pdu_emperical pys pyd pydu ys yd ydu;
640   end
```

D.2 det_ident_dynamic.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Perform predicted prefilter for bad data
% Arguments:
%   zd : vector of measurements
%   meas_d : data structure specifying the measurements
%   δ_bar : predicted network state
%   Psi : predicted network state error covariance matrix
%   params: data structure holding various network parameters
%   net : data structure holding the network
%   measurement_variance : variance of the measurement noise (assumes all
%       noise is i.i.d.
%   bad_data_threshold_dynamic : what percentile to detect bad data at
%   identify_threshold_d : how big a measurement residual needs to be (in
%       standard deviations) to be identified as bad.
%   nonlinear_SE : binary flag indicating to use linear approximation or
%       not to find the network state estimate
%   detect_bad_data_dynamic : binary flag indicating if we need to look
%       for bad data or not.
% Returns:
%   zd : filtered measurement vector with bad data removed
%   meas_d : list of what measurements are in zd
%   dofd : degrees of freedom following bad data removal
%   c_d : final χ^2 statistic
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [zd,meas_d,dofd,c_d]=det_ident_dynamic(zd,meas_d,δ_bar,Psi,  ...
    params,net, measurement_variance, ...
    bad_data_threshold_dynamic, identify_threshold_d, ...
    nonlinear_SE,detect_bad_data_dynamic)

n_bus=params.n_bus;
nswngindx=[1:net.swing-1, net.swing+1:n_bus];
n_meas=length(meas_d.iPI)+length(meas_d.iPF);
dofd=n_meas-(n_bus-1);
c_d=0;
if(detect_bad_data_dynamic == true)
```

```
37      bad_done=false;
38      % check to see if bad data are present and remove if necessary
39      while(bad_done==false)
40
41          % determine the degrees of freedom for the χ^2  test
42          n_meas=length(meas_d.iPI)+length(meas_d.iPF);
43          dofd=n_meas-(n_bus-1);
44
45          % get the measurement estimate
46          [zbar,Hd]=measurements(δ_bar,meas_d,params,net,¬nonlinear_SE);
47
48          % calculate the uncertainty of the predicted residuals
49          Psi_bar=Hd(:,nswngindx)*Psi*Hd(:,nswngindx)'+measurement_variance*eye(n_meas);
50
51          %calculate the χ^2 statistic
52          chi2d=sum((zd-zbar).^2./diag(Psi_bar));
53
54          %calculate the percentile for that statistic
55          c_d=chi2cdf(chi2d,dofd);
56
57          % is the statistic outside of our confidence interval?
58          %  if so (and if there are sufficient DOFs, then we have bad data)
59          if(c_d > bad_data_threshold_dynamic && dofd > 1)
60
61              % check measurement residuals to find a bad datum
62              [bad_mag,bad_idx]=max((zd-zbar)./diag(Psi_bar));
63              if((zd(bad_idx)-zbar(bad_idx))/sqrt(Psi_bar(bad_idx,bad_idx))<identify_threshold_d^2)
64                  break  % only identify if more than threshold stdevs away from expected
65              end
66              if(bad_idx > length(meas_d.mPI))
67                  % remove from injection list
68                  meas_d.mPF(bad_idx-length(meas_d.mPI),:)=[];
69                  meas_d.iPF(bad_idx-length(meas_d.mPI))=[];
70              else
71                  % remove from flow list
72                  meas_d.mPI(bad_idx)=[];
73                  meas_d.iPI(bad_idx)=[];
74              end
75
```

```
76              %remove the bad datum
77              zd(bad_idx)=[];
78              n_meas=n_meas-1;
79          else
80              bad_done=true; % we now pass the chi^2 test
81          end
82      end
83  end
```

D.3 est_det_ident_static.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Perform static network state estimation
% If detect_bad_data == true, also detect, identify, and remove the bad
%     data from the measurement vector
% Arguments:
%   zs : vector of measurements
%   meas_s : data structure specifying the measurements
%   params: data structure holding various network parameters
%   net : data structure holding the network
%   measurement_variance : variance of the measurement noise (assumes all
%       noise is i.i.d.
%   bad_data_threshold : what percentile to detect bad data at
%   identify_threshold : how big a measurement residual needs to be (in
%       standard deviations) to be identified as bad.
%   nonlinear_SE : binary flag indicating to use linear approximation or
%       not to find the network state estimate
%   detect_bad_data : binary flag indicating if we need to look for bad
%       data or not.
% Returns:
%   δ_hat : the network state estimate
%   infoIs : the information matrix of the network state estimate
%   obs : a vector indicating which busses are observable
%   iis : theoretic trace of reciprocal covariance matrix
%   dofs : degrees of freedom following bad data removal
%   c_s : final χ² statistic
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [δ_hat,infoIs,obs,iis,dofs, c_s]=est_det_ident_static( ...
    zs, meas_s,params, net, measurement_variance, ...
    bad_data_threshold, identify_threshold, ...
    nonlinear_SE,detect_bad_data)

n_bus=params.n_bus;
n_meas=length(meas_s.iPI)+length(meas_s.iPF);
dofs=n_meas-(n_bus-1);
c_s=0;
```

```
37
38 if(detect_bad_data == false)
39     % don't try to detect bad data, just estimate the network state
40     [δ_hat,infoIs,obs,iis]=SE_δ_static(zs,meas_s, ...
41         params,net,measurement_variance*eye(size(zs,1)),¬nonlinear_SE);
42 else
43     % check to see if bad data are present and remove if necessary
44     bad_done=false;
45     while(bad_done==false)
46         % estimate the state statically using the currently available
47         % measurements
48         [δ_hat,infoIs,obs,iis]=SE_δ_static(zs,meas_s, ...
49             params,net,measurement_variance*eye(size(zs,1)),¬nonlinear_SE);
50
51         % determine the degrees of freedom for the χ^2  test
52         n_meas=length(meas_s.iPI)+length(meas_s.iPF);
53         dofs=n_meas−(n_bus−1);
54
55         % get the measurement estimate
56         zhats=measurements(δ_hat,meas_s,params,net,¬nonlinear_SE);
57
58         %calculate the χ^2 statistic
59         chi2s=sum((zs−zhats).^2)/measurement_variance;
60
61         %calculate the percentile for that statistic
62         c_s=chi2cdf(chi2s,dofs);
63
64         % is the statistic outside of our confidence interval?
65         %  if so (and if there are sufficient DOFs, then we have bad data)
66         if(c_s > bad_data_threshold && dofs > 1)
67
68             % check measurement residuals to find a bad datum
69             [bad_mag,bad_idx]=max(zs−zhats);
70             if((zs(bad_idx)−zhats(bad_idx))/sqrt(measurement_variance)< identify_threshold^2)
71                 break  % only identify if more than _threshold_ stdevs away from expected
72             end
73             if(bad_idx > length(meas_s.mPI))
74                 % remove from injection list
75                 meas_s.mPF(bad_idx−length(meas_s.mPI),:)=[];
```

```
76                  meas_s.iPF(bad_idx-length(meas_s.mPI))=[];
77              else
78                  % remove from flow list
79                  meas_s.mPI(bad_idx)=[];
80                  meas_s.iPI(bad_idx)=[];
81              end
82
83              %remove the bad datum
84              zs(bad_idx)=[];
85          else
86              bad_done=true; % we now pass the χ^2 test
87          end
88      end
89  end
```

D.4 est_det_ident_dynamic.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Perform dynamic network state estimation
% If detect_bad_data == true, also detect, identify, and remove the bad
%     data from the measurement vector
% Arguments:
%   infoIs : the information matrix of the static network state estimate
%   δs : the estimate of the network state
%   P_Bar : predicted state error covariance matrix
%   Y_bar : information matrix (P_bar)^-1
%   y_bar : predicted information  Y_bar*x_bar
%   z : vector of measurements
%   meas : data structure specifying the measurements
%   params: data structure holding various network parameters
%   net : data structure holding the network
%   measurement_variance : variance of the measurement noise (assumes all
%       noise is i.i.d.
%   bad_data_threshold : what percentile to detect bad data at
%   identify_threshold : how big a measurement residual needs to be (in
%       standard deviations) to be identified as bad.
%   nonlinear_SE : binary flag indicating to use linear approximation or
%       not to find the network state estimate
%   detect_bad_data : binary flag indicating if we need to look for bad
%       data or not.
%   obs : indexes of observable states from static estimator
%   iis : performance metric of static estimator
%   M_δ2 : selection matrix to get  network state from dynamic
% Returns:
%   x_hat : the dynamic state estimate
%   P_hat : estimated state error covariance matrix
%   Y_hat : information matrix (P_hat)^-1
%   infoIs : the information matrix of the network state estimate
%   obs : a vector indicating which busses are observable
%   iis : theoretic trace of reciprocal covariance matrix
%   dof_dh : degrees of freedom following bad data removal
%   c_dh : final χ^2 statistic
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
37
38 function [x_hat,P_hat, Y_hat,infoIs,obs,iis,dof_dh, c_dh]=...
39     est_det_ident_dynamic( infoIs, δs, P_bar, Y_bar, y_bar,...
40     z, meas,params, net, measurement_variance, ...
41     bad_data_threshold, identify_threshold, ...
42     nonlinear_SE,detect_bad_data, obs, iis, M_δ2)
43
44 nswngindx=params.nswngindx;
45
46 n_bus=params.n_bus;
47 n_meas=length(meas.iPI)+length(meas.iPF);
48 dof_dh=n_meas-(n_bus-1);
49 c_dh=0;
50 Y_hat=Y_bar+M_δ2'*infoIs(nswngindx,nswngindx)*M_δ2;
51 y_hat=y_bar+M_δ2'*infoIs(nswngindx,obs)*δs(obs);
52 P_hat=inv(Y_hat);
53 x_hat=Y_hat\y_hat;
54
55 if(detect_bad_data == true)
56     % check to see if bad data are present and remove if necessary
57     bad_done=false;
58     δ_hat=zeros(n_bus,1);
59     δ_hat(nswngindx)=M_δ2*x_hat;
60     Psi=M_δ2*P_bar*M_δ2';
61
62     while(bad_done==false)
63
64         % get the measurement estimate
65         [zhat,Hd]=measurements(δ_hat,meas,params,net,¬nonlinear_SE);
66         G=inv(Hd(:,obs)'*Hd(:,obs))*measurement_variance;
67         K=Psi(obs-1,obs-1)*pinv(Psi(obs-1,obs-1)+G);
68         IMK=eye(size(K))-K;
69         V=measurement_variance*eye(length(z));
70         %calculate the χ² statistic
71         Psi_hat=V + Hd(:,obs)*(IMK*(Psi(obs-1,obs-1)+G)*IMK'-G)*Hd(:,obs)';
72         dof_dh=n_meas-length(obs);
73
74 %        chi2d=sum((z-zhat).^2./diag(Psi_hat));
75         chi2d=(z-zhat)'*inv(Psi_hat)*(z-zhat);
```

```
76
77          %calculate the percentile for that statistic
78          c_dh=chi2cdf(chi2d,dof_dh);
79          % is the statistic outside of our confidence interval?
80          %  if so (and if there are sufficient DOFs, then we have bad data)
81          if(c_dh >  bad_data_threshold && dof_dh > 1)
82              % check measurement residuals to find a bad datum
83              [bad_mag,bad_idx]=max(z-zhat);
84              if(  (z(bad_idx)-zhat(bad_idx))/sqrt(measurement_variance)...
85                      < identify_threshold)
86                  break  % only identify if more than _threshold_ stdevs away from expected
87              end
88              if(bad_idx > length(meas.mPI))
89                  % remove from injection list
90                  meas.mPF(bad_idx-length(meas.mPI),:)=[];
91                  meas.iPF(bad_idx-length(meas.mPI))=[];
92              else
93                  % remove from flow list
94                  meas.mPI(bad_idx)=[];
95                  meas.iPI(bad_idx)=[];
96              end
97
98              %remove the bad datum
99              z(bad_idx)=[];
100             n_meas=n_meas-1;
101         else
102             bad_done=true; % we now pass the χ² test
103             break;
104         end
105         % estimate the state statically using the currently available
106         % measurements
107         [δ_hat,infoIs,obs,iis]=SE_δ_static(z,meas, ...
108             params,net,measurement_variance*eye(size(z,1)),¬nonlinear_SE);
109         % determine the degrees of freedom for the χ²  test
110         n_meas=length(meas.iPI)+length(meas.iPF);
111         dof_dh=n_meas-(n_bus-1);
112
113     end
114 end
```

D.5 SE_delta_static.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%SE_δ_static performs static state estimation of the voltage
%angle given the measurements specified (meas), the network (net), and the
%line enable signals (ENBL)
% z is raw measurements, in order PI then PF
% meas consists of meas.PI (vector of busses having measurements) and
% meas.PF (vector of lines having measurements)
% V is the covariance of the measurement noise
%
% returns δ_hat_s, the voltage angles and infoIs, the information
% matrix for the state estimate (infoIs = H^T V^-1 H)
%
% Arguments:
%   z : vector of measurements
%   meas : data structure specifying the measurements
%   params: data structure holding various network parameters
%   V : variance of the measurement noise
%   linear : boolean flag, do estimation linear or nonlinear
% Returns:
%   δ : the network state estimate
%   infoIs : the information matrix of the network state estimate
%   observable : a vector indicating which busses are observable
%   iis : theoretic trace of reciprocal covariance matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [δ,infoIs,observable,iis]=SE_δ_static( ...
    z,meas,params,net,V,linear)

     n_bus=params.n_bus;

    % build the measurement Jacobian
    δ=zeros(n_bus,1); %angle in radians
    hPI=-imag(full(params.Y));
    hPI=hPI(meas.iPI,:);
    hPF=full(params.Y2);
    hPF=hPF(meas.iPF,:);
```

```
37      H=[hPI;hPF];
38      invV=inv(V);
39
40      % check to see which states are observable
41      observable=[1:n_bus];
42      observable(params.swing)=[];
43      if(rank(H(observable))<n_bus-1)
44          obs=observable(1);
45          miss=0;
46          for i=2:length(observable)
47              if( rank(  H(:,[obs,observable(i)])  )==i-miss )
48                  obs=[obs,observable(i)];
49              else
50                  miss=miss+1;
51              end
52          end
53          observable=obs;
54      end
55
56      temp_infoIs=H(:,observable)'*invV*H(:,observable);
57      L=temp_infoIs\H(:,observable)'*invV;
58      temp_δ=L*z;
59  %    iis=sum(1./diag(inv(temp_infoIs)));
60      iis=sum(1./diag(eye(length(observable))/temp_infoIs));
61
62      if(linear==false) %iterate on measurements if nonlinear
63
64          err2=sum(abs(temp_δ));
65
66          δ=zeros(n_bus,1);
67          for correct_i=1:10
68              δ(observable)=temp_δ;
69              z_hat=measurements(δ,meas,params,net,true);
70              old_δ=temp_δ;
71              temp_δ=temp_δ + L*(z-z_hat);%/1.1^(correct_i-1);
72
73              err2=sum(abs(temp_δ-old_δ));
74              if(err2/length(err2)  < 1e-13) %completion criteria
75                  break
```

```
76              end
77          end
78      end
79
80      % estimate only the observable states
81      δ(observable)=temp_δ;
82      infoIs=H'*invV*H;
83  end
```

D.6 measurements.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  measurements.m
%
%Calculates the measurement estimate based on the estimated
%angles given and the measurements specified (meas), the network (net),
% and the line enable signals (ENBL)
%
% Arguments:
%   δ : value for the network state to calculate measurement from
%   meas : data structure specifying the measurements
%   params: data structure holding various network parameters
%   linear : boolean flag, calculate measurements linearly or nonlinearly
% Returns:
%   z : vector of measurement estimates  h(\hat\δ)
%   H : measurement jacobian matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [z,H]=measurements(δ,meas,params,net,linear)

    n_bus=params.n_bus;
    n_branch=params.n_branch;
    nPI=length(meas.mPI);
    nPF=size(meas.mPF,1);
    hPI=-imag(full(params.Y));
    hPI=hPI(meas.iPI,:);
    hPF=full(params.Y2);
    hPF=hPF(meas.iPF,:);
    H=[hPI;hPF];
    if(linear)
        z=H*δ;
    else
        connectivity=full(sparse([1:n_branch,1:n_branch]',...
            [net.branch.i, net.branch.j]', ...
            [-ones(n_branch,1);ones(n_branch,1)],...
            n_branch,n_bus));
        branch_δ=connectivity*δ;
        zPF=1./[net.branch.X]'.*sin(branch_δ);
```

```
37          zPI=connectivity'*zPF;
38
39          zPI=zPI(meas.iPI);
40          zPF=zPF(meas.iPF);
41          z=[zPI;-zPF];
42      end
43
44  end
```

D.7 make_params.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  make_params
%
% Calculates and collects various parameters for use later
%
% Arguments:
%   net : data structure containing the network information
%   enable : which lines are enabled
% Returns:
%   params : collection of parameters stored sparsely to save memory
%
%revisions
% 2 Apr 08: fixed indexing of Admittance matrix from IPSYS when the swing
%           bus is not bus 1 by using Y_ipsys2index().
%14 Mar: fixed problem with b and g (off by a factor of j)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [params]=make_params(net,enable)
    Y=make_Y(net,enable);
    Y2=make_Y2(net,enable);
    n_branch=size(net.branch,2);
    n_bus=size(net.bus,2);
    %initialize parameter vectors
    r=zeros(n_bus*(n_bus+1)/2,1);
    x=zeros(n_bus*(n_bus+1)/2,1);
    b=zeros(n_bus*(n_bus+1)/2,1);
    g=zeros(n_bus*(n_bus+1)/2,1);
    bs=zeros(n_bus*(n_bus+1)/2,1);
    G=zeros(n_bus*(n_bus+1)/2,1);
    B=zeros(n_bus*(n_bus+1)/2,1);
    %vectorize the branch parameters
    for k=1:n_branch
        if(enable(k)==1)
            ni=min(net.branch(k).i,net.branch(k).j);
            nj=max(net.branch(k).i,net.branch(k).j);
            ij=ni+(nj-1)*(nj/2); %vectorized index
            ii=ni+(ni-1)*(ni/2); %vectorized index
```

```
37              jj=nj+(nj-1)*(nj/2); %vectorized index
38              r(ij)=net.branch(k).R;  %branch series resistance
39              x(ij)=net.branch(k).X;  %branch series reactance
40              bs(ij)=net.branch(k).B;  %branch shunt admittance
41              g(ij)=real(1/( r(ij)+j*x(ij))); %branch series conductance
42              b(ij)=imag(1/( r(ij)+j*x(ij))); %branch series susceptance
43              G(ij)=real(Y(ni,nj));
44              G(ii)=real(Y(ni,ni)); %real part of Admittance matrix entry
45              G(jj)=real(Y(nj,nj));
46              B(ij)=imag(Y(ni,nj));
47              B(ii)=imag(Y(ni,ni)); %imaginary part of Admittance matrix entry
48              B(jj)=imag(Y(nj,nj));
49          end
50      end
51
52      params.nswngindx=[1:net.swing-1, net.swing+1:n_bus];
53      params.r=sparse(r);
54      params.x=sparse(x);
55      params.bs=sparse(bs);
56      params.g=sparse(g);
57      params.b=sparse(b);
58      params.G=sparse(G);
59      params.B=sparse(B);
60      params.n_bus=n_bus;
61      params.n_branch=n_branch;
62      params.Y=sparse(Y);
63      params.swing=net.swing;
64      params.Y2=sparse(Y2);
65  end
```

D.8 make_Y.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  make_Y
%
% Calculates the linearized decoupled relationship betwen network state
% and the bus injections (i.e., measurement jacobian for real power
% injection).
%
% Arguments:
%   net : data structure containing the network information
% Returns:
%   Y : real power injection measurement jacobian
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Y=make_Y(net,enable)
% Y=make_Y(net)
%generate the admittance matix
%based on branch pi-model Zseries=(R+jX) and Yshunt=(jB)
%NOTE: does not account for off-nominal transformer values

%revisions
% 3 Apr 08 to account for shunt admittance loads at busses
% 8 Apr 08 to account for parallel lines

connectivity=[[net.branch.I]',[net.branch.J]'];
num_branches=size(connectivity,1);
num_busses=size(net.bus,2);
%echo on
j=sqrt(-1);
Y=zeros(num_busses,num_busses);
for  i=1:num_branches
%    connectivity(i,:)
    if(enable(i)==1)
        y=1/(net.branch(i).R+j*net.branch(i).X);
%        b=j*(net.branch(i).B);
b=0;
        Y(connectivity(i,1),connectivity(i,2))=Y(connectivity(i,1),connectivity(i,2))-y;
        Y(connectivity(i,2),connectivity(i,1))=Y(connectivity(i,2),connectivity(i,1))-y;
```

```
37          Y(connectivity(i,1),connectivity(i,1) )=Y( connectivity(i,1),connectivity(i,1))+y+b/2;
38          Y(connectivity(i,2),connectivity(i,2) )=Y( connectivity(i,2),connectivity(i,2))+y+b/2;
39      end
40  end
```

D.9 make_Y2.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  make_Y2
%
% Calculates the linearized decoupled relationship betwen network state
% and the line flows (i.e., measurement jacobian for real power flows).
%
% Arguments:
%   net : data structure containing the network information
%   enable : which lines are enabled
% Returns:
%   Y2 : real power flow measurement jacobian
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Y2=make_Y2(net,enable)

for i=1:length([net.branch.R])
    net.branch(i).R=0;  % make it lossless
end
swing=net.swing;
n_branch=length([net.branch]);
n_bus=length([net.bus]);
susceptance=1./[net.branch.X]';
connectivity=full(sparse([1:n_branch,1:n_branch]',...
                     [net.branch.i, net.branch.j]', ...
                   [-ones(n_branch,1);ones(n_branch,1)],n_branch,n_bus));

Y2=-sparse(diag(susceptance)*connectivity);
```

D.10 make_plots.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   make_plots.m
%
% Generate various plots based on simulated data
%
% Arguments:
% n_bus : number of busses (static states)
% n_x : number of dynamic states
% n_samples : how many seconds does the simulation run
% SEtext : descriptive text
% xs2_error, xd2_error, xdu2_error : squared estimate error
% bus_error_plots : boolean, plot state errors
% δ_hat : static estimate of network state
% δ_true : true value of network state
% x_hat : estimate of dynamic state (measured load)
% xu_hat : estimate of dynamic state (unmeasured load)
% ps, pd, pdu : theoretic sum inverse variances
% get_trace_info : boolean calculate and plot emperical trace info matrix
% Ps_emperical, Pd_emperical, Pdu_emperical: emperical state error
%               covariance matrices
% pys, pyd, pydu: theoretic log trace of information matrix
% detect_bad_data_dynamic : boolean BD test from predicted X
% detect_bad_data : boolean BD test from static \δ
% dynamic_est_BDfilter: boolean BD test from estimated X
% dofs : degrees of freedom from BD test static
% dofd : dof from BD test dynamic predicted
% dof_dh : dof from BD test dynamic estimate (measured P)
% dof_dhu : dof from BD test dynamic (unmeasured P)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function []=make_plots(n_bus,n_x, n_samples, SEtext, ...
        xs2_error, xd2_error, xdu2_error, ...
    bus_error_plots, δ_hat, δ_true, x_hat, xu_hat,  ...
        pd, pdu, ps, ...
    get_trace_info, Ps_emperical, Pd_emperical, Pdu_emperical, ...
        pyd, pydu, pys,...
    detect_bad_data_dynamic,detect_bad_data,dynamic_est_BDfilter,...
```

```
    dof_dh,dof_dhu,dofd,dofs...
    )

    if(bus_error_plots)

        %% static plots
         figure
         if(n_bus==118)
            which_plots=[3,69,89];
         elseif(n_bus==14)
            which_plots=[3,6,8];
         else
            which_plots=[1,2,3];
         end
         subplot(3,1,1)
         plot(δ_hat(which_plots,:)',':')
         hold on
         plot(δ_true(which_plots,:)')
         title([SEtext,' static', sprintf(', var=%f',measurement_variance)])

         %% dynamic plots
        %figure
        subplot(3,1,2)
        plot(x_hat( which_plots +(n_x-n_bus),:)',':');
        hold on
        %plot(x_bar([which_plots]+(n_x-n_bus),:)','-.');
        plot(δ_true(which_plots,:)')
        title([SEtext,' dynamic', sprintf(', var=%f',measurement_variance)])
        %axis([0 250 -1 1])

        %figure
        subplot(3,1,3)
        plot(xu_hat( which_plots +(n_x-n_bus),:)',':');
        hold on
        % plot(xu_bar([which_plots]+(n_x-n_bus),:)','-.');
        plot(δ_true(which_plots,:)')
        title([SEtext,' dynamic-integrated', sprintf(', var=%f',measurement_variance)])
    end

```

```
76      %% error plots
77      if(bus_error_plots)
78          figure
79          xd2=sum(-log(xd2_error'))/size(xd2_error',1);
80          xdu2=sum(-log(xdu2_error'))/size(xdu2_error',1);
81          xs2=sum(-log(xs2_error'))/size(xs2_error',1);
82
83          subplot(3,1,1)
84          %  plot(  -log([(xd2_error(3-1,:))',(xs2_error(3-1,:))' ]))
85          plot(  -log([(xd2_error(which_plots(1)-1,:))',(xdu2_error(which_plots(1)-1,:))', ...
86                                                      (xs2_error(which_plots(1)-1,:))' ]))
87          title([SEtext,[' -log average square error d1-',num2str(which_plots(1))], ...
88                       sprintf(', var=%f, iter=%d',measurement_variance,num_iter)])
89          legend(sprintf('d1-3 DSE: %f',xd2(which_plots(1)-1)), ...
90                        sprintf('du1-3 DSE: %f',xdu2(which_plots(1)-1)), ...
91                        sprintf('s1-3 SSE: %f',xs2(which_plots(1)-1)));
92
93          subplot(3,1,2)
94          % plot(-log([(xd2_error(2-1,:))',(xs2_error(2-1,:))' ]))
95          plot(-log([xd2_error(which_plots(2)-1,:)',(xdu2_error(which_plots(2)-1,:))', ...
96                                                    xs2_error(which_plots(2)-1,:)' ]))
97          title([SEtext,[' -log average square error d1-',num2str(which_plots(2))], ...
98                      sprintf(', var=%f, iter=%d',measurement_variance,num_iter)])
99          legend(  sprintf('d1-69 DSE: %f',xd2(which_plots(2)-1)), ...
100                     sprintf('du1-69 DSE: %f',xdu2(which_plots(2)-1)), ...
101                     sprintf('s1-69 SSE: %f',xs2(which_plots(2)-1)));
102
103         subplot(3,1,3)
104         % plot(-log([(xd2_error(2-1,:))',(xs2_error(2-1,:))' ]))
105         plot(-log([xd2_error(which_plots(3)-1,:)',(xdu2_error(which_plots(3)-1,:))', ...
106                                                   xs2_error(which_plots(3)-1,:)' ]))
107         title([SEtext,[' -log average square error d1-',num2str(which_plots(3))], ...
108                     sprintf(', var=%f, iter=%d',measurement_variance,num_iter)])
109         legend(  sprintf('d1-89 DSE: %f',xd2(which_plots(3)-1)), ...
110                  sprintf('du1-89 DSE: %f',xdu2(which_plots(3)-1)), ...
111                  sprintf('s1-89 SSE: %f',xs2(which_plots(3)-1)));
112     end
113
114     %% performance metric
```

```
115     figure
116     subplot(2,1,1)
117     plot(log(pd))
118     hold on
119     plot(log(pdu),'g')
120     plot(log(ps),'r')
121     xlabel('time (s)')
122     ylabel('log(\Sigma 1/\sigma^2_i)')
123     %title([SEtext, 'theoretic sum inverse variances', ...
124     %       sprintf(', var=%f',measurement_variance)])
125     title('theoretical performance index')
126     %legend('dynamic (known u)','old dynamic (known u)',...%'dynamic (unknown u)',
127     %    'static')
128
129     subplot(2,1,2)
130     plot(log(sum(1./xd2_error)'),'b')
131     hold on
132     plot(log(sum(1./xdu2_error)'),'g')
133     plot(log(sum(1./xs2_error)'),'r')
134     xlabel('time (s)')
135     ylabel('log(\Sigma 1/\sigma^2_i)')
136     title('emperical performance index')
137     %title([SEtext, 'emperical sum inverse variances', ...
138     %       sprintf(', var=%f',measurement_variance)])
139
140     %% log trace of information matrix
141 if(get_trace_info)
142
143     ys=zeros(1,n_samples);
144     yd=zeros(1,n_samples);
145     ydu=zeros(1,n_samples);
146     for newloop=1:n_samples
147         ys(newloop)=trace(pinv(Ps_emperical(:,:,newloop)));
148         yd(newloop)=trace(pinv(Pd_emperical(:,:,newloop)));
149         ydu(newloop)=trace(pinv(Pdu_emperical(:,:,newloop)));
150     end
151
152     figure
153     subplot(2,1,1)
```

```
154      plot(log(pyd),'b')
155      hold on
156      plot(log(pydu),'g')
157      plot(log(pys),'r')
158      xlabel('time (s)')
159      ylabel('log(tr(Y)')
160      title('theoretical information')
161  %    title([SEtext, 'theoretic log trace of information matrix', ...
162  %          sprintf(', var=%f',measurement_variance)])
163      subplot(2,1,2)
164  %    figure
165      plot(log(ys),'r')
166      hold on
167      plot(log(yd),'b')
168      plot(log(ydu),'g')
169      xlabel('time (s)')
170      ylabel('log(tr(Y)')
171      title('emperical information')
172  %    title([SEtext, ' emperical log trace of information matrix', ...
173  %           sprintf(', var=%f',measurement_variance)])
174  end
175      %%plot degrees of freedom for chi square test (bad data detection)
176      if(detect_bad_data_dynamic || detect_bad_data || dynamic_est_BDfilter)
177          figure;
178          hold on
179          if( dynamic_est_BDfilter)
180              plot(dof_dh' ,'g')
181              plot(dof_dhu' ,'c')
182          end
183          if(detect_bad_data_dynamic )
184              plot(dofd' ,'b')
185          end
186          if( detect_bad_data)
187              plot(dofs' ,'r')
188          end
189          hold off
190      end
191
192  end
```