Three-Phase Linear State Estimation with Phasor Measurements

Kevin David Jones

Thesis submitted to the faculty of Virginia Polytechnic Institute & State University in partial fulfillment of the requirements for the degree of:

Master of Science in Electrical Engineering

Virgilio A. Centeno, Chairman James S. Thorp Jaime De La Reelopez

May 2, 2011 Blacksburg, VA

Keywords: phasors, PMUs, state estimation, topology processing, matrices

©Copyright 2011, Kevin David Jones
Three-Phase Linear State Estimation with Phasor Measurements

by

Kevin David Jones

Abstract

Given the ability of the Phasor Measurement Unit (PMU) to directly measure the system state and the increasing implementation of PMUs across the electric power industry, a natural expansion of state estimation techniques would be one that employed the exclusive use of PMU data. Dominion Virginia Power and the Department of Energy (DOE) are sponsoring a research project which aims to implement a three phase linear tracking state estimator on Dominion’s 500kV network that would use only PMU measurements to compute the system state. This thesis represents a portion of the work completed during the initial phase of the research project. This includes the initial development and testing of two applications: the three phase linear state estimator and the topology processor. Also presented is a brief history of state estimation and PMUs, traditional state estimation techniques and techniques with mixed phasor data, a development of the linear state estimation algorithms and a discussion of the future work associate with this research project.
Acknowledgements

I wish to extend my sincere gratitude to Dr. James Thorp for his help and guidance regarding the work contained in this thesis. I have learned a tremendous amount from him and greatly value the time that we’ve spent working together.

I would also like to thank the rest of the members of my committee, Dr. Virgilio Centeno and Dr. Jaime De La Ree, and the rest of the faculty in the Center for Power & Energy for creating a positive learning environment for myself and the rest of the students. Because of this I believe that it is not the choice of discipline that defines the graduate school experience but the people that you choose surround yourself with. With this said, I would also like to thank my friends and colleagues here at Virginia Tech for their encouragement and companionship.

And finally, I would like to thank my mother, Kathy Jones, for her love, friendship, and constant support of my academic pursuits.
# Table of Contents

*Abstract* ........................................................................................................................................... ii

*Acknowledgements* ......................................................................................................................... iii

*Table of Contents* ............................................................................................................................ iv

*List of Figures* ................................................................................................................................... vii

*List of Tables* ................................................................................................................................... ix

Chapter 1 – Introduction & History ................................................................................................. 1  
  1.1 – History ................................................................................................................................. 2  
    1.1.1 – Background Discussion of State Estimation ......................................................... 2  
    1.1.2 – Background Discussion of PMUs ........................................................................... 4  
  1.2 – Motivation & Objective ..................................................................................................... 4  
  1.3 – Organization of the Thesis ............................................................................................... 5

Chapter 2 – Traditional State Estimation Techniques ....................................................................... 7  
  2.1 – Construction of the System Model ............................................................................... 7  
    2.1.1 – Component Modeling ............................................................................................. 8  
    2.1.2 – The Bus-Admittance Matrix ................................................................................. 10  
  2.2 – Maximum Likelihood Estimation .................................................................................... 12  
  2.3 – WLS State Estimation ........................................................................................................ 13  
    2.3.1 – WLS Algorithm ........................................................................................................ 13  
    2.3.2 – The Measurement Function ............................................................................... 16  
    2.3.3 – The Measurement Jacobian .................................................................................. 17  
    2.3.4 – A Numerical Example........................................................................................... 20  
  2.4 – Inclusion of PMU Data in Traditional State Estimators .................................................. 22  
    2.4.1 – Phasor Measurements Mixed with Traditional Measurements ......................... 22  
    2.4.2 – Phasor Measurements with a Linear Post-Processing Step ................................. 24  
  2.5 – Conclusion .......................................................................................................................... 26

Chapter 3 – The Linear State Estimation Problem .......................................................................... 27  
  3.1 – Introduction to Linear State Estimation ....................................................................... 27  
  3.2 – Linear State Estimation with $\pi$-Equivalent .................................................................. 28
4.3.3 – Code Structure ............................................................... 58
4.3.4 – Updating the System Matrix after a Contingency ................ 60

4.4 – Topology Processor Application in Matlab .......................... 62
   4.4.1 – System & Measurement Indexing .................................. 63
   4.4.2 – Input, Output, & Database Files .................................... 63
      4.4.2.1 – Input & Output .................................................. 63
      4.4.2.2 – Lookup Tables ................................................. 64

4.4.3 – Code Structure ............................................................ 66

4.5 – Conclusion ........................................................................ 68

Chapter 5 – Testing, Results, & Future Work .............................. 69
   5.1 – Testing of the State Estimator Application .......................... 69
      5.1.1 – Creating Three-Phase Unbalanced Impedances ............... 69
      5.1.2 – Generating a Fake Measurement Set ............................ 70
      5.1.3 – Testing the State Estimator ........................................ 73
         5.1.3.1 – State Estimator Testing Procedure ........................ 74
         5.1.3.2 – State Estimator Testing Results ........................... 79
   5.2 – Functionality of the Topology Processor ............................ 84
   5.3 – Discussion of Future Work .............................................. 87
   5.4 – Conclusion ..................................................................... 88

References .................................................................................. 90

Appendix A – Three Phase Linear State Estimation Application ......... 93
Appendix B – Topology Processor Application ............................... 127
Appendix C – WLS State Estimation Matlab Code ............................ 141
# List of Figures

| Figure 2-1 | Two-Port π-Model | 8 |
| Figure 2-2 | Transformer Branch Model | 9 |
| Figure 2-3 | Tap Changing Transformer Model | 9 |
| Figure 2-4 | Shunt Capacitor & Reactor Model | 10 |
| Figure 2-5 | Simple 5-Bus Example Network | 11 |
| Figure 2-6 | Two-Port π-Model | 16 |
| Figure 3-1 | Two-Port π-Model of a Transmission Line | 29 |
| Figure 3-2 | Example 5-Bus System Showing Current Measurements | 31 |
| Figure 4-1 | Single Bus Example | 46 |
| Figure 4-2 | Ring Bus Configuration | 47 |
| Figure 4-3 | Example 5-Bus System with Substation Detail | 48 |
| Figure 4-4 | Series Impedance Input File Example | 54 |
| Figure 4-5 | Shunt Impedance Input File Example | 55 |
| Figure 4-6 | Voltage Measurement Lookup Table | 56 |
| Figure 4-7 | Current Measurement Lookup Table | 57 |
| Figure 4-8 | State Estimator Pseudo-Code | 60 |
| Figure 4-9 | Current Measurement Lookup Table | 65 |
| Figure 4-10 | Breaker Lookup Table | 65 |
| Figure 4-11 | Full Network Bus/Branch Model | 66 |
| Figure 5-1 | Section of Matlab Testing Script (1) | 74 |
| Figure 5-2 | Section of Matlab Testing Script (2) | 74 |
| Figure 5-3 | Section of Matlab Testing Script (3) | 75 |
| Figure 5-4 | Section of Matlab Testing Script (4) | 75 |
| Figure 5-5 | Section of Matlab Testing Script (5) | 75 |
| Figure 5-6 | Section of Matlab Testing Script (6) | 76 |
| Figure 5-7 | Section of Matlab Testing Script (7) | 76 |
| Figure 5-8 | Section of Matlab Testing Script (8) | 77 |
| Figure 5-9 | Section of Matlab Testing Script (9) | 77 |
| Figure 5-10 | Dominion 500kV Network | 78 |
| Figure 5-11 | Estimator Output for 100 Iterations (w/PMU) | 80 |
| Figure 5-12 | Estimator Output & Measurements for 100 Iterations (w/ PMU) | 80 |
| Figure 5-13 | Estimator Output for 100 Iterations (w/o PMU) | 81 |
| Figure 5-14 | Estimator Output 8 for 100 Iterations (w/o PMU) | 81 |
| Figure 5-15 | Comparison of Real Part of Output & Measurements | 82 |
| Figure 5-16 | Comparison of Imaginary Part of Output & Measurements | 83 |
| Figure 5-17 | Comparison of Magnitude of Output & Measurements | 83 |
| Figure 5-18 | Comparison of Angle of Output & Measurements | 84 |
| Figure 5-19 | Example 5-Bus System with Current Measurements | 85 |
| Figure 5-20 | Topology Processor Application Output (Successful) | 86 |
| Figure 5-21 | Topology Processor Application Output (Unsuccessful) | 87 |
List of Tables

Table 2-1  Branch Impedances of Example Network  -------------------------------------------  11
Table 2-2  Measurement Data  ---------------------------------------------------------------  20
Table 2-3  Measurement Data  ---------------------------------------------------------------  21
Chapter 1
Introduction & History

Electric power systems account for a critical part of our society’s energy infrastructure. Over the years we have grown to depend on the near perfect reliability of these systems that have become a necessary part of our everyday lives. When we enter a room, we instinctively reach for the light switch without the slightest concern that it won’t turn on. All of our household appliances, communication devices, and almost all of our tools ranging from construction sites to our offices require electricity for operation. It is not as if we assume electricity will always be available, it is that we believe electricity will always be available.

This kind of reliability doesn’t happen without a great deal of effort from individuals such as electrical engineers and larger bodies such as electric utilities, universities, and government organizations. One of the key aspects to maintaining the reliability of a large system such as the electric power grid is finding a way to provide feedback to those that control it. Finding a way to accurately monitor the system has been the goal of engineers for the majority of the life of our electric grid. If system operators can be provided with appropriate information regarding the conditions of their systems, then they can use that information to make decisions that will improve not only the day-to-day reliability of the system but allow for engineers to plan more effectively for the future.

State estimation is facet of electric power engineering that has evolved out of these needs. Beginning in the 1960’s, engineers began developing ways to monitor their systems from a central control room. They developed communications systems to collect measurement information across the systems and developed system models that could portray the structure of the network. A computer then took in all of this information and computed an optimized portrait of the systems operating conditions called the system state. With this set of information, an adequately trained professional could understand
everything that is going on in the system and make operation and control decisions accordingly.

Many technologies have come along since the inception of state estimation that have improved its performance and drove it to become an integral part of control center operations. Today, Phasor Measurement Unit (PMU) technology will serve as the next step in improving the quality of the estimate of the system state, providing operators with better information to maintain a high level of system reliability. While PMUs are still noticeably more expensive than traditional measurement devices, the gains of monitoring the transmission level part of the system with these newer, time-synchronized, measurement devices stands to outweigh the financial drawbacks.

As part of a project with Dominion Virginia Power and the Department of Energy (DOE), Virginia Tech aims to develop a three phase linear tracking state estimator using exclusively PMU measurements on their 500kV network. This thesis represents the body of work completed by this student for the initial phase of the project.

1.1 History

This section investigates the history and development of the two critical technologies that make up the heart of this thesis. First is the history of state estimation in power systems as the end goal of the Dominion/DOE research project is to develop a linear state estimator on their 500kV network. Second is the Phasor Measurement Unit (PMU) which makes the appropriate types of system measurements so that a linear estimator can be formed.

1.1.1 Background Discussion of State Estimation

The state of a system can be defined as the minimum set of parameters that must be known to fully perceive the operating conditions of the system. When applied to electric power systems, the system state is known as the set of complex voltages at every node in the network [1]. In conjunction with the network topology and impedances (which are constant and known), every line flow or injection can then be calculated. In the past, operators collected a set of incomplete and unsynchronized measurements which
they would subsequently insert into a load flow calculation in an attempt to find the operating point of the system [2]. This was a very inexact approach and most times led to divergence of the load flow equations.

As the size and complexity of the power system grows it becomes exceedingly more important for system operators to know how the system is behaving [3]. This becomes most evident when simple contingencies occur, cascade to neighboring systems and eventually cause large blackouts. In most historical examples, the scale of the blackout could have been reduced if the system operators had had better and more up-to-date information regarding their system’s operating condition. After the Northeast Blackout of 1965, engineers took large steps to overhaul the operator’s load flow and create a more reliable tool for operators to use [2]. The solution to this problem was static state estimation and was first proposed by Fred Schweppe in the 1960’s [4, 5, 6]. This involved collection of voltage magnitudes, current injections, and real and reactive power flows such that there were more measurements than state variables, thus forming an over-determined system. These measurements were assigned a weight based on their accuracy and inserted into ‘load flow-like’ equations which were non-linear and required multiple iterations to arrive at a solution. The solution of this minimization was the system state. Then operators could use this information to make decisions regarding the control and operation of their power system.

Originally, measurement collection times (or scan times) could be very large due to delays in the communication systems and the computation times of the non-linear iterations could also be sizeable [2]. However, over several decades, state estimation has evolved greatly from its infancy as a mathematical curiosity to it’s now central role in the reliable operation and control of power systems [3]. Its effectiveness and functionality have changed and grown as technology has improved; as time has passed, power system operators have been provided with more abundant and more accurate system measurements to keep them informed on the operating conditions of the power grid.
1.1.2 Background Discussion of PMUs

Phasor Measurement Units (PMUs) are digital metering devices that use a DFT algorithm in conjunction with a precisely timed GPS signal to provide synchronized phasor measurements at different locations in the power system [7]. With GPS synchronization, this technology has the ability to synchronize measurements despite the large distances which may separate metering points. While PMUs were originally very costly, the cost associated with these devices has fallen over the last several decades as computer & GPS technology has improved [2]. While the cost of PMUs has decreased, the cost of installation due to security and communication has increased However, it is such that it is a much more justifiable capital expense and a substation control house can now be outfitted with PMUs and measure the bus voltage and line flows out of each feeder in the substation. Older problems that have plagued state estimators such as scan time could be eliminated with the exclusive use of synchronized phasor measurements [2]. Additionally, the traditional non-linear equations that create varying computation times and chance divergence can be replaced by linear equations that simplify the estimator and offer the operator a fresh system state much more frequently. With measurements that are coordinated so precisely in time, the estimation process can easily and practically be expanded to all three phases of the power system opening a completely new look into the operating conditions of the grid.

1.2 Motivation & Objective

The major objective of this thesis is to outline what is being done in the field of state estimation using PMU data exclusively by presenting the work completed thus far on the Dominion/DOE research project. The end goal of the project is to implement a three phase tracking linear state estimator on Dominion’s 500kV network and then use this state estimator to drive several other applications at Dominion’s control center including instrument transformer calibration and intelligent islanding. This thesis presents a portion of the work from the initial phase of the project which includes the initial software development of two of the applications. These are the topology processor, which will take breaker statuses and line current phasors and determine the most up-to-date
topology of the network, and the actual three phase linear state estimator which will use PMU data to determine the state of the system at 500kV.

This thesis discusses the background information on the topic of three phase linear state estimation and then presents the work done towards completing the goals in the initial phase of the Dominion/DOE research project. It begins with a brief history of the significance of power system state estimation followed by the presentation of the traditional form of state estimation. It continues with the development of the linear state estimation equations and then investigates the planned implementation of a three phase linear state estimation on Dominion Virginia Power’s 500kV network and the associated research and software development.

1.3 Organization of Thesis

This thesis is organized in the following way:

Chapter 1: Introduction & History

This chapter begins by first presenting the motivation and objective of the chapter by discussing background information regarding the Dominion/DOE research project and how the work presented in this thesis fits into the scope of the project. It continues by investigating the history of PMU development and state estimation as it has evolved over the decades and finishes by outlining the contents of each chapter.

Chapter 2: Traditional State Estimation Techniques

This chapter presents the mathematical formulation of the algorithm employed by traditional state estimation techniques. It explores system component modeling, maximum likelihood estimation, weighted least squares estimation (including the WLS algorithm and matrix formulation), and a brief discussion concerning statistical robustness of the weighted least squares estimator.
Chapter 3: Linear State Estimation

The chapter discusses the formulation of the positive sequence linear state estimation problem using phasor measurements exclusively. It presents justification for the paradigm shift from traditional non-linear techniques to linear methods using PMU data. The formulation of the state equation is presented using first a simple two-port pi-model and then broken down into several independent types of matrices. Rules for the population of these matrices and the formation of the complete state equation are presented. The chapter concludes with an alternative formulation of state equation which uses only real values instead of complex values.

Chapter 4: Three-Phase Linear State Estimator & Topology Processor

This chapter represents the body of work completed thus far for the Dominion/DOE research project. At the highest level this is the development of the three phase linear state estimation application and the topology processor application in Matlab. It includes the mathematical formulation of the matrices from Chapter 3 in three phases instead of just positive sequence and a discussion on three phase impedances. Additionally, it presents general considerations made for the development of the topology processor.

It continues by demonstrating the considerations taken when developing the Matlab code including: how to index system components and measurements, content and formatting of database files representing the system models, the structure and flow of the code itself, and the input and output of each of the applications.

Chapter 5: Conclusions & Future Work

The chapter concludes the thesis and summarizes the work completed. It concludes by presenting the results from the testing of the Matlab code. Besides the actual results of the testing this also includes several procedures for creating a three phase system model and fake measurement sets that are used to test the code. It also discusses the future work in the Dominion/DOE research project directly related to this and several other ideas for expansion of the topic.
Chapter 2

Traditional State Estimation Techniques

Power system state estimation refers to the collection of a redundant set of measurements from around the system and computing a state vector of the voltage at each observed bus. While technology has improved state estimators and other control center applications over many decades, the fundamental concepts and algorithms behind these proven techniques remain much the same. Measurements which are non-linear functions of the system state are collected and load-flow-like calculations are performed to iteratively determine the most probable system state from the known information. This chapter presents the mathematical basis for traditional state estimation techniques and investigates several reformulations of these algorithms to include phasor measurements in the estimator to improve the quality of the estimate.

2.1 Construction of the System Model

The state of the power system is a function of several things. These include system parameters such as real and reactive power flows, current injections, & voltages which are unknown but measured, network topology, and parameters such as resistance, reactance, and shunt susceptance of transmission lines which is assumed to be known [3].

The measurements are periodically sent into the control center over a SCADA network. However, the transmission line parameters and physical system model are carefully constructed offline prior to implementation. The system model only changes dynamically due to line outages or other contingencies and is typically handled by a topology processor [3]. This section outlines the details associated with constructing the system model for use with traditional non-linear state estimation techniques from individual component modeling to the formulation of the large system matrices.
2.1.1 Component Modeling

The power system is a very complicated network of electrical devices and components. However, for the purposes of state estimation, there are only a few major players that must be considered. Due to the similarity between traditional state estimators and power flow calculations most of the modeling considerations are the same. The components that are the most significant are the transmission lines, shunt capacitors or reactors, transformers [8]. Additionally, topology information is required which dictates how each of these components are connected together to form the electric power network. For this section assume per unit representation of the parameters.

Although there are more complex models of transmission lines that exist, for the purposes of state estimation, the two-port π-model equivalent of the transmission line is used.

\[ V_i \xrightarrow{I_{ij}} r_{ij} \xrightarrow{iX_{ij}} I_{ji} \xrightarrow{V_j} \]

\[ g_i + jb_i \quad g_j + jb_j \]

**Figure 2-1**: Two-port π-model

The transmission line model use in power flow calculations and most state estimation techniques uses only these four parameters to describe the transmission line. The resistance models the real copper losses in the conductor and the inductor models the energy stored in the magnetic field surrounding the conductor. The shunt impedance (usually just the susceptance) models the line charging. However, the shunt impedance is often entirely neglected especially on lower voltage systems [9].
Since it is assumed the models are already in per unit the transformer is modeled in a similar way.

\[ V \]

\[ r_1 \quad jx_1 \quad r_2 \quad jx_2 \quad V \]

\[ r_c \quad jx_m \]

\[ V \]

**Figure 2-2:** Transformer Branch Model

It has a series impedance and a shunt impedance. The resistance of the series impedance models the real losses in the copper coils and the inductance results from the fact that the conductors are arranged in a coil. The shunt impedance includes a real part which results from eddy current losses and an imaginary part (or magnetizing impedance) that results from the hysteresis losses. Both transmission lines and transformers have a sending and receiving end that serve as its connections to two different nodes in the network. Other types of transformers are modeled differently. For example, phase shifting transformers include a phase offset multiplier in the total impedance of the transformer branch.

\[ V \]

\[ r_1 \quad jx_1 \quad r_2 \quad jx_2 \quad V \]

\[ r_c \quad jx_m \]

\[ V \]

**Figure 2-3:** Tap Changing Transformer

Tap changing transformers work similarly though the tap setting, \( a \), may be included as a state variable. They are modeled using series impedance in series with the transformer model [3].
Shunt capacitors and reactors are devices which are installed in the network to serve as reactive power support and voltage control [3]. These are single port models and are connected to a single bus (not like the transmission lines and transformers). These are simply modeled as a shunt susceptance.

2.1.2 The Bus-Admittance Matrix

After the network parameters such as transmission line series and shunt impedances, transformer impedances, and shunt capacitors and reactors have been defined they can be assembled together to construct the network model of the system. This is what is called the admittance matrix or the Y-Bus of the power system. The admittance matrix of a power system takes the following form.

\[
\begin{bmatrix}
Y_{11} & Y_{12} & \cdots & Y_{1N} \\
Y_{21} & Y_{22} & \cdots & Y_{2N} \\
\vdots & \vdots & \ddots & \vdots \\
Y_{N1} & Y_{N2} & \cdots & Y_{NN}
\end{bmatrix}
\begin{bmatrix}
v_1 \\
v_2 \\
\vdots \\
v_N
\end{bmatrix}
= Y \ast V \quad (2.1)
\]

Admittances are used instead of impedances because the admittance matrix can be populated by inspection while the corresponding impedance matrix is extremely difficult to populate by hand. The rules for populating the admittance matrix of a network are derived from Kirchoff’s laws of current injections into a node. Sparing the derivation of the equations, there are two simple rules that can be used to construct the network admittance matrix by inspection.

1. The \(i^{th}\) element of the admittance matrix is the sum of the admittances of all of the lines connected to bus \(i\).
2. The \( ij^{th} \) element of the admittance matrix is the negative of the admittance connecting bus \( i \) to bus \( j \).

There are also several properties of the Y-Bus matrix. This matrix is generally complex in nature, structurally symmetric, very sparse for large networks, and nonsingular provided that each island contains a connection to ground [3]. Consider the following simple power system network to demonstrate the construction of the network model.

![Figure 2-5: Simple 5-bus example network](image)

Given are the impedances of each of the branches of this simple network

<table>
<thead>
<tr>
<th>From Bus</th>
<th>To Bus</th>
<th>R (p.u.)</th>
<th>X(p.u.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0.02</td>
<td>0.06</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>0.01</td>
<td>0.08</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0.05</td>
<td>0.10</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>0.02</td>
<td>0.07</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>0.01</td>
<td>0.07</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>0.03</td>
<td>0.09</td>
</tr>
</tbody>
</table>

Table 2-1: Branch impedances of example network

The bus admittance matrix will then take the following form.

\[
Y = \begin{bmatrix}
6.5 - 27.3i & -5.0 + 15.0i & 0 & -1.5 + 12.3i & 0 \\
-5.0 + 15.0i & 12.7 - 36.2i & -4.0 + 8.0i & 0 & -3.7 + 13.2i \\
0 & -4.0 + 8.0i & 6.0 - 22.0i & 0 & -2.0 + 14.0i \\
-1.5 + 12.3i & 0 & 0 & 4.8 - 22.3i & -3.3 + 10.0i \\
0 & -3.7 + 13.2i & -2.0 + 14.0i & -3.3 + 10.0i & 9.1 - 37.2i
\end{bmatrix}
\] (2.2)

If they were present, the shunt impedances from each of the components are added to the diagonal elements corresponding to each of their respective busses.
2.2 Maximum Likelihood Estimation

The mathematical theory powering state estimation techniques is a statistical method called maximum likelihood estimation. It begins by creating the likelihood function of the measurement vector. The likelihood function is simply the product of each of the probability density functions of each measurement. Maximum likelihood estimation aims to estimate the unknown parameters of each of the measurements’ probability density functions through an optimization [3].

It is commonly assumed that the probability density function for power system measurement errors is the normal (or Gaussian) probability density function.

\[
f(z) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{1}{2} \left(\frac{z - \mu}{\sigma}\right)^2}
\]

(2.3)

where \(z\) is the random variable of the probability density function, \(\mu\) is the expected value, and \(\sigma\) is the standard deviation. This function would yield the probability of a measurement being a particular value, \(z\). Therefore, the probability of measuring a particular set of \(m\) measurements each with the same probability density function is the product of each of the measurements probability density functions, or the likelihood function for that particular measurement vector [3].

\[
f_m(z) = \prod_{i=1}^{m} f(z_i)
\]

(2.4)

where \(z_i\) is the \(i^{th}\) measurement and

\[
[z] = \begin{bmatrix}
z_1 \\
z_2 \\
z_3 \\
\vdots \\
z_m
\end{bmatrix}
\]

(2.5)

Maximum likelihood estimation aims to maximize this function to determine the unknown parameters of the probability density function of each of the measurements.
This can be done by maximizing the logarithm of the likelihood function, $f_m(z)$, or minimizing the weighted sum of squares of the residuals [3]. This can be written as

$$\text{minimize } \sum_{i=1}^{m} W_i r_i^2$$

subject to $z_i = h_i(x) + r_i$ (2.6)

The solution to this problem is referred to as the \textit{weighted least squares} estimator for $x$.

2.3 WLS State Estimation

Power system state estimators use a set of redundant measurements taken from the power system to determine the most likely system state from the given information and assumptions. The state estimator becomes a weighted least squares estimator with the inclusion of the measurement error covariance matrix which serves to weigh the accuracy of each of the measurements. The physical system model information and measurements are part of the equality constraints of the basic weighted least squares optimization and are what make this algorithm specific to power systems. This section presents the solution to the weighted least squares problem and the system matrix formulation including the measurement function matrix and measurement Jacobian matrix.

2.3.1 WLS Algorithm

Several texts present material developing the mathematical algorithms for the weighted least squares estimator [3, 2, 4]. This section and section 2.3.2 follow most closely the derivation presented in [3]. Consider a measurement vector denoted by $z$ containing $m$ number of measurements and a state vector denoted by $x$ containing $n$ number of state variables.

$$[z] = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_m \end{bmatrix}, \quad [x] = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$ (2.7)
Traditional state estimation techniques employ measurement sets which are non-linear functions of the system state vector. These functions are denoted by $h_i(x)$ and can be assembled in vector form as well.

$$
[h(x)] = \begin{bmatrix}
    h_1(x_1 \ x_2 \ x_3 \ \cdots \ x_n) \\
    h_2(x_1 \ x_2 \ x_3 \ \cdots \ x_n) \\
    h_3(x_1 \ x_2 \ x_3 \ \cdots \ x_n) \\
    \vdots \\
    h_m(x_1 \ x_2 \ x_3 \ \cdots \ x_n)
\end{bmatrix} \quad (2.8)
$$

These functions, evaluated at the true system state would yield a measurement set containing the true measurement values. However, all of these measurements each have their own unknown error associated with them denoted by $e$ and shown in vector form.

$$
[e] = \begin{bmatrix}
    e_1 \\
    e_2 \\
    e_3 \\
    \vdots \\
    e_m
\end{bmatrix} \quad (2.9)
$$

The measurement errors are assumed to be independent of one another and have an expected value of zero. The state equation using non-linear functions to relate the system state vector to the set of measurements can now be written in its complete form.

$$
[z] = [h(x)] + [e] \quad (2.10)
$$

From the previous section, the solution to the state estimation problem can be formulated as a minimization of following objective function.

$$
J(x) = \sum_{i=1}^{m} \frac{(z_i - h_i(x))^2}{R_{ii}} \quad (2.11)
$$

This represents the summation of the squares of the measurement residuals weighted by their respective measurement error covariance. This can be rewritten as the following.

$$
J(x) = [z - h(x)]^T[R]^{-1}[z - h(x)] \quad (2.12)
$$
where \( R \) is the covariance matrix of the measurement errors and is diagonal in structure. Each of the diagonal elements is the covariance of its respective measurement and all of the off-diagonal elements are zero because the measurements are assumed to be independent. To find the minimization of this objective function the derivative should be set to zero. The derivative of the objective function is denoted by \( g(x) \).

\[
g(x) = \frac{\partial f(x)}{\partial x} = - \left[ \frac{\partial h(x)}{\partial x} \right]^T [R]^{-1} [z - h(x)] = 0 \tag{2.13}
\]

Let

\[
[H(x)] = \left[ \frac{\partial h(x)}{\partial x} \right] \tag{2.14}
\]

This matrix, \( H(x) \), is called the measurement Jacobian matrix. Ignoring the higher order terms of the Taylor series expansion of the derivative of the objective functions yields an iterative solution known as the Gauss-Newton method.

\[
x^{k+1} = x^k + \left[ [H(x^k)]^T [R]^{-1} [H(x^k)] \right]^{-1} \left[ [H(x^k)]^T [R]^{-1} [z - h(x^k)] \right] \tag{2.15}
\]

It is clear that the only information required to iteratively solve this optimization is the covariance matrix of measurement errors, \( R \), and the measurement function, \( h(x) \). The measurement Jacobian, \( H(x) \) is simply the derivative of the measurement function with respect to the state vector. The measurement function and measurement Jacobian can be constructed using the known system model including branch parameters, network topology, and measurement locations and type. The error covariance matrix should also be constructed prior to the iterations with the accuracy information of the meters installed in the system.

For the first iteration of the optimization the measurement function and measurement Jacobian should be evaluated at flat voltage profile, or flat start. A flat start refers to a state vector where all of the voltage magnitudes are 1.0 per unit and all of the voltage angles are 0 degrees. In conjunction with the measurements, the next iteration of the state vector can be calculated again and again until a desired tolerance is reached.
2.3.2 The Measurement Function

There are many different types of measurements that exist in a scheme such as this. These include real and reactive power bus injections and flows, line current flow magnitudes and bus voltage magnitudes. In order to develop equations to relate the state vector to each of these types of measurements the two-port π-model is assumed for network branches.

\[ P_i = V_i \sum_{j=1}^{N} V_j (G_{ij} \cos \theta_{ij} + B_{ij} \sin \theta_{ij}) \]  \hspace{1cm} (2.16)

\[ Q_i = V_i \sum_{j=1}^{N} V_j (G_{ij} \sin \theta_{ij} - B_{ij} \cos \theta_{ij}) \]  \hspace{1cm} (2.17)

And likewise with real and reactive power flows. The conductance and susceptance in these equations follows the notation of the above two-port π-model.

\[ P_{ij} = V_i^2 (g_i + g_{ij} - V_i V_j (g_{ij} \cos \theta_{ij} + b_{ij} \sin \theta_{ij}) \]  \hspace{1cm} (2.18)

\[ Q_{ij} = -V_i^2 (b_i + b_{ij} - V_i V_j (g_{ij} \sin \theta_{ij} - b_{ij} \cos \theta_{ij}) \]  \hspace{1cm} (2.19)
Additionally, the line current magnitude from bus $i$ to bus $j$ can be expressed as the following.

$$I_{ij} = \frac{\sqrt{P_{ij}^2 + Q_{ij}^2}}{V_i} = \frac{S_{ij}}{V_i}$$ \hspace{1cm} (2.20)

### 2.3.3 The Measurement Jacobian

While the measurement Jacobian is simply the derivative of the measurement function with respect to the state vector, for application purposes it is simpler to construct this matrix from a symbolic representation of the derivative of the measurement function. The measurement Jacobian has the following general structure.

$$[H] = \begin{bmatrix}
\frac{\partial P_{\text{inj}}}{\partial \theta} & \frac{\partial P_{\text{inj}}}{\partial V} \\
\frac{\partial P_{\text{flow}}}{\partial \theta} & \frac{\partial P_{\text{flow}}}{\partial V} \\
\frac{\partial Q_{\text{inj}}}{\partial \theta} & \frac{\partial Q_{\text{inj}}}{\partial V} \\
\frac{\partial Q_{\text{flow}}}{\partial \theta} & \frac{\partial Q_{\text{flow}}}{\partial V} \\
\frac{\partial I_{\text{mag}}}{\partial \theta} & \frac{\partial I_{\text{mag}}}{\partial V} \\
\frac{\partial V}{\partial \theta} & \frac{\partial V}{\partial V} \\
0 & \frac{\partial V_{\text{mag}}}{\partial V}
\end{bmatrix}$$ \hspace{1cm} (2.21)

The order of the measurement vector will correspond to the order of the rows in the measurement function, and therefore, the measurement Jacobian. While the above partitioning is not required, consistency between the measurement vector and these two matrices is important. Similarly, the columns will correspond to the order of the state vector. Once constructed, the Jacobian matrix elements are each non-linear functions of the state variable and are re-evaluated for each iteration of the estimation solution. There are generalized equations for each type of element that may appear inside of this matrix. These can be classified first by measurement type and second by variable with which the derivative has been taken with respect to. First are the partial derivatives of the Jacobian matrix elements corresponding to the real power injection measurements.

$$\frac{\partial P_i}{\partial \theta_i} = \sum_{j=1}^{N} V_i V_j ( - G_{ij} \sin \theta_{ij} + B_{ij} \cos \theta_{ij} ) - V_i^2 B_{ii}$$ \hspace{1cm} (2.22)
Next are the partial derivatives of the Jacobian matrix elements corresponding to reactive power injection measurements.

\[
\frac{\partial P_i}{\partial \theta_j} = V_i V_j (G_{ij} \sin \theta_{ij} - B_{ij} \cos \theta_{ij})
\]

(2.23)

\[
\frac{\partial P_i}{\partial V_j} = \sum_{j=1}^{N} V_j (G_{ij} \cos \theta_{ij} + B_{ij} \sin \theta_{ij}) - V_i G_{ii}
\]

(2.24)

\[
\frac{\partial P_i}{\partial V_j} = V_j (G_{ij} \cos \theta_{ij} + B_{ij} \sin \theta_{ij})
\]

(2.25)

Next are the partial derivatives of the Jacobian matrix elements corresponding to real power flow measurements.

\[
\frac{\partial Q_i}{\partial \theta_j} = \sum_{j=1}^{N} V_i V_j (G_{ij} \cos \theta_{ij} + B_{ij} \sin \theta_{ij}) - V_i^2 G_{ii}
\]

(2.26)

\[
\frac{\partial Q_j}{\partial \theta_j} = -V_i V_j (G_{ij} \cos \theta_{ij} + B_{ij} \sin \theta_{ij})
\]

(2.27)

\[
\frac{\partial Q_i}{\partial V_j} = \sum_{j=1}^{N} V_j (G_{ij} \sin \theta_{ij} - B_{ij} \cos \theta_{ij}) - V_i B_{ii}
\]

(2.28)

\[
\frac{\partial Q_i}{\partial V_j} = V_i (G_{ij} \sin \theta_{ij} - B_{ij} \cos \theta_{ij})
\]

(2.29)

Next are the partial derivatives of the Jacobian matrix elements corresponding to real power flow measurements.

\[
\frac{\partial P_{ij}}{\partial \theta_i} = V_j (g_{ij} \sin \theta_{ij} - b_{ij} \cos \theta_{ij})
\]

(2.30)

\[
\frac{\partial P_{ij}}{\partial \theta_j} = -V_i V_j (g_{ij} \sin \theta_{ij} - b_{ij} \cos \theta_{ij})
\]

(2.31)

\[
\frac{\partial P_{ij}}{\partial V_i} = -V_j (g_{ij} \cos \theta_{ij} + b_{ij} \sin \theta_{ij}) + 2(g_{ij} + g_i) V_i
\]

(2.32)

\[
\frac{\partial P_{ij}}{\partial V_j} = -V_i (g_{ij} \cos \theta_{ij} + b_{ij} \sin \theta_{ij})
\]

(2.33)
Next are the partial derivatives of the Jacobian matrix elements corresponding to reactive power flow measurements.

\[
\frac{\partial Q_{ij}}{\partial \theta_i} = -V_i V_j (g_{ij} \cos \theta_{ij} + b_{ij} \sin \theta_{ij})
\]  \hspace{1cm} (2.34)

\[
\frac{\partial Q_{ij}}{\partial \theta_j} = V_i V_j (g_{ij} \cos \theta_{ij} + b_{ij} \sin \theta_{ij})
\]  \hspace{1cm} (2.35)

\[
\frac{\partial Q_{ij}}{\partial V_i} = -V_j (g_{ij} \sin \theta_{ij} - b_{ij} \cos \theta_{ij}) - 2(b_{ij} + b_i) V_i
\]  \hspace{1cm} (2.36)

\[
\frac{\partial Q_{ij}}{\partial V_j} = -V_i (g_{ij} \sin \theta_{ij} - b_{ij} \cos \theta_{ij})
\]  \hspace{1cm} (2.37)

Next are the partial derivatives of the Jacobian matrix elements corresponding to voltage magnitude measurements. Note that the voltage magnitude is not a function of voltage magnitudes or angles at any bus besides its own.

\[
\frac{\partial V_i}{\partial V_i} = 1, \frac{\partial V_i}{\partial V_j} = 0, \frac{\partial V_i}{\partial \theta_i} = 0, \frac{\partial V_i}{\partial \theta_j} = 0
\]  \hspace{1cm} (2.38, 2.39, 2.40, 2.41)

Next are the partial derivatives of the Jacobian matrix elements corresponding to the current magnitude measurements. For these equations the shunt branch has been ignored.

\[
\frac{\partial I_{ij}}{\partial \theta_i} = \frac{g_{ij}^2 + b_{ij}^2}{l_{ij}} V_i V_j \sin \theta_{ij}
\]  \hspace{1cm} (2.42)

\[
\frac{\partial I_{ij}}{\partial \theta_j} = -\frac{g_{ij}^2 + b_{ij}^2}{l_{ij}} V_i V_j \sin \theta_{ij}
\]  \hspace{1cm} (2.43)

\[
\frac{\partial I_{ij}}{\partial V_i} = \frac{g_{ij}^2 + b_{ij}^2}{l_{ij}} (V_i - V_j \cos \theta_{ij})
\]  \hspace{1cm} (2.44)

\[
\frac{\partial I_{ij}}{\partial V_j} = -\frac{g_{ij}^2 + b_{ij}^2}{l_{ij}} (V_i \cos \theta_{ij} - V_j)
\]  \hspace{1cm} (2.45)
2.3.4 A Numerical Example

Consider the 5 bus system shown in Figure 2-5 whose network data is shown in Table 2-1. A load flow was solved to develop the system state and measurement set and normally distributed errors were added to the measurements. Table 2-2 shows the measurement information for the system including type, location, value, and error covariance. A traditional weighted least squares state estimator was coded in Matlab following the derivation in this document. It can be found in Appendix C.

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
<th>Value</th>
<th>Covariance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$P_{1-2}$</td>
<td>0.3132</td>
<td>0.008</td>
</tr>
<tr>
<td>2</td>
<td>$P_{1-4}$</td>
<td>0.037</td>
<td>0.008</td>
</tr>
<tr>
<td>3</td>
<td>$P_{3-5}$</td>
<td>0.0127</td>
<td>0.008</td>
</tr>
<tr>
<td>4</td>
<td>$P_{4-5}$</td>
<td>0.0004</td>
<td>0.008</td>
</tr>
<tr>
<td>5</td>
<td>$P_1$</td>
<td>-0.0321</td>
<td>0.010</td>
</tr>
<tr>
<td>6</td>
<td>$P_2$</td>
<td>0.052</td>
<td>0.010</td>
</tr>
<tr>
<td>7</td>
<td>$P_3$</td>
<td>0.018</td>
<td>0.010</td>
</tr>
<tr>
<td>8</td>
<td>$Q_{1-2}$</td>
<td>0.9396</td>
<td>0.008</td>
</tr>
<tr>
<td>9</td>
<td>$Q_{1-4}$</td>
<td>0.2960</td>
<td>0.008</td>
</tr>
<tr>
<td>10</td>
<td>$Q_{3-5}$</td>
<td>0.0888</td>
<td>0.008</td>
</tr>
<tr>
<td>11</td>
<td>$Q_{4-5}$</td>
<td>0.0011</td>
<td>0.008</td>
</tr>
<tr>
<td>12</td>
<td>$Q_3$</td>
<td>0.0080</td>
<td>0.010</td>
</tr>
<tr>
<td>13</td>
<td>$Q_4$</td>
<td>-0.2060</td>
<td>0.010</td>
</tr>
<tr>
<td>14</td>
<td>$Q_5$</td>
<td>-0.0040</td>
<td>0.010</td>
</tr>
<tr>
<td>15</td>
<td>$V_1$</td>
<td>1.0121</td>
<td>0.004</td>
</tr>
<tr>
<td>16</td>
<td>$V_2$</td>
<td>1.0028</td>
<td>0.004</td>
</tr>
<tr>
<td>17</td>
<td>$V_5$</td>
<td>1.0011</td>
<td>0.004</td>
</tr>
</tbody>
</table>

Table 2-2: Measurement Data

The initial Jacobian Matrix is as follows.

$$H = \begin{bmatrix}
-15 & 0 & 0 & 0 & 5 & -5 & 0 & 0 & 0 \\
0 & 0 & -12.31 & 0 & 1.54 & 0 & 0 & -1.54 & 0 \\
0 & 14 & 0 & -14 & 0 & 0 & 2 & 0 & -2 \\
0 & 0 & 10 & -10 & 0 & 0 & 3.33 & -3.33 & 0 \\
-15 & 0 & -12.31 & 0 & 6.54 & -5 & 0 & -1.54 & 0 \\
36.21 & -8 & 0 & -13.21 & -5 & 12.77 & -4 & 0 & -3.77 \\
-8 & 22 & 0 & -14 & 0 & -4 & 6 & 0 & -2 \\
5 & 0 & 0 & 0 & 15 & -15 & 0 & 0 & 0 \\
0 & 0 & 1.54 & 0 & 12.31 & 0 & 0 & -12.31 & 0 \\
0 & 0 & 2 & 0 & 0 & 14 & 0 & -14 & 0 \\
0 & 0 & 3.33 & 3.33 & 0 & 0 & 10 & -10 & 0 \\
4 & -6 & 0 & 2 & 0 & -8 & 22 & 0 & -14 \\
0 & 0 & 4.87 & 3.33 & -12.31 & 0 & 0 & 22.31 & -10 \\
3.77 & 2 & 3.33 & -9.11 & 0 & -13.21 & -14 & -10 & 37.21 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} \quad (2.46)$$
The state estimator converges after only a few iterations at the following state vector.

\[
x = \begin{bmatrix}
\theta_1 \\
\theta_2 \\
\theta_3 \\
\theta_4 \\
\theta_5 \\
V_1 \\
V_2 \\
V_3 \\
V_4 \\
V_5 
\end{bmatrix} = \begin{bmatrix}
0 \\
0.6073 \\
0.2120 \\
0.1089 \\
0.1031 \\
1.0373 \\
0.9844 \\
0.9952 \\
1.0087 \\
0.9950 
\end{bmatrix}
\] (2.47)

It can be seen that this is not a great estimate of the known system state. However, it is able to yield values for state variables that are not measured directly. Better accuracy could be achieved if the size of the measurement set increases. An expanded measurement set is shown in Table 2-3.

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
<th>Value</th>
<th>Covariance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(P_{1-2})</td>
<td>0.3132</td>
<td>0.008</td>
</tr>
<tr>
<td>2</td>
<td>(P_{1-4})</td>
<td>0.037</td>
<td>0.008</td>
</tr>
<tr>
<td>3</td>
<td>(P_{2-3})</td>
<td>0.0001</td>
<td>0.008</td>
</tr>
<tr>
<td>4</td>
<td>(P_{2-5})</td>
<td>0.0274</td>
<td>0.008</td>
</tr>
<tr>
<td>5</td>
<td>(P_{3-5})</td>
<td>0.0127</td>
<td>0.008</td>
</tr>
<tr>
<td>6</td>
<td>(P_{4-5})</td>
<td>0.0004</td>
<td>0.008</td>
</tr>
<tr>
<td>7</td>
<td>(P_1)</td>
<td>-0.0321</td>
<td>0.010</td>
</tr>
<tr>
<td>8</td>
<td>(P_2)</td>
<td>0.052</td>
<td>0.010</td>
</tr>
<tr>
<td>9</td>
<td>(P_3)</td>
<td>0.018</td>
<td>0.010</td>
</tr>
<tr>
<td>10</td>
<td>(P_4)</td>
<td>0.000</td>
<td>0.010</td>
</tr>
<tr>
<td>11</td>
<td>(P_5)</td>
<td>-0.037</td>
<td>0.010</td>
</tr>
<tr>
<td>12</td>
<td>(Q_{1-2})</td>
<td>0.9396</td>
<td>0.008</td>
</tr>
<tr>
<td>13</td>
<td>(Q_{1-4})</td>
<td>0.2960</td>
<td>0.008</td>
</tr>
<tr>
<td>14</td>
<td>(Q_{2-3})</td>
<td>0.0002</td>
<td>0.008</td>
</tr>
<tr>
<td>15</td>
<td>(Q_{2-5})</td>
<td>0.0959</td>
<td>0.008</td>
</tr>
<tr>
<td>16</td>
<td>(Q_{3-5})</td>
<td>0.0888</td>
<td>0.008</td>
</tr>
<tr>
<td>17</td>
<td>(Q_{4-5})</td>
<td>0.0011</td>
<td>0.008</td>
</tr>
<tr>
<td>18</td>
<td>(Q_1)</td>
<td>0.3370</td>
<td>0.010</td>
</tr>
<tr>
<td>19</td>
<td>(Q_2)</td>
<td>-0.1313</td>
<td>0.010</td>
</tr>
<tr>
<td>20</td>
<td>(Q_3)</td>
<td>0.0080</td>
<td>0.010</td>
</tr>
<tr>
<td>21</td>
<td>(Q_4)</td>
<td>-0.2060</td>
<td>0.010</td>
</tr>
<tr>
<td>22</td>
<td>(Q_5)</td>
<td>-0.0040</td>
<td>0.010</td>
</tr>
<tr>
<td>23</td>
<td>(V_1)</td>
<td>1.0120</td>
<td>0.004</td>
</tr>
<tr>
<td>24</td>
<td>(V_2)</td>
<td>1.003</td>
<td>0.004</td>
</tr>
<tr>
<td>25</td>
<td>(V_3)</td>
<td>1.0022</td>
<td>0.004</td>
</tr>
<tr>
<td>26</td>
<td>(V_4)</td>
<td>0.9980</td>
<td>0.004</td>
</tr>
<tr>
<td>27</td>
<td>(V_5)</td>
<td>1.001</td>
<td>0.004</td>
</tr>
</tbody>
</table>

Table 2-3: Measurement Data
Again, the state estimator converges after only a few iterations at the following state vector.

\[
x = \begin{bmatrix}
\theta_1 \\
\theta_2 \\
\theta_3 \\
\theta_4 \\
\theta_5 \\
V_1 \\
V_2 \\
V_3 \\
V_4 \\
V_5
\end{bmatrix} = \begin{bmatrix}
0 \\
0.1948 \\
0.2120 \\
0.1146 \\
0.1375 \\
1.0217 \\
0.9984 \\
0.9943 \\
1.0025 \\
0.9950
\end{bmatrix}
\tag{2.48}
\]

### 2.4 Inclusion of PMU Data in Traditional State Estimators

As stated previously, traditional state estimation techniques have improved over the last several decades, however, the fundamental concepts have remained the same. With the increased use of phasor measurement units in substations, engineers and operators have access to new types of measurements. Power system state estimation is one of many applications that can foreseeably benefit from this technological development. Given the idea that PMUs actually measure the system state instead of indirectly estimating it, the idea that inclusion of this type of data in a state estimator could improve the quality of the state estimate is a reasonable one \cite{10}. This section outlines two specific algorithms that include phasor measurements in traditional state estimators. The first method aims to mix the phasor measurements with the traditional measurements and solve in the same manner as before. The second includes the phasor measurements in a linear post-processing step with the output of the traditional state estimator.

#### 2.4.1 Phasor Measurements Mixed with Traditional Measurements

This first method of including phasor measurements in a traditional state estimator does so by mixing the traditional measurements of real and reactive power flows, injections, and voltage and current magnitudes with complex voltage and current
phasors and then proceeding as usual with iterative, non-linear solution. This section follows the derivation presented in [10].

Given two measurement vectors $z_1$ and $z_2$.

\[
[z_1] = \begin{bmatrix}
z_{11} \\
z_{12} \\
z_{13} \\
\vdots \\
z_{1m}
\end{bmatrix} \quad [z_2] = \begin{bmatrix}
z_{21} \\
z_{22} \\
z_{23} \\
\vdots \\
z_{2n}
\end{bmatrix}
\] (2.49, 2.50)

where $z_1$ is comprised of traditional measurements and $z_2$ is comprised of phasor measurements, the two measurement vectors can be vertically concatenated to serve as the measurement vector for this augmented formulation of the traditional state estimator.

Following the model from the previous section, the equality constraint for the optimization becomes the following.

\[
[z] = [z_1] = \begin{bmatrix}
z_1 \\
V_{\text{real}} \\
V_{\text{imag}} \\
I_{\text{real}} \\
I_{\text{imag}}
\end{bmatrix}
\] (2.51)

Where $h_2(x)$ is the non-linear functions relating the system state vector (which is in polar coordinates) to the phasor measurement vector $z_2$. The measurement error covariance matrix for the minimization takes the following form.

\[
[W] = \begin{bmatrix}
W_1 & 0 \\
0 & W_2
\end{bmatrix}
\] (2.53)

where $[W_1]$ is the measurement error covariances from the traditional measurements and $[W_2]$ is the measurement error covariances from the phasor measurements only it has been transformed to reflect the fact that the state vector is in polar form despite the rectangular form of the phasor measurements. The measurement Jacobian matrix is similarly constructed.
Then the weight least-squares solution is formulated similarly.

\[
[H(x)] = \begin{bmatrix} H_1(x) \\ H_2(x) \end{bmatrix} = \begin{bmatrix} \frac{\partial h_1(x)}{\partial x} \\ \frac{\partial h_2(x)}{\partial x} \end{bmatrix}
\]  

(2.54)

\[
[x_{k+1}] = [x_k] + [G(x_k)][H_1^T W_1^{-1}] [z_1 - h_1(x_k)] + [G(x_k)][H_2^T W_2^{-1}] [z_2 - h_2(x_k)]
\]  

(2.55)

Where

\[
[G(x_k)] = [H_1^T(x_k) W_1^{-1} H_1(x_k) + H_2^T(x_k) W_2^{-1} H_2(x_k)]^{-1}
\]  

(2.56)

With this, the algorithm can be computed in the same fashion as the traditional state estimator. The downfall of including phasor measurements in this way is that if one intended to incorporate the phasor measurements into a state estimator that was already in service it would require a tremendous amount of effort to make all of the necessary changes to the code [10]. The next section presents an alternative way to include phasor measurements with traditional state estimation techniques that avoids this dilemma.

### 2.4.2 Adding Phasor Measurements with a Linear Post-Processing Step

An alternative for including phasor measurements in traditional state estimation techniques which does not require manipulation of the current state estimator is investigated in this section. It will be seen in the next chapter how this method mirrors the linear state estimation method which uses phasor measurements exclusively. This section follows the derivation presented in [10].

In order to include the phasor measurements in the estimation process without altering the current state estimator structure is to take the calculated system state from the traditional state estimator use the phasor measurements to enhance the estimate with a linear calculation. First the state estimator must be converted from polar to rectangular coordinates and the associated covariance matrix must be transformed accordingly.
Here, $R$ is a rotation matrix used to transform the covariance matrix to correspond to rectangular coordinates instead of polar coordinates. Then, the calculated system state and the phasor measurement vector could be vertically concatenated and related to the system state by a linear equation.

$$\text{Cov}([x])_{\text{rect}} = [R'][\text{Cov}([x])][R']^T = [W'_1]$$

(2.57)

It can be seen that the system state is identically related to the partition of the measurement vector which contains the calculated system state and identically related to the voltage phasor measurements in the measurement vector for those busses that contain PMUs. The superscript above the identity matrix in the middle partition of the measurement function represents an identity matrix which has rows missing for those state variables that do not have a corresponding voltage phasor measurement. Additionally, there are system parameters in the lower partition of the measurement function that linearly related the system state to the line current phasor measurements in the measurement vector.

Define a new covariance matrix which includes both the error models for the calculated system state and the phasor measurement vector.

$$[W] = \begin{bmatrix} W'_1 & 0 \\ 0 & W'_2 \end{bmatrix}$$

(2.59)

Then, the solution to this over-determined linear equation is as follows.

$$[x^{(3)}] = [A^TW^{-1}A]^{-1}[W^{-1}A][z']$$

(2.60)
2.5 Conclusion

This chapter discussed traditional state estimation techniques and presented the formulation of the weighted least squares solution of a non-linear state estimation algorithm. Despite its flaws, this type of implementation is the most prevalent in electric utilities and has proven itself over many decades. However, PMU technology provides a more accurate and time-sensitive avenue for measurement collection and therefore the inclusion of PMU data in state estimation is a natural evolution of the technology. Two techniques for using PMU data in state estimation were also presented and discussed. In the next chapter, a linear formulation of the state estimation problem using PMU data exclusively will be presented.
Chapter 3

The Linear State Estimation Problem

It was the prevailing idea throughout most of the development of traditional state estimators that the precise simultaneous collection of measurements across the system was something that could never be accomplished. A large assumption that held all traditional state estimation techniques together was that the static state of the power system changed very slowly and operators could afford to have significant scan times. Even though some estimators today have scan times of only a few seconds, this could still be an eternity for several desirable applications in protection and control. PMUs allow for the synchronized collection of phasor measurements and with this technology becoming so prevalent in utilities, it is inevitable that it will be used for state estimation applications [2].

3.1 Introduction to Linear State Estimation

As presented in the previous chapter, the inclusion of PMU technology in state estimation may come in several forms. PMU measurements may be included by a slightly different formulation of the traditional non-linear weighted least squares or they may be taken into consideration after a preliminary system state has already been determined [10]. Even a small number of these precise measurements can weigh heavily on the accuracy of the overall state of the system [2]. However, a true application of PMU technology to state estimation would have all of the traditional measurements of real and reactive power injections and current and voltage magnitudes replaced by bus voltage phasors and line current phasors. If only PMU measurements are used, there are also no complications from the use of both polar and rectangular values in the state estimation process, as would be done when including PMU measurements in traditional state estimators.
If a state estimator could function with only PMU measurements as inputs then many issues associated with traditional state estimators could be resolved. Because PMUs are synchronized with GPS, the problem of scan time becomes irrelevant. One could imagine looking at the state of the power system with a traditional state estimator versus one which used explicitly synchronized PMU data and comparing it to putting on a pair of spectacles for the first time and finally having a focused view of the world. Once the problem of scan time has been erased, the only issue of time is the communication and computational delay between the collection of the measurements and the employment of useful information for decision making by the operation and control applications. Additionally, when using PMUs as metering devices, the state of the system is actually being directly measured. However, estimation is still necessary for including redundancy and bad data filtering. Because of this, the placement of the PMUs is critical for achieving a fully observable system with a sufficient amount of measurement redundancy [11,12]. It will be seen in the next chapter that the redundancy can be gained not only from the line currents (which are linearly related to the system state) but from redundant bus voltage measurements.

This chapter outlines linear state estimation from a single phase perspective. First, the basic mathematical concept of linear state estimation will be presented on a pi-model transmission line to demonstrate how voltage and current measurements are related to the system state. The next section outlines the formulation of the system matrices on a simple network. And finally, the complete linear state estimation equation is presented and solved. In the beginning of the next chapter, the material will be presented again to demonstrate the shift of linear state estimation from the positive sequence world to that which includes all three phases.

### 3.2 Linear State Estimation with π-Equivalent

To understand the fundamental difference between the measurements used in a traditional state estimator and the measurements used in a linear state estimator it is best to begin with a simple two-port π-model equivalent of a transmission line. Phadke and Thorp present the formulation of the linear state estimation problem with the π-model [2].
The state of this simple system will be the voltage magnitude and angle at each end of the transmission line. If there is a PMU at each end of the transmission line then it can be assumed that the measurement set for this system will consists of the voltage phasors at each end of the line and the line flows leaving each end of the line. Recall that because of the capacitance of transmission lines that the line current on each side of a single line will not be the same. Consider the π-equivalent of a transmission line shown in Figure 3-1.

![Figure 3-1: Two-port π-model of a transmission line](image)

All values will be considered to be rectangular because at the most basic level this is what the DFT inside of the PMU will return [13]. This is what gives the state equation its linear property. The system state is then the following complex vector.

\[
[x] = \begin{bmatrix} V_i \\ V_j \end{bmatrix} \quad (3.1)
\]

The error-free measurement set is considered to be the vertical concatenation of the voltage phasors at each end of the transmission line and the line flows from each end of the transmission line.

\[
[z] = \begin{bmatrix} V_i \\ V_j \\ I_{ij} \\ I_{ji} \end{bmatrix} \quad (3.2)
\]

The system state can clearly be related identically to the voltage measurements in this complex vector. However, the linear relationship between the system state and the line flows requires some effort. First, several quantities must be defined. The series admittance and shunt susceptibility of the transmission line are the following.
Sparing the derivation using Kirchoff’s laws, the relationship between the system state and the line current flows on this simple transmission line is as follows.

\[
y_{ij} = (r_{ij} + jx_{ij})^{-1} \quad (3.3)
\]

\[
y_{i0} = g_i + jb_i \quad (3.4)
\]

\[
y_{j0} = g_j + jb_j \quad (3.5)
\]

Then the complete state equation takes the following form.

\[
\begin{bmatrix}
I_{ij} \\
I_{ji}
\end{bmatrix} =
\begin{bmatrix}
y_{ij} + y_{i0} & -y_{ij} \\
-y_{ij} & y_{ij} + y_{j0}
\end{bmatrix}
\begin{bmatrix}
V_i \\
V_j
\end{bmatrix} \quad (3.6)
\]

In the following section, this equation will be broken down into a set of individual matrices that when combined will form the matrix that will relate a measurement set of a power system network to the system state. Sets of simple rules will be presented for the construction of each of these matrices and how to combine them.

### 3.3 Matrix Formulation

Discussed in this section are the rules for populating the matrices used in the state equation for single-phase linear state estimation. A simple fictitious 5 bus system has been used to demonstrate the construction of each of the matrices that are needed. These matrices are called the current measurement-bus incidence matrix, the voltage measurement-bus incidence matrix, the series admittance matrix, and the shunt susceptance matrix. Then, the system matrix relating the system state to the set of voltage and current phasor measurements is calculated using these four matrices. Explicit rules for constructing each of these matrices are included in each section.
3.3.1 Current Measurement-Bus Incidence Matrix

The current measurement-bus incidence matrix is a matrix that shows the location of the current flow measurements in the network. It is an $m \times b$ size matrix where $m$ is the number of current measurements in the network and $b$ is the number of buses which have a current measurement leaving the bus. It is populated using a few simple rules. These include:

1) Each row of the matrix corresponds to a current measurement in the network
2) Each column of the matrix corresponds to a bus in the system which has a current measurement leaving the bus.
3) If measurement $X$(corresponding to row $X$) leaves bus $Y$(corresponding to column $Y$) then the matrix element $(X, Y)$ will be a 1.
4) If measurement $X$(corresponding to row $X$) leaves bus $Y$ heading towards bus $Z$ (corresponding to column $Z$) then the matrix element $(X, Z)$ will be a -1.
5) All remaining entries will be identically zero.

Consider the fictitious 5 bus transmission network shown below. The presence of a current measurement is represented by an arrow above a CT.

![Figure 3-2: Example 5 Bus System Showing Current Measurements](image)

The current measurement bus index matrix for this system in a single phase representation would look like the following.

Figure 3-2: Example 5 Bus System Showing Current Measurements

The current measurement bus index matrix for this system in a single phase representation would look like the following.
3.3.2 Voltage Measurement Bus Incidence Matrix

The voltage measurement-bus incidence matrix is very similar to the current measurement-bus incidence matrix. It shows the relationship between a voltage measurement and its respective location in the network. Its purpose can most easily be understood by inspection of the state equation introduced in the first section of this chapter.

Because PMUs are used to make measurements, the voltage measurements are actually a direct measurement of the system state and therefore only require a simple identity relationship between the measurement and the state. The voltage measurement-bus incidence matrix is an $m \times b$ matrix where $m$ is the number of voltage measurements in the network and $b$ is the number of buses which have a voltage measurement. This matrix is populated using the following rules.

1) Each row of the matrix corresponds to a voltage measurement in the network.
2) Each column of the matrix corresponds to a bus in the system which has a voltage measurement.
3) If measurement $X$ (corresponding to row $X$) is located at bus $Y$ (corresponding to column $Y$) then the matrix element $(X, Y)$ will be a 1.
4) All remaining entries will be identically 0.

Consider again the small fictitious system shown in the previous section. The location of the voltage measurements are shown by the circles on the buses. For a single
phase representation, the voltage measurement-bus incidence matrix would look like the following.

\[
[I] = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]  

(3.9)

This matrix takes the form of the identity matrix because each bus is home to a voltage measurement. It is true that if the voltage measurements or associated state variables were reordered then it would not resemble the identity matrix. Additionally, if redundant voltage measurements were added to the system then columns would contain more than a one non-zero element. This will be seen later in the chapter when we investigate the real world application of this algorithm. What is important to take away from this relationship though is that PMU measurements yield a direct measurement of the system state and are therefore related identically to their respective state variable.

### 3.3.3 Series Admittance Matrix

The series admittance matrix is a diagonal matrix where the diagonal elements are the admittances of the lines being measured. It is an \( m \times m \) matrix where \( m \) is the number of current measurements in the network. It is populated using a single rule:

1) For measurement \( X \), the \((X,X)\) matrix element is the admittance of the branch being measured.

Referring back again to the 5 bus system, the following matrix is the single-phase version of the series admittance matrix of the system where \( y_i \) is the admittance of line \( i \).

\[
[Y] = \begin{bmatrix}
y_1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & y_2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & y_3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & y_4 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & y_5 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & y_6 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & y_7 \\
\end{bmatrix}
\]  

(3.10)
3.3.4 Shunt Admittance Matrix

The shunt admittance matrix is a matrix that relates the location of each current measurement to the shunt admittance of the line that it is measuring. It is an $m \times b$ matrix where $m$ is the number of current measurements in the network and $b$ is the number of the bus where the current being measured originates. It is populated using a single rule:

1) For measurement $X$ (corresponding to column $X$) leaving bus $Y$ (corresponding to row $Y$), the matrix element $(X, Y)$ is the shunt admittance of the side of the line where measurement $X$ was taken.

Referring back again to the 5 bus system, the following matrix is the single-phase version of the shunt admittance matrix of the system where $y_{i0}$ is the shunt admittance of line $i$.

$$
[Y_s] = \begin{bmatrix}
y_{10} & 0 & 0 & 0 & 0 \\
y_{30} & 0 & 0 & 0 & 0 \\
0 & 0 & y_{20} & 0 & 0 \\
0 & 0 & y_{50} & 0 & 0 \\
0 & 0 & 0 & y_{40} & 0 \\
0 & 0 & 0 & 0 & y_{50} \\
0 & 0 & 0 & 0 & y_{60}
\end{bmatrix}
$$

(3.11)

3.3.5 System Matrix Formulation

Consider the following linear state equation.

$$
[z] = \begin{bmatrix} E \\ I \end{bmatrix} = [H] [x] + [e]
$$

(3.12)

It can be seen that the set of measurements, $[z]$ is a vertical concatenation of the set of voltage and current phasor measurements, respectively. The system state, $[x]$ is then related to the set of measurements by a vertical concatenation of the voltage measurement-bus incidence matrix and a system matrix composed of the series and shunt admittance matrices and the current measurement-bus incidence matrix.

$$
[M] = [y][A] + [y_s]
$$

(3.13)
This matrix, $M$, relates the system state to the set of line flow phasor measurements. Then the state equation becomes the following.

$$[z] = [E] = \begin{bmatrix}[II] \end{bmatrix} [x] + [e]$$

(3.14)

Consider again the fictitious 5 bus system present earlier. The system matrix, $M$ will take the following form.

$$M = yA + y_s = \begin{bmatrix}
y_1 + y_{10} & -y_1 & 0 & 0 & 0 \\
y_3 + y_{30} & 0 & 0 & -y_3 & 0 \\
0 & -y_2 & y_2 + y_{20} & 0 & 0 \\
0 & 0 & y_5 + y_{50} & 0 & -y_5 \\
0 & -y_4 & 0 & 0 & y_4 + y_{40} \\
0 & 0 & -y_5 & 0 & y_5 + y_{50} \\
0 & 0 & 0 & -y_6 & y_6 + y_{60}
\end{bmatrix}$$

(3.15)

Then the two matrices are vertically concatenated. In the following equation, the full system state as well as each of the separate voltage and current measurements has been represented as they were in the diagram of the 5 bus system.

$$\begin{bmatrix}V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ I_1 \\ I_2 \\ I_3 \\ I_4 \\ I_5 \\ I_6 \\ I_7\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 \\
y_1 + y_{10} & -y_1 & 0 & 0 & 0 \\
y_3 + y_{30} & 0 & 0 & -y_3 & 0 \\
0 & -y_2 & y_2 + y_{20} & 0 & 0 \\
0 & 0 & y_5 + y_{50} & 0 & -y_5 \\
0 & -y_4 & 0 & 0 & y_4 + y_{40} \\
0 & 0 & -y_5 & 0 & y_5 + y_{50} \\
0 & 0 & 0 & -y_6 & y_6 + y_{60}
\end{bmatrix} \begin{bmatrix}x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} + \begin{bmatrix}e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \\ e_8 \\ e_9 \\ e_{10} \\ e_{11} \\ e_{12}\end{bmatrix}$$

(3.16)
3.3.6 An Alternative Formulation of the Linear State Estimation Equation

Recall the complex linear state equation where \([z]\) is the raw measurement vector of voltage and current phasors, \([x]\) is the system state, \([e]\) is a vector of the error in each of the measurements, and the system matrix is the vertical concatenation of the voltage measurement-bus incidence matrix and a product of several other system matrices.

\[
[z] = \begin{bmatrix} II \\ yA + y_s \end{bmatrix} [x] + [e] \tag{3.17}
\]

In Matlab, dealing with complex numbers is painless. Therefore, this formulation is satisfactory for any implementation of this type. However, it may not necessarily be as easy to deal with the complex numbers in the real-world implementation of a linear state estimator. It is known that PMUs return phasor values as complex numbers (real and imaginary). However, the real and imaginary values of that complex number travel as two separate numbers inside of the communication protocol actually returning the phasor to an application. Because of the way that certain mathematics libraries handle complex values it could be computationally beneficial if the linear state equation could be reformulated to use the real and imaginary parts of the phasor values separately. The following is a presentation of a non-complex formulation of the linear state equation as it is given in [14].

First, define the real and imaginary parts of the system state.

\[
[v_r] = \text{real}(x); \quad [v_x] = \text{imag}(x) \tag{3.18,3.19}
\]

And then, define the real and imaginary parts of the raw measurement vector.

\[
[z_r] = \text{real}(z); \quad [z_x] = \text{imag}(z) \tag{3.20,3.21}
\]

The linear state equation can then be rewritten in the following form.

\[
\begin{bmatrix} z_r \\ z_x \end{bmatrix} = \begin{bmatrix} II & 0 \\ gA + g_s & -bA - b_s \end{bmatrix} \begin{bmatrix} v_r \\ v_x \end{bmatrix} + [\xi] \tag{3.22}
\]
where

\[ gA + g_s = \text{real}(yA + y_s) \]  \hspace{1cm} (3.23)

\[ bA + b_s = \text{imag}(yA + y_s) \]  \hspace{1cm} (3.24)

and \( [II] \) is the exact same voltage measurement-bus incidence matrix formulated for the complex linear state equation.

It is true that the size of the matrices when using the complex equation is exactly one fourth of the size of the matrices with real and imaginary parts separated. At first glance, this might lead one to believe that the computation time for matrix inversion or multiplication would be substantially larger. However, remember that even though complex matrix is smaller in dimensions it has double the values and would require extra computation because the values are complex. Although, when using the formulation with real and imaginary separate, one may be able to avoid quite a bit of unnecessary computation time in the application by avoiding combining and separating complex numbers.

### 3.4 Solution to the Linear State Estimation Problem

The state equation presented in the last subsection is clearly an over-defined set of complex linear equations. Once the solution is obtained, the state can be determined by a simple multiplication of the measurement set with the pseudo-inverse of the system matrix. This allows for calculation of the system state with exactly one iteration for every new set of measurements unlike traditional non-linear techniques which require an unknown number of iterations for each new state [2]. Let the following be true.

\[ [B] = \left[ \begin{array}{c} II \\ yA + y_s \end{array} \right] \]  \hspace{1cm} (3.25)

Then the state equation can be rewritten as follows.

\[ [z] = [B][x] \]  \hspace{1cm} (3.26)
It desired to solve for the system state $[x]$. Because $[B]$ is taller than it is wide, a direct inverse is not achievable. The pseudo-inverse must be determined to solve for the system state $[2]$.

$$[x] = [(B^TB)^{-1}B^T][z] = [H][z] \quad (3.27)$$

This is the solution for an error-free measurement set. If the measurements contain errors then the covariance matrix appears in the solution. The covariance matrix and the introduction of error to the problem is discussed in the next chapter. The solution then takes the following form $[2]$.

$$[x] = [(B^TW^{-1}B)^{-1}B^TW^{-1}][z] = [H][z] \quad (3.28)$$

### 3.5 Conclusion

As a clear application of PMU technology to state estimation, this chapter presented the formulation of the linear state estimation problem using exclusively PMU measurements. A basic formulation using a two-port pi-model is first used followed by discussions of the matrices that are used to develop the system matrix (similar to the measurement function of traditional state estimation techniques). Since the end goal of the Dominion/DOE project is to have these applications implemented in code, this chapter has been structured to provide discrete rules for populating each of the matrix to aid a reader in better understanding accompanying code which is discussed in the following chapter and can be found in Appendix A. The next chapter discusses the application of linear state estimation techniques to all three phases of the power system and the development of the code used in the initial phase of the Dominion/DOE project.
Chapter 4
Three-Phase Linear Tracking State Estimator & Topology Processor

In the last chapter, the concept of linear state estimation was presented as an application of phasor measurement units to power system state estimation. The Dominion/DOE research project intends to apply this concept of linear state estimation to the Dominion 500kV network not just in positive sequence but in all three phases. Because of the ability of the PMU to take such synchronized measurements [15], linear state estimation of all three phases of the power system is a clear expansion of the single phase version [9]. This chapter will present again the concepts introduced in the previous chapter but will expand them to include all of the considerations that must be made when moving from the single phase, positive sequence world to the world of three phase state estimation. The mathematical formulation of this idea will be introduced as well as the practical changes and additions that arise from the desired real world implementation.

The following topics are presented and discussed in this chapter: a discussion of three phase impedance structures and system matrix formulation in three phases, network topology processing, and a discussion of the Matlab implementation of the three-phase linear state estimation application and the topology processor application.

4.1 System Model & Matrix Formulation

There are several differences that arise when formulating the linear state estimation problem in three phases instead of just one. First, the number of state variables will triple since each phase will be its own unique state variable. Second, the properties and structure of each of the system matrices will change slightly. This is due to the third difference which is the branch impedances in the system model are now a three phase representation of the branches. This section outlines some of these differences that arise when switching to three phases.
4.1.1 Three Phase Impedances

One of the most obvious changes when shifting from a positive sequence world to a one which considers all three phases of the power system is the representation of the impedances of each branch in the network. Clearly, each phase of each branch would have its own impedance and when the line is energized the self and mutual inductance of the lines will affect the impedances as well. Because there are three phases, one can take this idea and determine that the impedance of each branch of the power system network should be represented as a 3x3 matrix so that the relationship between current and voltage on that transmission line takes into consideration the effect of the impedance of each conductor as well as the mutual inductance from adjacent conductors.

In an ideal world, the impedance of each of the conductors would be equivalent and the conductors would be spaced equidistantly so that the effects of mutual inductance between phases would affect each phase equally. A 3x3 impedance matrix for an ideal branch like this one would resemble the following where all of the diagonal elements are the same and all of the off-diagonal elements are the same [9].

\[
\begin{bmatrix}
Z_D & Z_0 & Z_0 \\
Z_0 & Z_D & Z_0 \\
Z_0 & Z_0 & Z_D
\end{bmatrix}
\]

(4.1)

However, the impedance of each of the conductors is never exactly the same and the conductor spacing is usually determined by physical constraints of the supporting structures rather than the constraint of desired balanced impedances. Transposition of conductors may somewhat alleviate the unbalancing effects of the mutual inductance from different conductor spacing [3]. However, it is known that the Dominion 500kV network does not transpose its conductors. A branch with unequal conductor spacing such as this would resemble the following [9].

\[
\begin{bmatrix}
Z_{aa} & Z_{ab} & Z_{ac} \\
Z_{ba} & Z_{bb} & Z_{bc} \\
Z_{ca} & Z_{cb} & Z_{cc}
\end{bmatrix}
\]

(4.2)

In this case, the impedance and self-inductance of each phase is different as well as the effects of mutual inductance on each phase. There are not, however, nine different elements in this matrix because the mutual inductance between any two phases is the
same regardless of which phase is referenced. Therefore, there are a maximum of six different values for each branch’s 3x3 impedance matrix. For the purposes of modeling the Dominion 500kV network, six different values comprise each branch impedance 3x3 matrix.

4.1.2 Matrix Formulation

This section is a direct expansion of its Chapter 3 counterpart on matrix formulation for linear state estimation. This section does not cover the rules for populating each of the matrices used it simply demonstrates the differences between single phase and three phase representations of the same matrices. Therefore, Section 3.3 should be referenced if needed. Figure 3-2 has been used to demonstrate the construction of each of the matrices that are needed. These are the same matrices used in the single phase version and they include the current measurement-bus incidence matrix, the voltage measurement-bus incidence matrix, the series admittance matrix, and the shunt admittance matrix. The system matrix is also determined in the same fashion.

4.1.2.1 Current Measurement-Bus Incidence Matrix & Voltage Measurement-Bus Incidence Matrix

Recall the current measurement bus index matrix for the 5 bus system in a single phase representation presented in the previous chapter. It is an \( m \times b \) size matrix where \( m \) is the number of current measurements in the network and \( b \) is the number of busses which have a current measurement leaving the bus.

\[
[A] = \begin{bmatrix}
1 & -1 & 0 & 0 & 0 \\
1 & 0 & 0 & -1 & 0 \\
0 & -1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & -1 \\
0 & -1 & 0 & 0 & 1 \\
0 & 0 & -1 & 0 & 1 \\
0 & 0 & 0 & -1 & 1 
\end{bmatrix} \quad (4.3)
\]
To obtain the three phase version of the current measurement-bus incidence matrix for the same network is a simple task. Each matrix element should be replaced with a 3 x 3 version of itself. The identity matrix will replace all 1’s and a null 3 x 3 will replace all 0’s.

\[ [I] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad [0] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (4.4, 4.5) \]

The three-phase version of the current measurement-bus incidence matrix will then be a 3m x 3b sized matrix. The three phase representation of the current measurement-bus incidence matrix takes the following form.

\[
[A] = \begin{bmatrix}
I & -I & 0 & 0 & 0 \\
I & 0 & 0 & -I & 0 \\
0 & -I & I & 0 & 0 \\
0 & 0 & I & 0 & -I \\
0 & -I & 0 & 0 & I \\
0 & 0 & -I & 0 & I \\
0 & 0 & 0 & -I & I
\end{bmatrix} \quad (4.6)
\]

The three phase representation of the voltage measurement bus incidence matrix is changed in the exact same way. Each matrix element is replaced by a three phase version of itself where 1’s are replaced by 3x3 identity matrices and zeros are replaced by 3x3 null matrices. The three phase representation of the voltage measurement-bus incidence matrix for the simple 5 bus system from the previous chapter is shown below.

\[
[II] = \begin{bmatrix}
I & 0 & 0 & 0 & 0 \\
0 & I & 0 & 0 & 0 \\
0 & 0 & I & 0 & 0 \\
0 & 0 & 0 & I & 0 \\
0 & 0 & 0 & 0 & I
\end{bmatrix} \quad (4.7)
\]
4.1.2.2 Series Admittance Matrix

Recall the single-phase version of the series admittance matrix of the 5 bus system where \( y_i \) is the admittance of line \( i \).

\[
[Y] = \begin{bmatrix}
y_1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & y_3 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & y_2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & y_5 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & y_4 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & y_5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & y_6
\end{bmatrix}
\quad (4.8)
\]

The three-phase version of this matrix is populated in the same manner except each admittance value is a \( 3 \times 3 \) matrix instead of just the positive sequence value. The off-diagonal elements become \( 3 \times 3 \) zero matrices. The resulting matrix is a \( 3m \times 3m \) matrix and is technically no longer a diagonal matrix. A portion of this matrix is shown below.

\[
[Y] = \begin{bmatrix}
y_{1a} & y_{1b} & y_{1c} & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\
y_{1b} & y_{1d} & y_{1e} & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\
y_{1c} & y_{1e} & y_{1f} & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\
0 & 0 & 0 & y_{3a} & y_{3b} & y_{3c} & \ldots & 0 & 0 & 0 \\
0 & 0 & 0 & y_{3b} & y_{3d} & y_{3e} & \ldots & 0 & 0 & 0 \\
0 & 0 & 0 & y_{3c} & y_{3e} & y_{3f} & \ldots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & y_{6a} & y_{6b} & y_{6c} \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & y_{6b} & y_{6d} & y_{6e} \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & y_{6c} & y_{6e} & y_{6f}
\end{bmatrix}
\quad (4.9)
\]

4.1.2.3 Shunt Susceptance Matrix

Recall the single-phase version of the shunt admittance matrix of the 5 bus system where \( y_{io} \) is the shunt admittance of line \( i \). It is an \( m \times b \) matrix where \( m \) is the number of current measurements in the network and \( b \) is the number of the bus where the current being measured originates.
As with the series admittance matrix, the shunt admittances for the three-phase version of this matrix become $3 \times 3$ matrices and the remaining matrix elements become $3 \times 3$ zero matrices. It is a $3m \times 3b$ matrix. A portion of this matrix is shown below.

\[
[Y_s] = \begin{bmatrix}
y_{10} & 0 & 0 & 0 & 0 \\
y_{30} & 0 & 0 & 0 & 0 \\
0 & 0 & y_{20} & 0 & 0 \\
0 & 0 & y_{50} & 0 & 0 \\
0 & 0 & 0 & 0 & y_{40} \\
0 & 0 & 0 & 0 & y_{50} \\
0 & 0 & 0 & 0 & y_{60}
\end{bmatrix}
\] (4.10)

Once each of the matrices has been populated, the system matrix for the state equation is calculated the exact same way as the single phase equivalent.

\[
[Y_s] = \begin{bmatrix}
y_{10a} & y_{10b} & y_{10c} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\
y_{10b} & y_{10d} & y_{10e} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\
y_{10c} & y_{10e} & y_{10f} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\
y_{30a} & y_{30b} & y_{30c} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\
y_{30b} & y_{30d} & y_{30e} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\
y_{30c} & y_{30e} & y_{30f} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots & y_{40a} & y_{40b} & y_{40c} \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots & y_{40b} & y_{40d} & y_{40e} \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots & y_{40c} & y_{40e} & y_{40f} \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots & y_{50a} & y_{50b} & y_{50c} \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots & y_{50b} & y_{50d} & y_{50e} \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots & y_{50c} & y_{50e} & y_{50f} \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots & y_{60a} & y_{60b} & y_{60c} \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots & y_{60b} & y_{60d} & y_{60e} \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots & y_{60c} & y_{60e} & y_{60f}
\end{bmatrix}
\] (4.11)

\[
[z] = \begin{bmatrix}E \\ I \end{bmatrix} = \begin{bmatrix}I \\ II \end{bmatrix} [yA + y_s][x] + [e]
\] (4.12)
4.1.3 Conclusion

This section discussed the structure differences in the mathematical formulation of the linear state equation when shifting from single phase to three phase representation. This is largely due to the structure of three-phase branch impedances. In the next section, a general discussion of topology processing is presented. There are many ways to perform topology processing and many additional techniques that can be included under the umbrella of topology processing. Many of these have been presented in [16]. However, the discussions in the following section only encompass those concerns specific to the Dominion project and the Matlab implementation of the topology processor application.

4.2 Topology Processing

In power system state estimation, the system state is a function of several different items. These include the redundant set of measurements collected throughout the system and their associated error covariances, the network parameters such as branch impedances and the network topology of the system [16]. For example, if line currents or bus voltages read zero because the equipment is out of service for maintenance or due to a contingency it will inadvertently affect the value of the system state if the system model does not reflect these changes. If enough of these types of errors accumulate it could even lead to divergence of the solution in traditional estimation techniques [16]. Therefore, the network topology model used by the state estimator algorithm must change to reflect the actual topology of the network as contingencies occur. The subset of state estimation that is responsible for observing the changes is network topology as the system operates is the topology processor [16]. This section discusses the facets of network topology processing specific to the topology processor application developed for the Dominion/DOE project.

4.2.1 Outage Criteria

The topology processor is responsible for interpreting data collected across the system and determining the current network topology of the system. For the topology processor application developed for the Dominion/DOE project there are two types of
measurements that will be used to decipher the topology of the system. Obviously, breaker statuses are a good indicator of which equipment is out of service and if the breakers at each end of a transmission line are open one can infer that that particular branch is out of service and can be removed from the network model. In addition to the breaker statuses, line current flows can also indicate a branch is out of service. It is reasonable to assume that an out of service line will have no current flow and the associated metering would indicate this. A good question, however, is how to determine what value of the current measurement is sufficiently low to say that it is identically zero. This is not discussed in this thesis but should be a topic of future work for the project as some type of logical filter must be applied to the current measurements before they are interpreted by the topology processor. This is discussed in a later section on code structure of the topology processor. With both the line current flows to indicate an out of service line and breaker statuses to confirm the outages, the topology processor has several layers of redundancy for determining network outages.

### 4.2.2 Bus vs. Substation

Another consideration when implementing a state estimator into a real world situation is the concept of the location of the voltage measurement. Consider the bus shown below.

![Single Bus Example](image)

**Figure 4-1:** Single Bus Example

On paper, this bus would be considered a single bus and a PMU would return a single voltage measurement for that bus/substation. In practice, and in the case with this project, that bus is representing an entire substation with many busses each being
separately measured by the PMU [16]. For example, a common bus configuration at 500kV is the ring bus.

Here there are 4 busses each separated by a breaker and because of this there will be 4 separate voltage measurements to consider. Therefore, when counting measurements or writing state equations, it is important to keep in mind the concept of a bus vs. a substation. In previous sections, concepts have been presented where each substation is represented by a single bus and yields a single state variable. This was done for simplicity in explaining a concept. However, there will actually be many busses at the substation that are measured but still yield only one state variable. There are two ideas that give value to having multiple voltage measurements at a single substation.

Consider first the scenario where all breakers in the ring bus configuration are closed and all four busses are being measured individually. Additionally, all four busses are connected and it can safely be assumed that they are all at the same potential. Therefore, there are three redundant voltage measurements for that same state variable. The increase in redundancy means a much more accurate and robust solution for the state variable in question. State estimation is an application of this fundamental concept so it would only be natural to take advantage of it. The redundancy can also be lost if each bus becomes its own state variable.

To begin the second idea, consider again the same scenario where all four breakers are closed. However, only one of the four busses is being measured. Because all
of the breakers are closed there is still a direct measurement of the state variable in question. However, if the bus that is being measured must be isolated due to some contingency, then the measurement of the state variable is lost and the estimate will be entirely dependent on any line flow measurements if they are available. These are significantly less accurate measurements due to errors in the CTs [17].

It can be seen that the idea of a single bus representing a substation is an incomplete concept for such an application as state estimation. The entire substation configuration must be considered when determining which quantities will be measured and planning the installation of equipment. The robustness of the state estimator depends directly on the available redundancy during different system configurations, particularly after some contingency such as a fault.

Recall the fictitious 5 bus system from Section 3.3. If each line had its own bus at the end so that there were sufficient breakers to isolate each line then the one line diagram might look like the following.

![Figure 4-3: Example 5 Bus System with Substation Detail](image)

Figure 4-3: Example 5 Bus System with Substation Detail

Obviously, the voltage measurement bus incidence matrix would change. Instead of the matrix that is nearly an identity matrix, the columns for each substation will have
multiple values of 1. The same system, then, considering multiple busses at a substation yields the following voltage measurement-bus incidence matrix.

\[
[I] = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]  \tag{4.13}

Where \(I\) is a 3x3 identity matrix and the zeros are each a null 3x3. This now introduces another problem. Since each substation will have many of its own measurements but only a single state variable, it will be very important to consider how all of this quantities will be organized and numbered since the construction of the state equations is dependent on the location of the measurements in the network.

### 4.2.3 Conclusion

This section presented some of the general considerations to be made before beginning the development of the topology processor application in Matlab. Breaker statuses in conjunction with line current flows from PMU measurements are brought into the processor and used to determine the most likely network topology. This information is used to edit the network model used by the estimator algorithm. The following sections present the work done in developing both the three-phase linear state estimation application and the topology processor application in Matlab.
4.3 Three Phase Linear State Estimation Application in Matlab

As part of the initial tasks for the development of the three-phase linear tracking state estimator to be employed on Dominion’s 500kV network, a proof-of-concept was required. Its importance was to demonstrate the functionality of the theory presented in several texts on linear state estimation as well as provide a way to do initial testing and foresee many of the logistical problems that would eventually arise later in the project.

This section and the following sections represent the body of work completed thus far during the initial phase of the Dominion/DOE Three Phase Tracking Linear State Estimator Project. The first phase of the Dominion/DOE research project was to develop Matlab code that would serve as a foundation for the final application code to be written for actual use. This document includes the details regarding the development and testing of the three phase linear state estimator and the topology processor. Initially implementing the applications in Matlab is valuable because it allows for simple coding and testing of the concept. While this is true, it is still important to mirror the Matlab code as closely as possible to the code that will be written in the future. Therefore, it is also true that many of the discussions presented in this section and proceeding sections will mirror the considerations taken in the future when the code is translated into C# for use on the open source platform Dominion will run the completed application on.

There are several issues that are significant for the development of this and other applications. First, the engineers at Dominion desire to be able to edit the code if needed but do not wish to hire expert software engineers to be able to do so. Therefore, simplicity and readability are of significant concern and much time has been taken in order to structure the code as such. In addition, there has been much documentation created to ensure that the engineers will be able to use and make changes to their code and system database. A second issue is with computational speed. The specifications for the three-phase linear tracking state estimator is that it should return a system state every 30\textsuperscript{th} of a second. Therefore, computation time must be taken into consideration when structuring the code. Additionally, input and output of the three-phase linear tracking
state estimator and interaction between its individual applications is extremely important. These applications will be exchanging information quite frequently and it will be important to make certain that the information ends up where it needs to be in the order that it needs to be.

Presented in this section is a brief description of the details of the development of the three phase linear state estimator application in Matlab. A subsection presents detailed information on the formatting and indexing of each of the necessary system information files as well as the formatting of system input and output. The latter is very important because this information will be shared with other applications besides the state estimator such as the topology processor.

The state estimator application was written to function on any network given the appropriately formatted input files. A later section in this chapter discusses the formatting of the input files for the state estimator as well as the format of the input and output of the application.

### 4.3.1 System & Measurement Indexing

The completed application that Dominion will run will include many sub-applications which will share information, inputs, and outputs. All of these must somehow be related to the actual system being monitored and done so in a way that each of the sub-applications may share information coherently. While it is outside the scope of this thesis to define these parameters for each of the sub-applications it is still important for the development of the Matlab code to have a proper way to index system components and measurements. There are several things that must be uniquely identifiable by the code. These are discussed in this section.

The final application will bring in over 1200 different measurements including voltage phasors, current phasors, and breaker statuses and each of these must be uniquely identified by its type and number. Additionally, the location of each of these measurements is critical to appropriately constructing the database files, matrices, and measurement vectors. Therefore, each measurement must have a corresponding location number. For current phasor measurements, the line number and substation of origin must be noted. For voltage phasor measurements, the substation number and bus number must
be noted (recall the difference between a substation and a bus from Section 4.2.2). The breaker status measurements should have a substation number and breaker number. Additionally, the set of indexes for each different parameter must begin at 1 and increment by 1 only. No arbitrary values can be used. For a small example consider again the system shown in Figure 4-3. Each substation, bus, voltage phasor measurement, current phasor measurement, breaker and breaker measurement have been number beginning with 1 and ending with the total number of those particular pieces of information.

4.3.2 Input, Output, & Database Files

Once the system model has been determined and indexed it must be translated into a format that a computer program can easily access and understand. Recall from the section on system and measurement indexing how important it is to correctly identify each component of the system and each individual measurement. This concept carries through to the application of the system by way of the formatting of the database representing the system model and the input files that simulate streaming input.

There are four main database files that represent the system model: the series impedance input file, the shunt susceptance input file, and the voltage measurement and current measurement look-up tables. These four files contain all of the necessary indexing and topology information necessary to construct the system matrices discussed in the previous sections. Additionally, the formatting of the input files which contain the raw measurements from the field must be constructed in a predictable way such that the computer program will interpret them in the same way each time it is read. The output of each of the estimator applications must also be constructed in a similar manner so that the human-machine-interface (HMI) can present the information to the user in a valuable and accurate format.

Presented in this section are examples of each of these types of database, input, and output files. Additionally, examples are provided to accompany a fictitious system that has been used throughout this chapter as a way to demonstrate concepts.
4.3.2.1 Input & Output

The three phase linear state estimator is responsible for taking in system information and determining the most likely state of the system. While the system state is a function of many things, some of these are constant and for the purposes of developing code would not be considered inputs. The inputs to the state estimator are parameters of the system that could change on each iteration of the program. These include synchronized voltage and current phasor measurements from across the system, the most recent network topology information and communication from other.

Another student is developing an application as part of this same research project that aims to calibrate the instrument transformer. Once these errors are known to the estimator application, the covariance matrix which contains the error covariance information for each of the metering devices should be updated in order to provide the most accurate state estimate possible. These error covariances will also serve as an input to the state estimator.

For the purposes of the Matlab version of the state estimator application, the input measurements should be formatted into a column vector in which the measurements have been ordered by their unique measurement index number and are complex in nature. The topology information should be in the bus-branch format [16] and have a column where logic 1 or 0 represents an in service or out of service line respectively. Finally, the error covariances should be contained in a multi-column vector ordered in the same manner as the measurements. One column will correspond to the RCF and the other to the PACF.

The output of the state estimator is a multi-column vector assembled in increasing order of state variable index numbers. One column is for the real part of the voltage and the second column is for the imaginary part.

4.3.2.2 Series Impedance Input File

One of several database files that have to be constructed offline for use by the estimator applications is the series impedance input file. The Matlab implementation of this input file is a *.m file containing a two-dimensional vector that contains the series impedance information for each of the branches in the network. They are indexed by their
Consider the subset of entries shown below. These entries are from one of the branches in the *Figure 3-2*. The full series impedance input file for this example system can be found in *Appendix A* of this document.

\[
\begin{array}{ccc}
\text{Line 4} & 4 & 0.00028157 \ 0.00144660; \\
& 4 & 0.00023118 \ 0.00066119; \\
& 4 & 0.00025302 \ 0.00147850; \\
& 4 & 0.00023871 \ 0.00075234; \\
& 4 & 0.00022445 \ 0.00076813; \\
& 4 & 0.00026677 \ 0.00146360; \\
\end{array}
\]

*Figure 4-4*: Series Impedance Input File Example

The first column in each of the entries is simply the assigned branch number for that line. The second and third column in each of the entries is the resistance and reactance, respectively, of each of the elements in the 3x3 impedance matrix.

There are six entries in this subset of the input file. Recall from the previous section regarding three phase impedances that on this Dominion network, the conductor configuration allows for six different elements in the 3x3 impedance matrix. Each of these entries corresponds to one of those elements. Therefore, their order is extremely important. Shown below is the 3x3 impedance matrix correctly populated using the information above.

\[
\begin{bmatrix}
0.00028157 + j0.00144660 & 0.00023118 + j0.00066119 & 0.00023871 + j0.00075234 \\
0.00023118 + j0.00066119 & 0.00025302 + j0.00147850 & 0.00022445 + j0.00076813 \\
0.00023871 + j0.00075234 & 0.00022445 + j0.00076813 & 0.00026677 + j0.00146360
\end{bmatrix} \quad (4.14)
\]

### 4.3.2.3 Shunt Impedance Input File

Another database file constructed offline for use by the estimator applications is the shunt susceptance input file. It, too, is a *.m* file containing a two dimensional vector with the branch number and reactance of each of the branches in the network. As with the series impedance input file, this file and its formatting are also important to the accurate construction of several of the system matrices discussed previously.
Shown below is a subset of entries from an example input file. These, too, correspond to the example 5 bus system from the previous sections. Additionally, the full example input file can be found in Appendix A.

<table>
<thead>
<tr>
<th>Line 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 0.0477640;</td>
</tr>
<tr>
<td>4 -0.0042036;</td>
</tr>
<tr>
<td>4 0.0460330;</td>
</tr>
<tr>
<td>4 -0.0103150;</td>
</tr>
<tr>
<td>4 -0.0110050;</td>
</tr>
<tr>
<td>4 0.0480700;</td>
</tr>
</tbody>
</table>

**Figure 4-5:** Shunt Impedance Input File Example

As with the series impedance input file, the first column simply represents the branch number for the given line. The second column is the reactance of the shunt susceptance for that line. Note that there is no resistance modeled for the shunt susceptance.

There are also six entries in each subset of this input file for the exact same reasons that the subsets in the series impedance input file contain six entries. The order is still important and follows the same rules as its series impedance counterpart. Shown below is a correctly populated shunt susceptance matrix for line 4 of example 5 bus system.

\[
\begin{bmatrix}
  j0.0477640 & -j0.0042036 & -j0.0103150 \\
  -j0.0042036 & j0.0460330 & -j0.0110050 \\
  -j0.0103150 & -j0.0110050 & j0.0480700
\end{bmatrix}
\]  

(4.15)

4.3.2.4 Voltage Measurement Lookup Table

The voltage measurement look-up table serves as a database of all of the necessary information for the state estimator and the topology processor to determine the location of each of the voltage measurements as they are brought in from the input file. The actual implementation will have these measurements streaming in from the data concentrator. However, once a full set of measurements has been input from the data
concentrator they will be passed as a full set of measurements to the desired applications. Therefore, the Matlab implementation uses a *.m file to serve as an example.

This particular lookup table contains three different values which collectively present a sufficient amount of information that all of the applications that need to know the location of the voltage measurements could determine them. These three categories are the individual voltage measurement number, the number of the bus where the measurement was taken and the number of the substation where that bus is located. This information, in the format given, is slightly redundant. Recall from the section on system & measurement indexing that each bus in the system has its own unique identifier, not each bus in each substation. Therefore, the substation number is redundant for determine the location of the voltage measurement. Despite this fact, its inclusion in the table is still important because it makes writing the troubleshooting code and error messages much simpler and allows for one less look-up table.

Shown below is an example voltage measurement look-up table.

```matlab
%               Measurement #  Bus #  Substation #
lookup_table = [ 1    1      1;
                 2    2      1;
                 3    6      3;
                 4    7      3;
                 5    8      5;
                 6    9      5;
                 7   10      5];
```

**Figure 4-6:** Voltage Measurement Lookup Table

Recall from the sections on matrix formulation the example 5 bus system where the individual busses inside the substation have been taken into consideration. This voltage measurement look-up table corresponds exactly to this fictitious system. One can use this table, the example system and the rules presented in the section on measurement indexing to understand how the table is constructed.
4.3.2.5 Current Measurement Lookup Table

The current measurement look-up table also serves as a database of all of the necessary information for state estimator and topology processor to determine the location and direction of the current measurements in the network. This particular look-up table has significantly more items that allow for sufficient identification of current measurements from the input file.

There are six categories in the current measurement lookup table. The categories for this table include the measurement number, the number of the bus where the current measurement originates and the number of the bus where it extends to, the number of the substation where the current measurement originates from and the number of the substation where it extends to, and the number of the line on which the measurement is located. As with the voltage measurement look-up table there is some redundancy. It would be possible to determine the location of the measurement with only three of these pieces of information: the measurement number, the line number, and either the ‘from’ or ‘to’ bus. However, there is added value in this redundancy for the same reason as with the voltage measurement look-up table.

Below is an example current measurement look-table for the example 5 bus system used previously.

```
lookup_table = [
  1  1  3  1  2  1;
  2  2 11  1  4  3;
  3  6  4  3  2  2;
  4  7  9  3  5  5;
  5  5  5  5  2  4;
  6  9  7  5  3  5;
  7 10 12  5  4  6];
```

**Figure 4-7:** Current Measurement Lookup Table

Again, recall from the sections on matrix formulation the example 5 bus system where the individual busses inside the substation have been taken into consideration. This current measurement look-up table corresponds exactly to this fictitious system. One can use this table, the example system and the rules presented in the section on measurement indexing to understand how the table is constructed.
4.3.3 Code Structure

Put most simply, the three-phase state estimator application is responsible for taking in raw phasor measurements from the system and performing some mathematics to determine the estimate of the system state. However, it must also take into consider the network configuration given by the topology processor application (the topology processor application and its structure are part of a later section of this thesis) and calibration information from the calibration application being developed by another student.

The Matlab implementation of the three-phase state estimator does not allow for exact timing or synchronicity. This is due to several reasons. First, as a Matlab programmer, one does not have control over how Matlab requests processing time from its host computer. Therefore, real-time testing is next to impossible. What has to happen is that a single iteration of the estimator is written as a Matlab function. Then, either some or all of the function is executed based on several logical inputs or flags to the function indicating the operating state of the system.

There are several functions that make up the state estimator application. The first set of functions is responsible for the initial population of the matrices which make up the system matrix. This would only be executed the first time the function is called or if there have been a sufficient number of topology changes to warrant a repopulation of the system matrix due to numerical precision. A second function calculates the pseudo-inverse of the system matrix while taking into consideration the covariance matrix containing the information about the measurement errors. A third function would take the result of the second function and update its elements to reflect any topology change or loss of measurement without having to repopulate all of the matrices and take the pseudo-inverse again. A fourth function is responsible for updating the covariance matrix every time the application is recalibrated. And finally, a function which takes the measurement vector and multiplies it by the pseudo-inverse to obtain the system state. This function is the only function that will execute on every iteration.

Given this information, the Matlab version of the three-phase state estimator application can be explained to function in one of the following four ways.
1) The system is in steady state and the application takes the stored value of the pseudo-inverse from memory and multiplies it by the latest set of data to estimate the system state.

2) A change is observed in the topology of the system and the pseudo-inverse is updated to reflect these changes. After a sufficient number of updates of the pseudo-inverse, however, the entire matrix must be repopulated from scratch. Then the application will take the new pseudo-inverse and multiply by the latest set of data to estimate the system state.

3) The CTs or PTs are recalibrated and a new covariance matrix is populated. The pseudo-inverse must then again be recalculated. As before, the application will take the latest set of measurements and estimate the state of the system.

4) An overlap of events 2 & 3.

The repopulation of the system matrix (either after a change) or while the whole application is offline can be broken down into several functions and has been done so to modularize the code and increase readability. In a previous section, four matrices were described that when combined in certain fashion, the system matrix would be formed. These matrices include, the current measurement-bus incidence matrix, the voltage measurement-bus incidence matrix, the series admittance matrix and the shunt susceptance matrix. A Matlab function was written for the population of each of these matrices based on the rules outlined in Chapter 3 discussing linear state estimation only in a three-phase form as discussed earlier in this chapter.

getDVPincidence_matrix(current_measurement_info)
getDVPII_matrix(voltage_measurement_info)
getDVPseries_matrix(current_measurement_info,series_info)
getDVPshunt_matrix(current_measurement_info,shunt_info)

In the Matlab application, the function arguments contain the relevant system information for populating each of these matrices. In the real application, the functions
will request information from a database stored on a computer with non-volatile memory for the storage of all of the system information. The Matlab code also includes output messages that display on successful completion of each part of the estimator application. All of the code associated with this application can be found in Appendix A. The pseudo code of the Matlab version of the three phase state estimator application is shown below.

```matlab
	topology_info = input_from_topology_processor
	z = new_measurement_set

	if (first_time_running) {
		[A] = getDVP:incidence_matrix(database_files);
		[II] = getDVP:II_matrix(database_files);
		[Y] = getDVP:series_matrix(database_files);
		[YS] = getDVP:shunt_matrix(database_files);
		[K] = [II; Y*A*YS];
		[W] = getDVP:covariance_matrix(error_info);
		[W, 1] = inverse(W);
		[M] = computePseudo_Inverse(K, W);
	}

	if (calibration_has_changed) {
		[W] = getDVP:covariance_matrix(error_info);
		[W, 1] = inverse(W);
		[M] = computePseudo_Inverse(K, W);
	}

	if (network_has_changed) {
		[M] = updateDVP:system_matrix(M);
		if (network_has_changed >> 1) {
			first_time_running = 1;
		}
	}

	[OUTPUT] = M*z;
```

**Figure 4-8:** State Estimator Pseudo-Code

### 4.3.4 Updating the System Matrix after a Contingency

It could be quite cumbersome and time consuming for the state estimator to repopulate the system matrix and take the pseudo-inverse each time there is a change in network topology. Therefore, for computational efficiency, it would be beneficial to simply update certain elements of the pseudo-inverse of the system matrix instead of
recalculating it altogether. Presented in this section is a method used to update this matrix after a line (measurements) are lost due to some contingency [7]. First, recall the linear state equation.

$$[z] = [E] = \begin{bmatrix} H \\ yA + y_s \end{bmatrix} [x] + [e]$$

(4.15)

Then, the solution to this over-defined set of linear equations is known to be the following.

$$[x] = \left( H^T H \right)^{-1} H^T [z] = [M][z]$$

(4.16)

The following is then also true.

$$\hat{z} = H(H^T H)^{-1} H^T [z] = [Z][z]$$

(4.17)

Now, assuming a line with current measurements numbered $l$ and $m$ trips. Define a matrix, $[K]$, which has the same dimensions as $[H]$ and is zero except for two $3x3$ partitions which will contain identity matrices. The location of the $3x3$ partitions is determined by the current measurement numbers, $l$ and $m$, and their respective locations in the network. Let $b_l$ be the bus number where current measurement $l$ originated and $b_m$ be the bus number where current measurement $m$ originated.

$$K(3l - 2: 3l, 3b_l - 2: 3b_l) = [I]$$

(4.18)

$$K(3m - 2: 3m, 3b_m - 2: 3b_m) = [I]$$

(4.19)

Next, define a matrix, $S$, which will have the following generic structure.

$$[S] = [K^T Z K] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & S_1 & 0 & S_3 \\ 0 & 0 & 0 & 0 \\ 0 & S_3 & 0 & S_2 \end{bmatrix}$$

(4.20)

These partitions of $S$ can be symbolically defined as elements of $Z$.

$$S_1 = Z(3l - 2: 3l, 3l - 2: 3l)$$

(4.21)

$$S_2 = Z(3m - 2: 3m, 3m - 2: 3m)$$

(4.22)
Next, define a matrix, $T$ where

\[
[T] = [I - S]^{-1} = \begin{bmatrix}
I & 0 & 0 & 0 \\
0 & I - S_1 & 0 & -S_3 \\
0 & 0 & I & 0 \\
0 & -S_3 & 0 & I - S_2
\end{bmatrix}^{-1} = \begin{bmatrix}
I & 0 & 0 & 0 \\
0 & U_1 & 0 & -U_3 \\
0 & 0 & I & 0 \\
0 & -U_3 & 0 & U_2
\end{bmatrix}^{-1}
\] (4.24)

The partitions of $T$ can also be symbolically defined.

\[
T_3 = (U_3 - U_2 U_3^{-1} U_1)^{-1}
\] (4.25)

\[
T_1 = T_3 U_2 (U_3)^{-1}
\] (4.26)

\[
T_2 = T_3 U_1 (U_3)^{-1}
\] (4.27)

Then, these matrices can be used to update the pseudo-inverse of the system matrix.

\[
[M_1] = [M][I - KTK^T Z][I - KK^T]
\] (4.28)

A function was written in Matlab to serve the purpose of updating the pseudo-inverse of the system matrix. This can be found in Appendix A.

### 4.4 Topology Processor Application in Matlab

The second Matlab application developed as part of the initial phase of the Dominion/DOE research project is the network topology processor. This application serves to interpret raw measurements including line flows and breaker statuses and determine the most probable network model. This information is then passed to many other applications being developed including the three phase linear state estimator. This section presented the considerations made during the development of the topology processor application in Matlab and describes the appropriate format for the files containing the system information.
The topology processor application was written to function on any network given the appropriately formatted input files. A later section in this chapter discusses testing procedures and results are presented and discussed.

### 4.4.1 System & Measurement Indexing

The idea presented previously on indexing each parameter in the state estimator application carries over to the topology processor application. Each of the different parameters should be uniquely indexed. Current measurements, breaker statuses, line numbers, and bus/substation numbers should be indexed appropriately and correspond to the indexing employed by any other applications using the same type of information. *Appendix B* can be referenced for examples of how this was done in the Matlab version of the applications.

### 4.4.2 Input, Output, & Database Files

The topology processor application functions similarly to the state estimator application in that it has specifically formatted inputs, outputs, and lookup tables containing necessary system information. The inputs to the topology processor are the line current phasors and the breakers statuses from across the network. The output of the topology processor includes the most recent bus/branch model of the network and output messages indicating inconsistencies in the data. There are three database files containing system information. First, there is a current measurement lookup table and a breaker location lookup table. Additionally, the bus/branch model of the full network is included. This section discusses the format and content of these files.

#### 4.4.2.1 Input & Output

The topology processor is responsible for interpreting system data and determining the most recent network model of the system. The application takes as input breaker statuses and line current phasors from across the network. The current
measurements come in as the same format they do in the state estimator application: a multi-column vector containing the measurement number and the measurement value in the first and second column respectively. The breaker statuses are arranged in a similar format that includes the breaker number and associated breaker status as a 1 or a 0.

The output of the topology processor is then a bus/branch model of the network which includes an extra column for the service status of each line. The dimensions vector containing the bus/branch model will not change based on topology, only the last column changes to reflect line outages. Examples of each of these can be found in Appendix B.

4.4.2.2 Lookup Tables

The database files used by the topology processor application include a current measurement lookup table, a breaker location lookup table, and a bus/branch model describing the full network. This section describes the composition of each of these database files and provides examples.

The current measurement lookup table for the topology processor is not as detailed as the current measurement lookup table for the state estimator. This lookup table only contains the current measurement numbers and the location of each of those current measurements in the network. The location is given as the line number the measurement originated from. Below is an example of the current measurement lookup table for the topology processor application. This table corresponds to the simple 5 bus system in Figure 4-3.
The breaker location lookup table is slightly more complicated. It is organized by the line numbers of each branch in the network. Following each line number is a series of breaker numbers which all must be ‘open’ in order for the line to be out of service. For this application it has been assumed that no more than four breakers would be required to open any particular line. Other breaker configurations in the substation could yield lesser numbers. In this case, the unfilled slots in the input file should be filled with zeros. Below is an example of the breaker lookup table for the topology processor application. This also corresponds to the simple 5 bus system in Figure 4-3.

```
lookUp_table = [ 
    % Current Meas  Line Number 
    1             1;
    2             1;
    3             2;
    4             2;
    5             3;
    6             3;
    7             4;
    8             4;
    9             5;
    10            5;
    11            6;
    12            6];
```

**Figure 4-9:** Current Measurement Lookup Table

```
lookUp_table = [ 
    % Line Number  BS1  BS2  BS3  BS4 
    1             1    2    4    5;
    2             4    6    7    8;
    3             2    3    10   11;
    4             5    6    13   14;
    5             8    9    13   15;
    6             11   12   14   15];
```

**Figure 4-10:** Breaker Lookup Table
Finally, it is valuable for the topology processor to have access to the bus/branch model of the full network. This is contained in another database file and is formatted as follows.

```
full_network = [
    % Line Number  From Bus  To Bus  Status
    1             1         2       1;
    2             2         3       1;
    3             1         4       1;
    4             2         5       1;
    5             3         5       1;
    6             4         5       1];
```

**Figure 4-11:** Full Network Bus/Branch Model

### 4.4.3 Code Structure

In addition to implementing the three-phase state estimation application in Matlab, the first phase of the Dominion project includes developing a topology processor application in Matlab. As discussed in a previous chapter, the topology processor is responsible for telling the state estimator the topology information of the network in the form of a list of in-service branches each with the number of the busses that they are connected to. In a sense, it serves to determine the one-line diagram of the system and construct it in a tabular form. This will be referred to as the bus/branch model or bus/branch information. The topology processor takes in breaker statuses and other system information including line flows and determines which branches are in service. The topology processor is then also responsible for detecting inconsistencies in this data to prevent topology errors. This section discusses the structure of the code of the Matlab implementation of the topology processor application and how it relates to each of these ideas.

Since at the 500kV level there will not be topological changes very often it is not necessary to perform an analysis of the data to determine the status of each line for every iteration of the program. Therefore, upon receiving the latest set of data from the system, the program should first check for a change from the last set of data. There are two types of data that the topology processor will use. The first and most obvious is the circuit breaker statuses in the network. The topology processor will also use line flows to verify
the information provided by the breaker statuses in order to provide a more robust identification of out-of-service branches.

The first step will be to translate the line current flow measurements from analog readings into a logical ‘on-off’ type of information. The program should look at the current measurement and decide either ‘there is current flowing in this line’ or ‘there is no current flowing in this line’. The criteria for determining the state of each of the branches is discussed in another section of this document. This information is then assembled into a vector in a specific order and compared to the same vector from the previous iteration of the program. If the two vectors are subtracted and the vector is identically zero then there has been no change in the line current since the previous iteration. It may be safe to assume that this means there have been no topological changes since the last iteration, however, the breaker statuses (which exist originally as Boolean information) should be compared in the same fashion.

If no change is detected then the topology processor should output the bus/branch list from the previous iteration of the program. If a change is detected then the program should attempt to determine what exactly has changed in the network. Previously, a vector was computed that was the difference between the Boolean line flow information from the most recent iteration and the previous iteration. If there is a non-zero value in this vector, its position inside of the vector will indicate which branch is affected and the sign of the value (positive or negative) will indicate whether it was removed from service or put in service. The program will reference a lookup table that will indicate which circuit breakers are associated with the service status of each branch. Then, the program will check each breaker in this list. If the line was put back in service then the circuit breaker statuses should all read ‘closed’. And if the line was taken out of service then the circuit breaker statuses should all read ‘open’. If both of these pieces of information agree with each other, then the most recent bus/branch list will be updated by the program and output to the state estimator or any other applications that may need topology information of the system. If there are inconsistencies between these two pieces of information then the line in question will not be updated on the bus/branch list. If there is a scenario where two lines change during the same iteration and the change in one branch is verified while

67
the other is not, then the branch that is verified will be updated and the other branch will not.

4.5 Conclusion

This chapter presented the transition of the linear state estimation algorithm from the single phase to the three phase world. It discussed the development of two of the applications being developed as part of the Dominion/DOE research project and provided specifics concerning input, output, database files and code structure. All of the code written for this project has been included in the Appendices of this document. The next chapter will present the testing procedure and results from testing the state estimator application along with a demonstration of the functionality of the topology processor application. It also discusses the future work related to the Dominion/DOE research project.
Chapter 5
Testing, Results, & Future Work

As discussed previously, the purpose of implementing these applications at the beginning of the project is to demonstrate the proof-of-concept and identify potential issues associated with the actual implementation of the application code or system architecture. In order to achieve this, the developed code must be tested against some type of constraint in order to prove its successful functionality. This chapter discusses the testing of the Matlab implementation of the three-phase estimator application and the topology processor application.

5.1 Testing of the State Estimator Application

Since a three-phase linear state estimator has never been implemented before, this section includes both the development of the testing procedures as well as the results and analysis of the tests themselves. Many issues arose when attempting to test the code due to the unavailability of resources such as system information. Therefore, several subsections are included that discuss creating fictitious systems and measurements. Additionally, a discussion of the testing procedure and testing script is presented. The final subsection presents the results of the state estimator’s ability to filter out measurement errors compared to a known system state.

5.1.1 Creating Three-Phase Unbalanced Impedances

Early in the process, there was an absence of any real system data including three phase line impedances. The only information present was a positive sequence model of the 500kV network. It was necessary to have realistic three phase impedance structures for the testing of the three phase linear state estimation code, so an approach was taken to create approximated three phase impedance structures [18].
First, a standard three phase impedance structure was selected from the EPRI Transmission Line Reference Book [19]. The 500kV network was believed to be made up mostly of transmission lines with a horizontal configuration so a value of impedance that most closely represented this was used in this procedure. The positive sequence equivalent impedance of this three phase impedance structure was calculated. Given the following three phase impedance structure

\[
Z_{abc} = \begin{bmatrix}
Z_{aa} & Z_{ab} & Z_{ac} \\
Z_{ab} & Z_{bb} & Z_{bc} \\
Z_{ac} & Z_{bc} & Z_{cc}
\end{bmatrix}
\]  

(5.1)

The positive sequence impedance can be calculated as follows [9].

\[
Z_1 = \frac{1}{3}(Z_{aa} + Z_{bb} + Z_{cc} - Z_{ab} - Z_{ac} - Z_{bc})
\]  

(5.2)

The relative length of the transmission lines in the network could be determined by dividing each line’s positive sequence impedance from the system model by the positive sequence value obtained from the transmission line reference. This will not be an actual length because both values are complex in nature and therefore this scaling factor will also be complex. A scaling factor is then determined for each of the lines in the network. The last step to generate a three phase impedance structure for each of the transmission lines is to multiply the base value given from the book by each of the scaling factors determined previously. This creates a set of three phase impedances.

### 5.1.2 Generating a Fake Measurement Set

Because of the stage of development of the project, real system data could not be produced to test the three phase linear state estimation MATLAB code. In the absence of real data, a creative approach had to be taken to test the code. The following summarizes the procedure for creating ‘fake’ system measurements [20]. The goal was to generate a realistic set of measurements and a corresponding system state in three phases. Introducing imbalances into these pseudo measurements was also desired.

In order to accomplish this, several things were needed. First, two three phase admittances matrices must be constructed. One is the desired system model using unbalanced three phase impedances. The other is a balanced equivalent of the desired
system model. The three phase admittance matrix has the same structure on the whole as the single phase admittance matrix. However, instead of each non-zero entry being a single admittance, they are now a 3x3 square symmetrical admittance matrix of the following form.

\[
Y_{\text{unbalanced}} = \begin{bmatrix}
Y_{aa} & Y_{ab} & Y_{ac} \\
Y_{ab} & Y_{bb} & Y_{bc} \\
Y_{ac} & Y_{bc} & Y_{cc}
\end{bmatrix}
\] (5.3)

A zero entry in the single phase admittance matrix would become a 3x3 zero matrix. The structure of a balanced three phase admittance has the following form.

\[
Y_{\text{balanced}} = \begin{bmatrix}
Y_{D} & Y_{O} & Y_{O} \\
Y_{O} & Y_{D} & Y_{O} \\
Y_{O} & Y_{O} & Y_{D}
\end{bmatrix}
\] (5.4)

The diagonal elements of the balanced three phase impedance are each the average of the diagonal elements of the unbalanced three phase impedance [3].

\[
Y_{D} = \frac{Y_{aa} + Y_{bb} + Y_{cc}}{3}
\] (5.5)

The off-diagonal elements of the balanced three phase impedance are each the average of the off-diagonal elements of the unbalanced three phase impedance [3].

\[
Y_{O} = \frac{Y_{ab} + Y_{bc} + Y_{ac}}{3}
\] (5.6)

Therefore, each 3x3 element of the three phase unbalanced admittance matrix can be converted into a balanced three phase equivalent.

The three phase balanced and unbalanced impedance matrices must also be determined.

\[
[Z_{\text{unbalanced}}] = [Y_{\text{unbalanced}}]^{-1}
\] (5.7)

\[
[Z_{\text{balanced}}] = [Y_{\text{balanced}}]^{-1}
\] (5.8)

The last piece that must be determined before generating a set of unbalanced three phase measurements and a corresponding system state is performing a positive sequence load flow on the 500kV network. This task was performed using PSS/E. Different load models were not considered as only a single snapshot of the system is really necessary for
evaluating the performance of the three phase linear state estimator code. The solution to the load flow is the single phase equivalent of the system state. This must be expanded to three phases by adding and subtracting 120° from each state variable. This is the equivalent of a perfectly balanced three phase version of the same system. Unbalances can be introduced into the system state and current injections using the following procedure [20]. First, two sets of current injections are computed using the balanced three phase system state.

\[
[I_{\text{unbalanced}}] = [Y_{\text{unbalanced}}][V_{\text{balanced}}] \\
[I_{\text{balanced}}] = [Y_{\text{balanced}}][V_{\text{balanced}}]
\]

(5.9)  
(5.10)

Then, an unbalanced system state can be determined using the unbalanced three phase impedance matrix and the balanced set of current injections determined previously.

\[
[V_{\text{unbalanced}}] = [Z_{\text{unbalanced}}][I_{\text{balanced}}]
\]

(5.11)

The final system state and corresponding set of current injections that will be used will be a linear combination (an average) of the respective balanced and unbalanced sets.

\[
[V_{\text{final}}] = \frac{([V_{\text{balanced}}] + [V_{\text{unbalanced}}])}{2}
\]

(5.12)

\[
[I_{\text{final}}] = \frac{([I_{\text{balanced}}] + [I_{\text{unbalanced}}])}{2}
\]

(5.13)

Because these are a linear combination then the following is true.

\[
[I_{\text{final}}] = [Y_{\text{unbalanced}}][V_{\text{final}}]
\]

(5.14)

It can be seen that now there is a solved three phase unbalanced system and a corresponding system state.

The currents, however, are injections and not line flows and therefore cannot be substituted as measurements because the PMUs that will provide the actual measurements return line flows and the equations have been written for such measurements [20]. However, since the \(B\) system matrix used in the estimator algorithm uses the same impedance data, it can be used in conjunction with the three phase unbalanced system state to generate a set of line current measurements.
\[
[l_{\text{line flows}}] = [B_{\text{system}}][V_{\text{final}}] 
\] (5.15)

The full set of error free measurements will then be the vertical concatenation of the three phase unbalanced voltages and three phase unbalanced line flows.

\[
[z_{\text{true}}] = \begin{bmatrix}
[V_{\text{final}}] \\
[l_{\text{line flows}}]
\end{bmatrix} 
\] (5.16)

And the corresponding error free system state is simply the three phase unbalanced voltages.

\[
[x_{\text{true}}] = [V_{\text{final}}] 
\] (5.17)

A simple sanity check of the above would be to insert these values into the linear estimator equation.

\[
[z_{\text{true}}] = \begin{bmatrix}
I \\
yA + y_s
\end{bmatrix} [x_{\text{true}}] 
\] (5.18)

### 5.1.3 Testing the State Estimator

The three phase linear state estimator application was developed in Matlab as part of the initial phase of the Dominion/DOE research project. The Matlab code for this application can be found in Appendix A. Developing in Matlab first allows for quick development and testing without worry or concern of the platform that the final application will be run on. It also makes it easier to define specifications of application input and output and to foresee logistical issues that may arise in later phases of the project. However, as discussed in a few pervious subsections, in the absence of real system data, many other steps had to be taken to test the Matlab application. This section discusses the testing procedure and results for the three phase linear state estimator Matlab application. The first subsection describes the script written for testing the state estimator application and the second subsection discusses the results from each of these tests.
5.1.3.1 State Estimator Testing Procedure

The testing script begins by clearing any data in the Matlab Workspace so that it might not interfere with any of the calculations executed by the script and so that when reviewing the results, all of the data in the Matlab Workspace is from the script calculations only. Before the actual testing of the application begins, the test setup portion of the script is executed. The first step is to take the positive sequence load flow data in a text file and turn it into a three phase unbalanced system state to be used by the testing script.

% Takes solved load flow data and the system model to create a three-phase
% system state. This is assumed to be the true state of the system.
[actual_state,I] = getDVPdata();

Figure 5-1: Section of Matlab Testing Script

It also returns unbalanced three phase current injections from each bus. The inner workings of this function are discussed in a previous subsection. Next, the unbalanced three phase system state is used to generate a set of voltage measurements to serve as input to the state estimator. This step is required because each substation monitored with PMU data will have more than one voltage measurement at that location. Therefore, there is more than one voltage measurement per state variable.

% Since each measured substation actually has many busses inside of it and
% each voltage is measured multiple times, the calculated system state from
% the load flow data is extrapolated so that there are five voltage
% measurements at each substation.
[voltage_measurement_set] = extrapolateDVPvoltage_measurements(actual_state);
[number_of_voltage_measurements,x] = size(voltage_measurement_set);

Figure 5-2: Section of Matlab Testing Script

The next step in the test setup would be to generate a set of current measurements that correspond to the system state determined by the load flow and the three phase admittance matrix. However, these measurements are line flows and not injections. Therefore, the simplest way to accomplish this is multiplying the system state by the lower partition of the system matrix relating the system state to the current measurements. Therefore, before the current measurements can be calculated the system matrix should be populated.
This function returns the lower partition of the system matrix, \([k]\), and the entire system matrix, \([K]\). Then, the current measurements can be calculated from the lower partition.

The next part of the test setup does several things. First, it sets the number of iterations for the test and determines the number of state variables and current measurements. It also creates a space for all of the data created by the test to be stored in the Matlab Workspace. These include, for each iteration, a space for the output of the state estimator, a space for the voltage and current measurements, and a space to keep line current flows that are calculated using the output of the state estimator.
Now, everything is in place to begin testing of the state estimator. What follows will be executed multiple times based on the number of iterations set above. The first step in each iteration is to take the calculated current and voltage measurements from the test setup and induce some additive normally distributed errors to each of the measurements. Since the measurements are complex, an error is added to both the real and imaginary parts of the measurement. Matlab has a built-in function for generating random errors. A function has been written and can be found in Appendix A that adds these errors to the voltage and current measurements and populates a corresponding error covariance matrix for the measurement set. The measurement variance for the voltage measurements has been assumed to be 0.01 and the measurement variance for the current measurements has been assumed to be 0.03. This follows the idea that current measurements are less accurate that voltage measurements due to the nature of the metering device [16].

% Introduce normally distributed errors to the generated sets of voltage % and current measurements. Additionally, populate a corresponding % covariance matrix for these errors.
[W,voltage_measurements,current_measurements] = errornormal(...
    voltage_measurement_set,current_measurement_set);

Figure 5-6: Section of the Matlab Testing Script

The errors are added to the original system state and current flows at each iteration. This simulates a system in steady state with independent, random measurement errors during each iteration of the estimator. Next the error induced voltage and current measurements can be concatenated together to form the measurement vector.

% Assemble the full measurement set by concatenating the voltage % measurements and current measurements
measurements = [voltage_measurements;current_measurements];

Figure 5-7: Section of the Matlab Testing Script

At the first iteration of the program, the inverse of the error covariance matrix is taken and the pseudo-inverse of the system matrix is taken. These two matrices are used in the estimation of the system state.
The final step of each iteration of the testing procedure is the calculation of the system state using the pseudo-inverse of the system matrix including the error covariance matrix. Then the data created in each it is stored in its respective data structure.

```matlab
if i == 1
    % Take the inverse of the covariance matrix
    W_1 = W\eye(size(W));

    % Compute the pseudo-inverse of the system matrix
    [X,y] = size(K);
    M = (K.'*W_1*K)\(eye(y) * (K.'*W_1));
end
```

**Figure 5-8:** Section of the Matlab Testing Script

The remaining part of the testing script is simply for plotting the results of the test. There are several different kinds of plots that are used to show the effectiveness of the state estimator application. These are presented and discussed in the following section. The measurement vector used as input to the state estimator application should mirror as closely as possible the measurement vector that will actually be available when the system is fully implemented on the Dominion 500kV network. Shown in Figure 5-10 is Dominion’s 500kV network and the locations of all of the PMUs that will be installed at the substations. For the measurement vector, voltage measurements will only be available for those substations where PMUs are located. There will, however, be more than one voltage measurement at each of these substations. Current measurements will only exist for lines which are adjacent to substations which have PMUs. The current measurements will not be redundant in the same way that the voltage measurements are redundant. This information will be reflected in the current measurement lookup table and the voltage measurement lookup table and can be found in Appendix A.
Figure 5-10: Dominion 500kV Network
5.1.3.2 State Estimator Testing Results

The desired function of any state estimator is to filter out the measurement error using redundancy and knowledge about the measurement errors. The testing script used to test the Matlab application of the three phase linear state estimator using the data created by the test to demonstrate this in several ways.

1) The true value and the estimated value of a single state variable are plotted on the complex plane for each iteration of the testing procedure. The raw voltage measurements corresponding to that particular state variable are also included on the same plot. This type of plot shows the ability of the state estimator to filter out measurement error and hone in on the true value of the state variable.

2) A different plot which shows the estimators ability to filter out measurement error from the entire state vector. These plots only show results from a single iteration. Since the true value of the system state is known from the load flow, it is compared to the corresponding voltage measurements and estimated state vector and plotted to show the effects. This is done in four different plots: a plot showing the effects on the real part of the state variable, a plot showing the effects on the imaginary part of the state variable, a plot showing the effects on the magnitude of the state variable, and a plot showing the effects on the angle of the state variable.

*Figure 5-11* and *Figure 5-12* are two plots showing a single state variable over 100 iterations of the testing script. This particular state variable corresponds to a substation which is monitored by a PMU. *Figure 5-11* shows in green the true value of the state variable and in blue the estimated values of the state variable for 100 iterations. *Figure 5-12* shows in blue the estimated values of the state variable and in red the voltage measurements that correspond to this particular state variable. Both of these plots are from the same set of iterations so they can be visually compared to each other. These plots show very clearly the ability of the estimator to filter out measurement errors and determine the best estimate of the system state. It can be seen from *Figure 5-11* that the true value of the state variable differs from the estimated value by some value on the order of $10^{-3}$. *Figure 5-13* and *Figure 5-14* are the same type of plot but show a state variable which corresponds to a substation which is not monitored by a PMU.
Figure 5-11: State Estimator Output for 100 Iterations

Figure 5-12: State Estimator Output & Voltage Measurements for 100 Iterations
Figure 5-13: State Estimator Output for 100 Iterations (w/o PMU)

Figure 5-14: State Estimator Output for 100 Iterations (w/o) PMU
The next set of plots show the effects of the estimation process on the entire state vector for a single iteration of the testing script. The absolute value of the difference between the voltage measurements and the actual state vector are shown in red. The absolute value of the difference between the estimated state vector and the actual state vector are shown in blue.

Figure 5-15: Comparison of Real Part of Output & Measurements
**Figure 5-16:** Comparison of Imaginary Part of Output & Measurements

**Figure 5-17:** Comparison of Magnitude of Output & Measurements
It can be seen from these plots that the state estimator can effectively filter out measurement normally distributed random additive errors and estimate with a good degree of precision the system state. In the case of all four comparisons, the estimator has the ability to come within approximately 0.002 of the true value of each of the parameters.

### 5.2 Functionality of the Topology Processor

The testing of the topology processor Matlab application is a much simpler task than the state estimator application. This is true for several reasons. First, since the successful functionality of the application is based more on a correctly formatted input file versus properly placed measurements and measurement redundancy, the application did not have to be tested on a model of the Dominion 500kV network. It was simply tested on the small 5-bus system that has been repetitively used throughout this document for examples. Secondly, testing of this application did not depend on having raw current measurements. The current measurements were modeled as logic 1 or 0 representing...
either ‘flow’ or ‘no flow’, respectively. The pre-processing of the raw current measurements that would result in a set of data such as this was not discussed in this thesis and is a topic of future work for the project.

As discussed in a previous chapter, the topology processor is responsible for taking current flow data and breaker statuses and determining the network topology of the system. It first does so by checking if there is current flowing on each of the branches in the network. If the application determines that there is no current flowing on a particular branch it references a lookup table to see which breakers must be open for that particular line to be out of service. If all of the corresponding breakers are out of service then the topology processor will update the bus/branch model to reflect this change. Otherwise, it should output some error message indicating to the operator that the breaker information and the current information do not correspond to each other. In this case, the application will not update the bus/branch model of the network. Examples of this are shown in this section.

Recall the simple 5 bus system shown in Figure 5-19. This system does differ from the previously presented 5 bus system. There is a current measurement at each end of each line.

![Figure 5-19: Example 5-Bus System with Current Measurements](image-url)
Consider if the system was in steady state and all of the lines were in service. Imagine now that a fault occurred on the line connecting Substation 1 to Substation 4 and relays take this line out of service. Current measurements 5 and 6 should read zero and breakers 2, 3, 11, and 12 should read open. To test the topology processors response to this scenario, the input files representing the breaker statuses and current measurements must be edited before the simulation is run. With current measurements and breaker statuses reflecting the scenario described above serving as measurement inputs to the topology processor, the application outputs the following to the Matlab Command Window.

\[
\text{OUTPUT} =
\begin{array}{cccc}
1 & 1 & 2 & 1 \\
2 & 2 & 3 & 1 \\
3 & 1 & 4 & 0 \\
4 & 2 & 5 & 1 \\
5 & 3 & 5 & 1 \\
6 & 4 & 5 & 1 \\
\end{array}
\]

**Figure 5-20:** Topology Processor Application Output (Successful)

This is the bus/branch model of the network with a column indicating the service status of the each of the lines. The first column is the line number. The second and third columns are the ‘From Bus’ and ‘To Bus’, respectively. The fourth column is the column which shows the status of the line. It can be seen that line 3 is shown to be out of service.

Consider the same scenario only now, one of the breakers still reads closed, maybe simply due to latency of the breaker status signal inside the substation. The topology processor would see that all of the required breakers did not open and will alert the user. It will also output the most recent bus/branch model. This bus branch model will not reflect the potential out of service line. The application will output the following to the Matlab Command Window.
Figure 5.21: Topology Processor Application Output (Unsuccessful)

This is just one example of the functionality of the topology processor application developed in Matlab. All of the topology processor application code can be found in Appendix B.

5.3 Future Work

As discussed previously, the material in this document represents the body of work completed as part of the first of three phases of the Dominion/DOE research project. The aim of the project is to develop a three-phase linear tracking state estimator and several other applications that will function alongside of it and implement these applications on Dominion’s 500kV network. This is contingent on the installation of a large set of PMUs in Dominion’s 500kV substations over the next three years as part of a larger system architecture that will eventually serve to bring phasor measurements to the applications being developed.

While these applications, including those presented in this document as well as others, have been originally developed in Matlab they must be translated to another language so that can function on the platform that will run inside of the final system architecture. Many of the specifics of the architecture and overall functionality of the system is beyond the scope of this thesis. However, the platform that this software will run on is important. The project team has selected an open source platform for the applications to run on called openPDC. More information regarding openPDC can be found here [21].
There is a great deal of effort that must be taken to move these applications from Matlab over to openPDC and subsequently test their functionality. The Matlab code must be translated into C# as part of the second phase of the Dominion/DOE research project. This language is specified because of the way that openPDC allows for user developed applications. It must also be formatted in such a way that openPDC knows when and what pieces of the applications to execute. Additionally, at that point in the project there will not be sufficient numbers of installed PMUs to test the applications on the openPDC platform and a way to test the applications. Finding a way to test the applications in openPDC with a large enough pool of measurements may prove to be quite difficult. Other smaller concerns for the second phase of the project include evaluation of computation time for each of the applications. Since the original specifications of the project are time sensitive, each of the applications must run below certain time thresholds. Additionally, a technique to process the raw current measurements into logic 1 or 0 to represent ‘flow’ or ‘no flow’ must be developed for the topology processor application. The third and final phase of the project is reserved for the final testing and implementation when the end-to-end system has been installed and all remaining details have been specified.

5.4 Conclusion

This thesis began with an introduction to the Dominion/DOE research project and a history of traditional state estimation and phasor measurement units. It followed with a presentation of traditional state estimation techniques and provided a numerical example. It continued on to discuss several methods for including phasor measurements in traditional state estimation techniques and then developing a formulation of a completely linear state equation which uses PMU data exclusively. Examples were presented for matrix formulation and the mathematical solution to two formulations of the linear estimation algorithm and a transition into three phase linear state estimation.

The fourth chapter began presenting the body of work completed as part of the initial phase of the Dominion/DOE research project. This included the development of the three-phase linear state estimation application and the topology processor application. The thesis then concluded with the testing procedures and results of testing the state
estimator application and a discussion of the functionality of the topology processor application. The state estimator application successfully demonstrated its ability to filter out normally distributed measurement errors and the topology processor application demonstrated its ability to determine a line outage and provide error messages in the case of inconsistent measurements. Finally, a brief discussion of the future direction of the Dominion/DOE research project has presented as it relates to the future work of the development of these two applications.
References


Appendix A

Three Phase Linear State Estimator Application

% Three Phase Linear Tracking State Estimator Testing Script
clear
%-----------------------------------------------------------------------
%----------------------------------- Begin Test Setup ---------------------
%-----------------------------------------------------------------------

% Takes solved load flow data and the system model to create a three-phase
% system state. This is assumed to be the true state of the system.
[actual_state,I] = getDVPdata();

% Since each measured substation actually has many busses inside of it and
% each voltage is measured multiple times, the calculated system state from
% the load flow data is extrapolated so that there are five voltage
% measurements at each substation.
[voltage_measurement_set] = extrapolateDVPvoltage_measurements(actual_state);
[number_of_voltage_measurements,x] = size(voltage_measurement_set);

% Assembles the system matrix. This matrix is used to calculate a true set
% of line current flows to serve as the current measurement set. This will
% also be inverted in the state equation.
[k,K] = getDVPsystem_matrix(DVP_series,DVP_susceptance,
    DVP_current_measurement_lookup_table,
    DVP_voltage_measurement_lookup_table);

% Create a set of current measurements by multiplying the system matrix by
% the actual state of the system.
[current_measurement_set] = k*actual_state;

%-----------------------------------------------------------------------
%----------------------------------- End Test Setup ---------------------
%-----------------------------------------------------------------------

% Set the number of iterations for the test
number_of_iterations = 100;
% Determine the number of state variables
[number_of_state_variables,x] = size(actual_state);
% Determine the number of current measurements
[number_of_current_measurements,x] = size(current_measurement_set);
% Create a historian to store the estimator output at each iteration
output_historian = zeros(number_of_state_variables,number_of_iterations);
% Create a historian to store the current flows at each iteration
current_flow_historian = zeros(number_of_current_measurements,number_of_iterations);
% Create a historian to store the current measurements at each iteration
current_measurement_historian = zeros(number_of_current_measurements,
number_of_iterations);
% Create a historian to store the voltage measurements at each iteration
voltage_measurement_historian = zeros(number_of_voltage_measurements,...
    number_of_iterations);

for i = 1:1:number_of_iterations
% Introduce normally distributed errors to the generated sets of voltage
% and current measurements. Additionally, populate a corresponding
% covariance matrix for these errors.
[W,voltage_measurements,current_measurements] = errornormal(...
    voltage_measurement_set,current_measurement_set);

% Assemble the full measurement set by concatenating the voltage
% measurements and current measurements
measurements = [voltage_measurements;current_measurements];

if i == 1
    % Take the inverse of the covariance matrix
    W_1 = W\eye(size(W));
    % Compute the pseudo-inverse of the system matrix
    [x,y] = size(K);
    M = (K.'*W_1*K)/(eye(y)*(K.'*W_1));
end

% Compute the state estimate
OUTPUT = M*measurements;

% Store the most recent output for plotting
output_historian(:,i) = OUTPUT;
current_flow_historian(:,i) = k*OUTPUT;
current_measurement_historian(:,i) = current_measurements;
voltage_measurement_historian(:,i) = voltage_measurements;
end

%-----------------------------------------------------------------------------%
%-----------------------------------------------------------------------------%

for k = 1:1:2
    % When k = 1, plot the state variable and the estimator output. when
    % k = 2, plot the estimator output and the voltage measurements.
    % Create new figure
    figure
    hold on

for j = 1:1:number_of_iterations
    if j == 1 && k == 1
% Plot the true state variable
a = real(actual_state(j));
b = imag(actual_state(j));
% Plot a green circle
plot(a,b,'go','MarkerSize',10)
end
%
\begin{verbatim}
end
% Plot the calculated state variable for 100 iterations
w = real(output_historian(1,j));
x = imag(output_historian(1,j));
% Plot blue stars
plot(w,x,'b*')
if k == 1
    title('Estimator Output','FontSize',14,...
         'FontAngle','ITALIC',...
         'FontName','Times New Roman',...
         'FontWeight','BOLD')
xlabel('Real (p.u.)','FontSize',12,...
       'FontAngle','ITALIC',...
       'FontName','Times New Roman',...
       'FontWeight','BOLD')
ylabel('Imaginary (p.u.)','FontSize',12,...
       'FontAngle','ITALIC',...
       'FontName','Times New Roman',...
       'FontWeight','BOLD')
end
if k == 2
    title('Measurements & Estimator Output','FontSize',14,...
          'FontAngle','ITALIC',...
          'FontName','Times New Roman',...
          'FontWeight','BOLD')
xlabel('Real (p.u.)','FontSize',12,...
       'FontAngle','ITALIC',...
       'FontName','Times New Roman',...
       'FontWeight','BOLD')
ylabel('Imaginary (p.u.)','FontSize',12,...
       'FontAngle','ITALIC',...
       'FontName','Times New Roman',...
       'FontWeight','BOLD')
end
\end{verbatim}

95
% Add the legends to the plots
if k == 1
    legend('State Variable','Estimator Output','Location','Best')
elseif k == 2
    legend('Estimator Output','Voltage Measurements','Location','Best')
end
hold off
end

%-------------------------------------------------------------------------
%-------------------------------------------------------------------------
for k = 1:1:2
    % When k = 1, plot the state variable and the estimator output. when
    % k = 2, plot the estimator output and the voltage measurements.

    % Create new figure
    figure
    hold on
for j = 1:1:number_of_iterations

    if j == 15
        % Plot the true state variable
        a = real(current_measurement_set(j));
        b = imag(current_measurement_set(j));
        % Plot a green circle
        plot(a,b,'go','MarkerSize',10)
    end

end

% Plot the calculated state variable for 100 iterations
w = real(current_flow_historian(15,j));
x = imag(current_flow_historian(15,j));
% Plot blue stars
plot(w,x,'b*')

if k == 1
    title('Calculated Current Flow','FontSize',14,...
        'FontAngle','ITALIC',...
        'FontName','Times New Roman',...
        'FontWeight','BOLD')
xlabel('Real (p.u.)','FontSize',12,...
        'FontAngle','ITALIC',...
        'FontName','Times New Roman',...
        'FontWeight','BOLD')
ylabel('Imaginary (p.u.)','FontSize',12,...
        'FontAngle','ITALIC',...
        'FontName','Times New Roman',...
        'FontWeight','BOLD')
if k == 2
  \% Plot the voltage measurement for 100 iterations
  y = real(current_measurement_historian(15,j));
  z = imag(current_measurement_historian(15,j));
  \% Plot red stars
  plot(y,z,'r*')
  title('Measurements & Calculated Current Flow',...
  'FontSize',14,...
  'FontAngle','ITALIC',...
  'FontName','Times New Roman',...
  'FontWeight','BOLD')
  xlabel('Real (p.u.)',...
  'FontSize',12,...
  'FontAngle','ITALIC',...
  'FontName','Times New Roman',...
  'FontWeight','BOLD')
  ylabel('Imaginary (p.u.)',...
  'FontSize',12,...
  'FontAngle','ITALIC',...
  'FontName','Times New Roman',...
  'FontWeight','BOLD')
end
end
\%
% Create an independent variable for all graphs
x = 1:1:number_of_state_variables;
\%
% Choose an iteration to plot
iteration = 27;
\%
% Create spaces for data from test to sit
measurement = zeros(number_of_state_variables,1);
output = zeros(number_of_state_variables,1);
for i = 1:1:(number_of_state_variables/3)
  \% Only select one Voltage measurement per state variable
  measurement(3*i-2) = voltage_measurement_historian(15*i-14,iteration);
  measurement(3*i-1) = voltage_measurement_historian(15*i-13,iteration);
  measurement(3*i) = voltage_measurement_historian(15*i-12,iteration);
  output(3*i-2) = output_historian(3*i-2,iteration);
output(3*i-1) = output_historian(3*i-1,iteration);
output(3*i)   = output_historian(3*i,iteration);
end

% Calculate the real and imaginary parts of the state variables and the
% magnitude and angles of the state variables. You cannot plot complex
% values in this way.
real_actual_state = real(actual_state);
imag_actual_state = imag(actual_state);
mag_actual_state = abs(actual_state);
ang_actual_state = angle(actual_state);

% Do the same for the estimator output vector
real_output_state = real(output);
imag_output_state = imag(output);
mag_output_state = abs(output);
ang_output_state = angle(output);

% Do the same for the voltage measurements
real_measurement = real(measurement);
imag_measurement = imag(measurement);
mag_measurement = abs(measurement);
ang_measurement = angle(measurement);

% Prepare to plot the error in the real part of the phasor by comparing the
% estimator output and the voltage measurement to the actual state vector.
real_metric1 = abs(real_actual_state - real_output_state);
real_metric2 = abs(real_actual_state - real_measurement);
% Open a new figure
figure
hold on
% Plot the comparison of the estimator output to the actual state with
% blue circles
plot(x,real_metric1,'bo','MarkerSize',5)
% Plot the comparison of the voltage measurement to the actual state with
% red circles
plot(x,real_metric2,'ro','MarkerSize',5)
title('Error of Real Part of Phasor',...
'FontSize',14,...
'FontAngle','ITALIC',...
'FontName','Times New Roman',...
'FontWeight','BOLD')
xlabel('Real (p.u.)',...
'FontSize',12,...
'FontAngle','ITALIC',...
'FontName','Times New Roman',...
'FontWeight','BOLD')
ylabel('Real Error (p.u.)',...
'FontSize',12,...
'FontAngle','ITALIC',...
'FontName','Times New Roman',...
'FontWeight','BOLD')
legend('Estimator Output','Voltage Measurements','Location','Best')

% Prepare to plot the error in the imaginary part of the phasor by
% comparing the estimator output and the voltage measurement to the
% actual state vector.
imag_metric1 = abs(imag_actual_state - imag_output_state);
imag_metric2 = abs(imag_actual_state - imag_measurement);
% Open a new figure
figure
hold on
% Plot the comparison of the estimator output to the actual state with % blue circles
plot(x,imag_metric1,'bo','MarkerSize',5)
% Plot the comparison of the voltage measurement to the actual state with % red circles
plot(x,imag_metric2,'ro','MarkerSize',5)
title('Error of Imaginary Part of Phasor',...
    'FontSize',14,...
    'FontAngle','ITALIC',...
    'FontName','Times New Roman',...
    'FontWeight','BOLD')
xlabel('State Variable',...
    'FontSize',12,...
    'FontAngle','ITALIC',...
    'FontName','Times New Roman',...
    'FontWeight','BOLD')
ylabel('Imaginary Error(p.u.)',...
    'FontSize',12,...
    'FontAngle','ITALIC',...
    'FontName','Times New Roman',...
    'FontWeight','BOLD')
legend('Estimator Output','Voltage Measurements','Location','Best')

% Prepare to plot the error in the magnitude of the phasor by % comparing the estimator output and the voltage measurement to the % actual state vector.
mag_metric1 = abs(mag_actual_state - mag_output_state);
mag_metric2 = abs(mag_actual_state - mag_measurement);
% Open a new figure
figure
hold on
% Plot the comparison of the estimator output to the actual state with % blue circles
plot(x,mag_metric1,'bo','MarkerSize',5)
% Plot the comparison of the voltage measurement to the actual state with % red circles
plot(x,mag_metric2,'ro','MarkerSize',5)
title('Error of Phasor Magnitude',...
    'FontSize',14,...
    'FontAngle','ITALIC',...
    'FontName','Times New Roman',...
    'FontWeight','BOLD')
xlabel('State Variable',...
    'FontSize',12,...
    'FontAngle','ITALIC',...
    'FontName','Times New Roman',...
    'FontWeight','BOLD')
ylabel('Magnitude (p.u.)',...
    'FontSize',12,...
    'FontAngle','ITALIC',...
    'FontName','Times New Roman',...
% Prepare to plot the error in the angle of the phasor by comparing the estimator output and the voltage measurement to the actual state vector.
ang_metric1 = abs(ang_actual_state - ang_output_state);
ang_metric2 = abs(ang_actual_state - ang_measurement);

% Open a new figure
figure
hold on

% Plot the comparison of the estimator output to the actual state with blue circles
plot(x,ang_metric1,'bo','MarkerSize',5)

% Plot the comparison of the voltage measurement to the actual state with red circles
plot(x,ang_metric2,'ro','MarkerSize',5)

title('Error of Phasor Angle',... 
'FontSize',14,... 
'FontAngle','ITALIC',... 
'FontName','Times New Roman',... 
'FontWeight','BOLD')

xlabel('State Variable',... 
'FontSize',12,... 
'FontAngle','ITALIC',... 
'FontName','Times New Roman',... 
'FontWeight','BOLD')

ylabel('Degrees',... 
'FontSize',12,... 
'FontAngle','ITALIC',... 
'FontName','Times New Roman',... 
'FontWeight','BOLD')

legend('Estimator Output','Voltage Measurements','Location','Best')
function [V_Set, I_Set] = getDVPdata()

% Get a balanced set of three phase voltages
V = createVOLTAGE('DVPvoltages.txt');

% Get a balanced three phase admittance and impedance matrix
[YbusBalanced] = getDVPybus_balanced();

% Get an unbalanced three phase admittance and impedance matrix
[YbusUnBalanced, ZbusUnBalanced] = getDVPybus();

% Transpose V so that it is a column matrix
V = transpose(V);

% For balanced current injections use a balanced admittance matrix and a
% balanced set of voltages
IBalanced = YbusBalanced*V;

% For unbalanced current injections use an unbalanced admittance matrix and
% a balanced set of voltages
IUnBalanced = YbusUnBalanced*V;

% V is balanced
VBalanced = V;

% For unbalanced voltages use an unbalanced impedance matrix and balanced
% current
VUnBalanced = ZbusUnBalanced*IBalanced;

% Create Data Set by averaging the balanced and unbalanced sets
V_Set = (VBalanced+VUnBalanced)/2;
I_Set = (IBalanced+IUnBalanced)/2;
end

AUTHOR: Kevin D. Jones
Virginia Tech
LAST MODIFIED: 04/24/10
function [V] = createVOLTAGE(data)

% Open the file containing the necessary data
file = fopen(data);

% Extract the number of busses from the data file
b = textscan(file,' %u',1);

 Where the first number in the file (here it's 13) represents
 the total number of busses in the network. The first column
 of numbers represents the bus number of the respective bus.
 The second column of numbers represents the bus voltage
 magnitudes in per unit. The third column of numbers
 represents the bus voltage angle in degrees

 This function creates balanced three phase voltages

 % ARGUMENTS: data - title of the input file
 e.g. createVOLTAGE('PosSeqVoltages.txt')

 % OUTPUTS: V - set of balanced three phase voltages in a single column
 matrix in the same order as the single phase value.

 % AUTHOR: Kevin D. Jones
 Virginia Tech

 % LAST MODIFIED: 02/09/10

 function [V] = createVOLTAGE(data)
% Turn the cell into an integer
busses = b{1};

% For each bus
for k = 1:1:busses

    % Extract the bus
    bus(k) = textscan(file,' %f',1);
    % Extract the voltage magnitude
    v(k) = textscan(file,' %f',1);
    % Extract the voltage degree
    d(k) = textscan(file,' %f',1);
    % Loop

end

% Convert all of these cells into matrices so that they can be more easily
% manipulated
Bus = cell2mat(bus);
Vmag = cell2mat(v);
Vang = cell2mat(d);
Vang = Vang*pi/180;

% Calculate complex voltages (a + jb) from the magnitude and angle of the
% bus voltages
for i = 1:1:busses
    V(3*i-2)= complex(Vmag(i)*cos(Vang(i)),Vmag(i)*sin(Vang(i)));
    V(3*i-1)= complex(Vmag(i)*cos(Vang(i)-2*pi/3),Vmag(i)*sin(Vang(i)-
        2*pi/3));
    V(3*i)= complex(Vmag(i)*cos(Vang(i)+2*pi/3),Vmag(i)*sin(Vang(i)+2*pi/3));
end
end
% FUNCTION: getDVPybus()
% DESCRIPTION: Populates the admittance matrix of the DVP 500kV network.
% Generates arrays with all of the admittance and
% susceptance information to be used by other functions.
% Calculates the impedance matrix of the DVP 500kV network.
% ARGUMENTS: None
% OUTPUT: series_admittance_array - A 3*Bx3 array where B is the number of branches in the network. It is a list of the 3x3 admittance matrices for each branch.
% shunt_array - A 3*Bx3 array where B is the number of branches in the network. It is a list of the 3x3 shunt susceptance matrices for each branch.
% ybus - The admittance matrix of the network
% zbus - The impedance matrix of the network
% printableYBUS - A matrix in the form of the admittance matrix where the real and imaginary parts have been separated so that it can easily be moved to an Excel spreadsheet
% AUTHOR: Kevin D. Jones
% Virginia Tech
% LAST MODIFIED: 04/24/10

function [ybus, zbus] = getDVPybus()

% Clear the Command Window
% Rclc

% Input the system data increments
[number_of_lines] = size(connectivity);
[number_of_busses] = max(connectivity);
[from_to] = connectivity();
[impedance] = series();
[shunt_susceptance] = susceptance();

% Separate
from = from_to(:,1);
to = from_to(:,2);
resistance = impedance(:,1);
reactance = impedance(:,2);

% Create an array to put 3x3 impedance matrices for each line in
% It will be 3 x 3*number_of_lines
series_array = zeros(3*number_of_lines(1,1),3);

% Now populate the array with the impedance values
for i = 0:1:number_of_lines-1
    index = i*6 + 1;

% The rest of the function code would go here.
j = i + 1;
series_array(3*j-2,1) = complex(resistance(index), reactance(index));
series_array(3*j-2,2) =
complex(resistance(index+1), reactance(index+1));
series_array(3*j-2,3) =
complex(resistance(index+3), reactance(index+3));
series_array(3*j-1,1) =
complex(resistance(index+1), reactance(index+1));
series_array(3*j-1,2) =
complex(resistance(index+2), reactance(index+2));
series_array(3*j-1,3) =
complex(resistance(index+4), reactance(index+4));
series_array( 3*j ,1) =
complex(resistance(index+3), reactance(index+3));
series_array( 3*j ,2) =
complex(resistance(index+4), reactance(index+4));
series_array( 3*j ,3) =
complex(resistance(index+5), reactance(index+5));
end

% Create an array to put 3x3 impedance matrices for each line in
% It will be 3 x 3*number_of_lines
shunt_array = zeros(3*number_of_lines(1,1),3);

% Now populate the array with the impedance values
for i = 0:1:number_of_lines-1
  index = i*6 + 1;
  j = i + 1;
  shunt_array(3*j-2,1) = complex(0,shunt_susceptance(index));
  shunt_array(3*j-2,2) = complex(0,shunt_susceptance(index+1));
  shunt_array(3*j-2,3) = complex(0,shunt_susceptance(index+3));
  shunt_array(3*j-1,1) = complex(0,shunt_susceptance(index+1));
  shunt_array(3*j-1,2) = complex(0,shunt_susceptance(index+2));
  shunt_array(3*j-1,3) = complex(0,shunt_susceptance(index+4));
  shunt_array( 3*j ,1) = complex(0,shunt_susceptance(index+3));
  shunt_array( 3*j ,2) = complex(0,shunt_susceptance(index+4));
  shunt_array( 3*j ,3) = complex(0,shunt_susceptance(index+5));
end

% Calculate Admittances from Impedances
series_admittance_array = zeros(3*number_of_lines(1,1),3);

for i = 1:1:number_of_lines
  series_admittance_array(3*i-2:3*i,1:3) = inv(series_array(3*i-2:3*i,1:3));
end

%Create a space for the admittance matrix
ybus = zeros(3*number_of_busses(1,1));

% Populate Off-Diagonal Elements

% Each off diagonal entry is the negative of the admittance connecting the
% two associated busses. If there is no connection then the entry is zero
for i = 1:1:number_of_lines
    ybus(3*from(i)-2:3*from(i),3*to(i)-2:3*to(i)) = -
    series_admittance_array(3*i-2:3*i,1:3);
    ybus(3*to(i)-2:3*to(i),3*from(i)-2:3*from(i)) = -
    series_admittance_array(3*i-2:3*i,1:3);
end

% Populate Diagonal Elements
nextDiagonalEntry = zeros(3);

% The diagonal entry is the sum of all of the admittances connected to the
% associated bus.
for i = 1:1:number_of_busses
    for j = 1:1:number_of_lines
        if ((from(j) == i) || (to(j) == i))
            % Keep a running total
            nextDiagonalEntry = nextDiagonalEntry +
            series_admittance_array(3*j-2:3*j,1:3);
        end
    end
    % Put the total in the correct location in the Ybus Matrix
    ybus(3*i-2:3*i,3*i-2:3*i)= nextDiagonalEntry;
end

% Add the Shunt Susceptance to the Diagonals
for i = 1:1:number_of_lines
    % The shunt susceptance is only added to the diagonal entries in the matrix
    % and only half of the total shunt susceptance is added to the diagonal
    % entry associated with each bus. The total shunt susceptance is considered
    % because both the list of busses where the branch originates and
    % terminates are used
    a = from(i);
    b = to(i);
    % Bus where the branch originates
    ybus(3*a-2:3*a,3*a-2:3*a)=ybus(3*a-2:3*a,3*a-2:3*a)+shunt_array(3*i-2:3*i,1:3);
    % Bus where the branch terminates
    ybus(3*b-2:3*b,3*b-2:3*b)= ybus(3*b-2:3*b,3*b-2:3*b)+shunt_array(3*i-2:3*i,1:3);
end

% Compute the inverse of Ybus but do not use the MATLAB function
% inv(matrix) because it is numerically less robust than using the
% backslash (\) for matrix division when dealing with large matrices
zbus = inv(ybus);

% This should yeild Identity
I = ybus*zbus;
% Now to condition the numbers so that they can easily be looked at
% The result of the above multiplication should yeild the identity matrix
% but due to limited numerical precision the values may not be identically
% 1 or 0 in the matrix. This for-loop cycles through each element in the
% matrix and determines whether it is intened to be a 1 or a 0 by checking
% to see if the real parts are greater than 0.9999 to be a 1 or less than
% 0.0001 to be a zero. This makes the sanity check of calculating the
% identity
% matrix easier to see.

% Cycle through the
for j = 1:1:3*number_of_busses
    for k = 1:1:3*number_of_busses
        if real(I((j),(k))) > 0.9999
            I((j),(k)) = 1;
        elseif real(I((j),(k))) < 0.0001
            I((j),(k)) = 0;
        end
    end
end

% This separates the real and imaginary parts of the Ybus so that it can be
% put into an Excel spreadsheet to be easier to read and distribute.
realYBUS = real(ybus);
imagYBUS = imag(ybus);
printableYBUS = zeros(3*number_of_busses(1,1),6*number_of_busses(1,1));
for i = 1:1:3*number_of_busses
    printableYBUS(:,2*i-1) = realYBUS(:,i);
    printableYBUS(:,2*i) = imagYBUS(:,i);
end

end
function [ybus, zbus] = getDVPybus_balanced()

% Clear the Command Window
clc

% Input the system data increments
[number_of_lines] = size(connectivity);
[number_of_busses] = max(connectivity);
[from_to] = connectivity();
[impedance] = series();
[shunt_susceptance] = susceptance();

% Separate
from = from_to(:,1);
to = from_to(:,2);
resistance = impedance(:,1);
reactance = impedance(:,2);

% Create an array to put 3x3 impedance matrices for each line in % It will be 3 x 3*number_of_lines
series_array = zeros(3*number_of_lines(1,1),3);

% Now populate the array with the impedance values
for i = 0:1:number_of_lines-1
index = i*6 + 1;
j = i + 1;
%
% Take the average of diagonal elements
diagonal = (complex(resistance(index), reactance(index))+...
    complex(resistance(index+2), reactance(index+2))+...
    complex(resistance(index+5), reactance(index+5)))/3;
%
% Take the average of off-diagonal elements
off_diagonal = (complex(resistance(index+1), reactance(index+1))+...
    complex(resistance(index+3), reactance(index+3))+...
    complex(resistance(index+4), reactance(index+4)))/3;
%
% Make the diagonal elements the same
% Make the off-diagonal elements the same
series_array(3*j-2,1) = diagonal;
series_array(3*j-2,2) = off_diagonal;
series_array(3*j-2,3) = off_diagonal;
series_array(3*j-1,1) = off_diagonal;
series_array(3*j-1,2) = diagonal;
series_array(3*j-1,3) = off_diagonal;
series_array(3*j ,1) = off_diagonal;
series_array(3*j ,2) = off_diagonal;
series_array(3*j ,3) = diagonal;
end
%
% Create an array to put 3x3 impedance matrices for each line in
% It will be 3 x 3*number_of_lines
shunt_array = zeros(3*number_of_lines(1,1),3);
%
% Now populate the array with the impedance values
for i = 0:1:number_of_lines-1
    index = i*6 + 1;
j = i + 1;
%
% Take the average of diagonal elements
    diagonal = (complex(0,shunt_susceptance(index))+...
        complex(0,shunt_susceptance(index+2))+...
        complex(0,shunt_susceptance(index+5)))/3;
%
% Take the average of off-diagonal elements
    off_diagonal = (complex(0,shunt_susceptance(index+1))+...
        complex(0,shunt_susceptance(index+3))+...
        complex(0,shunt_susceptance(index+4)))/3;
%
% Make the diagonal elements the same
% Make the off-diagonal elements the same
    shunt_array(3*j-2,1) = diagonal;
    shunt_array(3*j-2,2) = off_diagonal;
    shunt_array(3*j-2,3) = off_diagonal;
    shunt_array(3*j-1,1) = off_diagonal;
    shunt_array(3*j-1,2) = diagonal;
    shunt_array(3*j-1,3) = off_diagonal;
    shunt_array(3*j ,1) = off_diagonal;
    shunt_array(3*j ,2) = off_diagonal;
    shunt_array(3*j ,3) = diagonal;
end
%
% Calculate Admittances from Impedances
series_admittance_array = zeros(3*number_of_lines(1,1),3);
%
for i = 1:1:number_of_lines
series_admittance_array(3*i-2:3*i,1:3) = inv(series_array(3*i-2:3*i,1:3));
end

%Create a space for the admittance matrix
ybus = zeros(3*number_of_busses(1,1));

%Populate Off-Diagonal Elements

% Each off diagonal entry is the negative of the admittance connecting the
% two associated busses. If there is no connection then the entry is zero
for i = 1:1:number_of_lines
    ybus(3*from(i)-2:3*from(i),3*to(i)-2:3*to(i)) = -
        series_admittance_array(3*i-2:3*i,1:3);
    ybus(3*to(i)-2:3*to(i),3*from(i)-2:3*from(i)) = -
        series_admittance_array(3*i-2:3*i,1:3);
end

%Populate Diagonal Elements
nextDiagonalEntry = zeros(3);

% The diagonal entry is the sum of all of the admittances connected to the
% associated bus.
for i = 1:1:number_of_lines
    for j = 1:1:number_of_lines
        % If the branch is connected to this particular bus then add it to
        % this particular diagonal entry in the matrix
        if ((from(j) == i) || (to(j) == i))
            % Keep a running total
            nextDiagonalEntry = nextDiagonalEntry +
                series_admittance_array(3*j-2:3*j,1:3);
        end
    end
    % Put the total in the correct location in the Ybus Matrix
    ybus(3*i-2:3*i,3*i-2:3*i)= nextDiagonalEntry;
    % Clear the running total so that the diagonals don't sum together
    nextDiagonalEntry = 0;
end

%Add the Shunt Susceptance to the Diagonals
for i = 1:1:number_of_lines
    % The shunt susceptance is only added to the diagonal entries in the matrix
    % and only half of the total shunt susceptance is added to the diagonal
    % entry associated with each bus. The total shunt susceptance is considered
    % because both the list of busses where the branch originates and
    % terminates are used
    a = from(i);
    b = to(i);
    % Bus where the branch originates
    ybus(3*a-2:3*a,3*a-2:3*a)=ybus(3*a-2:3*a,3*a-2:3*a)+shunt_array(3*i-2:3*i,1:3)/2;
    % Bus where the branch terminates
    ybus(3*b-2:3*b,3*b-2:3*b)= ybus(3*b-2:3*b,3*b-2:3*b)+shunt_array(3*i-2:3*i,1:3)/2;
% Compute the inverse of Ybus but do not use the MATLAB function
% inv(matrix) because it is numerically less robust than using the
% backslash (\) for matrix division when dealing with large matrices
zbus = inv(ybus);

% This should yeild Identity
I = ybus*zbus;

% Now to condition the numbers so that they can easily be looked at
% The result of the above multiplication should yeild the identity matrix
% but due to limited numerical precision the values may not be identically
% 1 or 0 in the matrix. This for-loop cycles through each element in the
% matrix and determines whether it is intened to be a 1 or a 0 by checking
% to see if the real parts are greater than 0.9999 to be a 1 or less than
% 0.0001 to be a zero. This makes the sanity check of calculating the
% identity
% matrix easier to see.

% Cycle through the
for j = 1:1:3*number_of_busses
    for k = 1:1:3*number_of_busses
        if real(I((j),(k))) > 0.9999
            I((j),(k)) = 1;
        elseif real(I((j),(k))) < 0.0001
            I((j),(k)) = 0;
        end
    end
end

% This separates the real and imaginary parts of the Ybus so that it can be
% put into an Excel spreadsheet to be easier to read and distribute.
realYBUS = real(ybus);
imagYBUS = imag(ybus);
printableYBUS = zeros(3*number_of_busses(1,1),6*number_of_busses(1,1));
for i = 1:1:3*number_of_busses
    printableYBUS(:,2*i-1) = realYBUS(:,i);
    printableYBUS(:,2*i) = imagYBUS(:,i);
end

end
% % FUNCTION: getDVPsystem_matrix() % % DESCRIPTION: Takes in system parameters and measurement location % information and populates the system matrix used in the % state estimation equation. % % ARGUMENTS: series_branch_impedance - the respective system's series % impedance information % shunt_branch_impedance - the respective system's shunt % impedance information % current_measurement_locations - lookup table with % Information regarding the location of the current % measurements. % voltage_measurement_locations - lookup table with % Information regarding the location of the voltage % measurements. % % OUTPUTS: k - The lower partition of the system matrix corresponding to% the relationship between the current flows and the system% state. % K - The full system matrix % % AUTHOR: Kevin D. Jones % Virginia Tech % % LAST MODIFIED: 04/02/11 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [k,K] = getDVPsystem_matrix(series_branch_impedance,...
    shunt_branch_impedance, current_measurement_locations,...
    voltage_measurement_locations)

% Populate the measurement-bus incident matrix
[A] = getDVPinidence_matrix_v2(current_measurement_locations);
disp('Populated Measurement-Bus Incidence Matrix -A-');

% Populate the series admittance matrix
[Y] = getDVPseries_matrix_v2(current_measurement_locations,...
    series_branch_impedance);
disp('Populated Series Admittance Matrix -Y-');

% Populate the shunt admittance matrix
[Ys] = getDVPshunt_matrix_v2(current_measurement_locations,...
    shunt_branch_impedance);
disp('Populated Shunt Admittance Matrix -Ys-');

% Populate the II matrix
[II] = getDVPII_matrix_v2(voltage_measurement_locations);
disp('Populated II Matrix');

% Compute the K matrix
k = Y*A + Ys;
K = [II;k];
disp('Computed K Matrix');
end
function [A] = getDVPincidence_matrix_v2(current_measurement_info)

% Determine the number of current measurements
[number_of_current_measurements, x] = size(current_measurement_info);

% Import the Current Measurement Look-Up Table
Current_Measurement_Lookup_Table = current_measurement_info;

current_measurement_numbers = Current_Measurement_Lookup_Table(:,1);
from_substation = Current_Measurement_Lookup_Table(:,4);
to_substation = Current_Measurement_Lookup_Table(:,5);

% Determine the number of busses
max_from = max(from_substation);
max_to = max(to_substation);
number_of_busses = max([max_from max_to]);

% For the non-zero entries in the matrix
One = eye(3);

% Create the space for the full A matrix and
% make it all zeros because most of the entries
% will be zero.
A = zeros(3*number_of_current_measurements,3*number_of_busses);

% Loop to cycle through the branches and
% populate the 'A' Matrix
for i = 1:1:number_of_current_measurements

  % For clarity
  CM = current_measurement_numbers(i);
  B1 = from_substation(i);
  B2 = to_substation(i);

  % From bus of measurement
  A((3*CM-2):(3*CM),(3*B1-2):(3*B1)) = One;

  % To bus of measurement
  A((3*CM-2):(3*CM),(3*B2-2):(3*B2)) = -One;

end

end
function [Y] = getDVPseries_matrix_v2(current_measurement_info, ...
    series_impedance)

% Determine the number of current measurements
[number_of_current_measurements,x] = size(current_measurement_info);

% Import the Current Measurement Look-Up Table
Current_Measurement_Lookup_Table = current_measurement_info;

current_measurement_numbers = Current_Measurement_Lookup_Table(:,1);
    from_substation = Current_Measurement_Lookup_Table(:,4);
    to_substation = Current_Measurement_Lookup_Table(:,5);
    measured_lines = Current_Measurement_Lookup_Table(:,6);

% Import the series impedance information
series_impedance_table = series_impedance;
line_numbers = series_impedance_table(:,1);
resistance = series_impedance_table(:,2);
reactance = series_impedance_table(:,3);

% Determine the number of lines
number_of_lines = max(line_numbers);

% Create an array to put 3x3 impedance matrices for each line in
% It will be 3 x 3*number_of_line
series_array = zeros(3*number_of_lines(1,1),3);

% Now populate the array with the impedance values
for i = 0:1:number_of_lines-1
    index = i*6 + 1;
    j = i + 1;
series_array(3*j-2,1) = complex(resistance(index), reactance(index));
series_array(3*j-2,2) = complex(resistance(index+1),reactance(index+1));
series_array(3*j-2,3) = complex(resistance(index+3),reactance(index+3));
series_array(3*j-1,1) = complex(resistance(index+1),reactance(index+1));
series_array(3*j-1,2) = complex(resistance(index+2),reactance(index+2));
series_array(3*j-1,3) = complex(resistance(index+4),reactance(index+4));
series_array(3*j,1) = complex(resistance(index+3),reactance(index+3));
series_array(3*j,2) = complex(resistance(index+4),reactance(index+4));
series_array(3*j,3) = complex(resistance(index+5),reactance(index+5));
end

% Calculate Admittances from Impedances
series_admittance_array = zeros(3*number_of_lines,3);
for i = 1:1:number_of_lines
    series_admittance_array(3*i-2:3*i,1:3) = inv(series_array(3*i-2:3*i,1:3));
end

% Create a space for the 'Y' matrix to sit and fill
% it with zeros since all of the off diagonal entries
% are zero
Y = zeros(3*number_of_current_measurements,3*number_of_current_measurements);
for i = 1:1:number_of_current_measurements
    m = measured_lines(i);
    Y(3*i-2:3*i,3*i-2:3*i) = series_admittance_array(3*m-2:3*m,1:3);
end
end
function [Ys] = getDVPshunt_matrix_v2(current_measurement_info, ... 
    shunt_impedance)

    % Determine the number of current measurements
    [number_of_current_measurements, x] = size(current_measurement_info);

    % Import the Current Measurement Look-Up Table
    Current_Measurement_Lookup_Table = current_measurement_info;

    current_measurement_numbers = Current_Measurement_Lookup_Table(:,1);
    from_substation = Current_Measurement_Lookup_Table(:,4);
    to_substation = Current_Measurement_Lookup_Table(:,5);
    measured_lines = Current_Measurement_Lookup_Table(:,6);

    % Import the shunt impedance information
    shunt_impedance_table = shunt_impedance;
    line_numbers = shunt_impedance_table(:,1);
    shunt_susceptance = shunt_impedance_table(:,2);

    % Determine the number of lines
    number_of_lines = max(line_numbers);

    % Determine the number of busses
    number_of_busses = max(max([from_substation to_substation]));

    % Create an array to put 3x3 impedance matrices for each line in
    % It will be 3 x 3*number_of_lines
    shunt_array = zeros(3*number_of_lines, 3);

    % Now populate the array with the impedance values
for i = 0:1:number_of_lines-1
    index = i*6 + 1;
    j = i + 1;
    shunt_array(3*j-2,1) = complex(0,shunt_susceptance(index));
    shunt_array(3*j-2,2) = complex(0,shunt_susceptance(index+1));
    shunt_array(3*j-2,3) = complex(0,shunt_susceptance(index+3));
    shunt_array(3*j-1,1) = complex(0,shunt_susceptance(index+1));
    shunt_array(3*j-1,2) = complex(0,shunt_susceptance(index+2));
    shunt_array(3*j-1,3) = complex(0,shunt_susceptance(index+4));
    shunt_array( 3*j ,1) = complex(0,shunt_susceptance(index+3));
    shunt_array( 3*j ,2) = complex(0,shunt_susceptance(index+4));
    shunt_array( 3*j ,3) = complex(0,shunt_susceptance(index+5));
end

% Create a space for the 'Ys' matrix to sit and fill
% it with zeros since most of the entries are zero
Ys = zeros(3*number_of_current_measurements,3*number_of_busses);

for i = 1:1:number_of_current_measurements
    CM = current_measurement_numbers(i);
    F = from_substation(i);
    L = measured_lines(i);
    Ys(3*CM-2:3*CM,3*F-2:3*F) = shunt_array(3*L-2:3*L,1:3);
end

end
function [II] = getDVPII_matrix_v2(voltage_measurement_info)

% Determine the number of current measurements
[number_of_voltage_measurements, x] = size(voltage_measurement_info);

% Import the Current Measurement Look-Up Table
Voltage_Measurement_Lookup_Table = voltage_measurement_info;

voltage_measurement_numbers = Voltage_Measurement_Lookup_Table(:,1);
bus_numbers = Voltage_Measurement_Lookup_Table(:,2);
substation_numbers = Voltage_Measurement_Lookup_Table(:,3);

% Determine the number of substations
number_of_substations = max(substation_numbers);

% For the non-zero entries in the matrix
One = eye(3);

% Create the space for the full A matrix and
% make it all zeros because most of the entries
% will be zero.
II = zeros(3*number_of_voltage_measurements, 3*number_of_substations);

% Populate the voltage measurement/substation incidence matrix
for i = 1:1:number_of_voltage_measurements
    VM = voltage_measurement_numbers(i);
    SN = substation_numbers(i);
    II((3*VM-2):(3*VM), (3*SN-2):(3*SN)) = One;
end
function [W, V, I] = errornormal(voltage_measurements, current_measurements)

[number_of_voltage_measurements, x] = size(voltage_measurements);
[number_of_current_measurements, x] = size(current_measurements);

% Set the variance for the voltage measurements
voltage_variance = 0.01;
% Set the variance for the current measurements
current_variance = 0.03;

% Generate random errors for the real part of the voltage
VerrorReal = voltage_variance * randn(number_of_voltage_measurements, 1);
% Generate random errors for the imaginary part of the voltage
VerrorImag = voltage_variance * randn(number_of_voltage_measurements, 1);

% Generate random errors for the real part of the current
IerrorReal = current_variance * randn(number_of_current_measurements, 1);
% Generate random errors for the imaginary part of the current
IerrorImag = current_variance * randn(number_of_current_measurements, 1);

% Create an empty covariance matrix
square = number_of_voltage_measurements + number_of_current_measurements;
W = zeros(square);

% Populate the covariance matrix with
% the voltage measurement covariance
for i = 1:1:number_of_voltage_measurements
    W(i, i) = voltage_variance^2;
    %W(i+square, i+square) = voltage_variance^2;
end

% and the current measurement covariance
for i = 1:1:number_of_current_measurements
    j = i + number_of_voltage_measurements;
    W(j, j) = current_variance^2;
    %W(j+square, j+square) = current_variance^2;
end

% Separate the real and imaginary parts of the voltage
realVoltage = real(voltage_measurements);
imagVoltage = imag(voltage_measurements);

% Separate the real and imaginary parts of the current
realCurrent = real(current_measurements);
imagCurrent = imag(current_measurements);

% Add the error to the real and imaginary parts of the voltage
realVoltage = realVoltage + VerrorReal;
imagVoltage = imagVoltage + VerrorImag;

% Add the error to the real and imaginary parts of the current
realCurrent = realCurrent + IerrorReal;
imagCurrent = imagCurrent + IerrorImag;

% Return them to a complex form
V = complex(realVoltage, imagVoltage);
I = complex(realCurrent, imagCurrent);

End
function [new_M] = updateDVPsystem_matrix_v2(M,Z,network_topology)

old_M = M;

% Import the Current Measurement Look-Up Table
Current_Measurement_Lookup_Table = current_measurement_lookup_table;

% Determine the number of current measurements
number_of_current_measurements = max(Current_Measurement_Lookup_Table(:,1));
% For clarity
ncm = number_of_current_measurements;

% Determine the number of branches
number_of_lines = max(Current_Measurement_Lookup_Table(:,6));

% Import the Voltage Measurement Look-Up Table
Voltage_Measurement_Lookup_Table = voltage_measurement_lookup_table;

% Determine the number of current measurements
number_of_voltage_measurements = max(Voltage_Measurement_Lookup_Table(:,1));
% For clarity
nvm = number_of_voltage_measurements;

for i = 1:1:number_of_lines
    if network_topology(i,4) == 0
        out_of_service_line = network_topology(i,1);
        measurement_count = 0;
        for j = 1:1:number_of_current_measurements
            if Current_Measurement_Lookup_Table(j,6) == out_of_service_line
                measurement_count = measurement_count + 1;
            end
        end
    end
end
if measurement_count == 1
    % Current Measurement Number
    l = Current_Measurement_Lookup_Table(j,1);
    % Location of Measurement
    b1 = Current_Measurement_Lookup_Table(j,4);
else if measurement_count == 2
    % Current Measurement Number
    m = Current_Measurement_Lookup_Table(j,1);
    % Location of Measurement
    bm = Current_Measurement_Lookup_Table(j,4);
end
end
end

if measurement_count == 2
    % implement the procedure that we know how to do
    K = zeros(size(H));
    K(((3*(l+nvm)-2):(3*(l+nvm)),3*b1-2:3*b1) = eye(3);
    K(((3*(m+nvm)-2):(3*(m+nvm)),3*bm-2:3*bm) = eye(3);

    S = transpose(K)*Z*K;
    S1 = Z((3*(l+nvm)-2:3*(l+nvm),3*(l+nvm)-2:3*(l+nvm));
    S2 = Z((3*(m+nvm)-2:3*(m+nvm),3*(m+nvm)-2:3*(m+nvm));
    S3 = Z((3*(l+nvm)-2:3*(l+nvm),3*(m+nvm)-2:3*(m+nvm));

    U1 = eye(3)-S1;
    U2 = eye(3)-S2;
    U3 = S3;

    T3 = symbolic_inverse(U3-U2*symbolic_inverse(U3)*U1);
    T1 = T3*U2*symbolic_inverse(U3);
    T2 = T3*U1*symbolic_inverse(U3);

    T = eye(size(S));
    T((3*b1-2:3*b1,3*b1-2:3*b1) = T1;
    T((3*bm-2:3*bm,3*bm-2:3*bm) = T2;
    T((3*b1-2:3*b1,3*bm-2:3*bm) = T3;
    T((3*bm-2:3*bm,3*b1-2:3*b1) = T3;

    new_M = old_M*(eye(3*(nvm+ncm))-K*T*transpose(K)*Z)*eye(3*(nvm+ncm))-K*transpose(K);
    old_M = new_M;
else if measurement_count == 1
    % handle this differently
end
end
end
function [test_network_series_impedance] = series_v2()

test_network_series_impedance = [
    %%%% Line 1 %%%%  
    1 0.0015927 0.0158480;  
    1 0.0012621 0.0062693;  
    1 0.0015922 0.0157600;  
    1 0.0012558 0.0073553;  
    1 0.0012557 0.0073109;  
    1 0.0015764 0.0158220;  
    %%%% Line 2 %%%%  
    2 0.0020546 0.0146110;  
    2 0.0016109 0.0061807;  
    2 0.0019251 0.0147140;  
    2 0.0016374 0.0072157;  
    2 0.0015736 0.0072661;  
    2 0.0019695 0.0146800;  
    %%%% Line 3 %%%%  
    3 0.0017831 0.0110150;  
    3 0.0015669 0.0058332;  
    3 0.0017563 0.0110300;  
    3 0.0016093 0.0058028;  
    3 0.0015903 0.0051659;  
    3 0.0018405 0.0109690;  
    %%%% Line 4 %%%%  
    4 0.00028157 0.00144660;  
    4 0.00023118 0.00066119;  
    4 0.00025302 0.00147850;  
    4 0.00023871 0.00075234;  
    4 0.00022445 0.00076813;  
    4 0.00026677 0.00146360;  
    %%%% Line 5 %%%%  
    5 0.00036673 0.0041405;  
    5 0.00029743 0.0023407;  
    5 0.00036734 0.0041399;  
    5 0.00029743 0.0023407;  
    5 0.00029772 0.0021140;  
    5 0.00036734 0.0041399;  
    %%%% Line 6 %%%%  
    6 0.0016041 0.0181690;  
    6 0.0013022 0.0097014;  
    6 0.0016111 0.0181620;  
    6 0.0013022 0.0097014;  
    6 0.0013057 0.0093858;  
    6 0.0016111 0.0181620];

end
function [test_network_shunt_impedance] = susceptance_v2()

test_network_shunt_impedance = [
    %%%% Line 1 %%%%  
    1 0.5549 
    1 -0.061993 
    1 0.55826 
    1 -0.13503 
    1 -0.13358 
    1 0.56876 
    %%%% Line 2 %%%%  
    2 0.487890; 
    2 -0.050338; 
    2 0.484280; 
    2 -0.118510; 
    2 -0.119810; 
    2 0.503440; 
    %%%% Line 3 %%%%  
    3 0.351110; 
    3 -0.080377; 
    3 0.342080; 
    3 -0.076195; 
    3 -0.034458; 
    3 0.352970; 
    %%%% Line 4 %%%%  
    4 0.0477640; 
    4 -0.0042036; 
    4 0.0460330; 
    4 -0.0103150; 
    4 -0.0110050; 
    4 0.0480700; 
    %%%% Line 5 %%%%  
    5 0.122200; 
    5 -0.027732; 
    5 0.118500; 
    5 -0.025546; 
    5 -0.011453; 
    5 0.124290; 
    %%%% Line 6 %%%%  
    6 0.563840; 
    6 -0.068671; 
    6 0.537040; 
    6 -0.075475; 
    6 -0.059400; 
    6 0.51
]
function [lookup_table] = current_measurement_lookup_table()

lookup_table = [
    % Current
    % Measurement #   From     To     From     To
    % Bus #   Bus #    Substation #    Substation#   Line #
    1           1       3           1              2            1;
    2           2       11          1              4            3;
    3           6       4           3              2            2;
    4           7       9           3              5            5;
    5           8       5           5              2            4;
    6           9       7           5              3            5;
    7           10      12          5              4            6];

function [lookup_table] = voltage_measurement_lookup_table()

% Measurement #   Bus #   Substation #
lookup_table = [1           1       1;
2           2       1;
3           6       3;
4           7       3;
5           8       5;
6           9       5;
7           10      5];
Appendix B

Topology Processor Application

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% BEGIN HELP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% FUNCTION: DVP_TOPOLOGY_PROCESSOR_v3

% DESCRIPTION: Takes line current measurements and breaker statuses and
% updates the bus/branch model of the system to reflect any
% change detected.

% ARGUMENTS: NONE

% OUTPUT: OUTPUT - The bus/branch model of the network with an additional
% column to represent service status of each branch.

% AUTHOR: Kevin D. Jones
% Virginia Tech

% LAST MODIFIED: 3/14/11

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% END HELP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [OUTPUT] = DVP_TOPOLOGY_PROCESSOR_v3()

% ____________________________________________________________
% % % % % Start program % % % % % % % % % % % % % % % % % % % % % % % % % % %

disp(''); % New line
disp('------------------- DVP Topology Processor v3 -------------------')
disp('----------------- Program Running -----------------')
disp(''); % New line

% Input the base topology information
% This information represents the full network topology and information
% regarding number of breaker and current measurements. This information
% will be stored somewhere in memory. This only has to be done once when
% the program first runs.
[number_of_current_measurements, number_of_breakers, number_of_lines,...
 full_network, current_measurement_lookup_table,...
 breaker_status_lookup_table] = getDVPtopology_data_v3();

% ____________________________________________________________

% Input the current measurements
% This input represents a set of data that will be input from openPDC
[current_measurements] = currentMeasurements();

% Check the values of the line flows to see which flows are zero. This
% would indicate that a line is out of service.
[number_of_zero_currents,zero_current_measurements,...
potential_out_of_service_lines = checkDVLine_flows_v3...
    (current_measurements, number_of_current_measurements, ...
    current_measurement_lookup_table);

% Input the breaker statuses
% This input represents a set of data that will be input from openPDC
[breaker_statuses] = breakerStatuses();

% This checks the breaker statuses to see which breakers are open
[number_of_open_breakers, open_breakers] = checkDVBreaker_status_v3...
    (number_of_breakers, breaker_statuses);

% Determine the lines that are actually out of service by comparing the
% lines showing zero current flow with the breakers that are open.
[out_of_service_lines] = checkDVNetwork_topology_v3(open_breakers,...
    number_of_breakers, number_of_lines, potential_out_of_service_lines,...
    breaker_status_lookup_table, number_of_current_measurements);

% Update the network topology to reflect any changes that have occurred.
[OUTPUT] = updateDVNetwork_topology_v3(full_network,...
    number_of_lines, out_of_service_lines);

disp(OUTPUT);

end
% FUNCTION: getDVPtopology_data_v3
% DESCRIPTION: Obtains the full network topology data, lookup tables, and
% other topology parameters from input files
% ARGUMENTS:
% OUTPUT: OUTPUT_1 - Number of current measurements in the network
% OUTPUT_2 - Number of breaker statuses in the network
% OUTPUT_3 - Number of lines in the network
% OUTPUT_4 - Full Network Topology (Line/From Bus/To Bus)
% OUTPUT_5 - Current Measurement Lookup Table
% OUTPUT_6 - Breaker Status Lookup Table
% AUTHOR: Kevin D. Jones
% Virginia Tech
% LAST MODIFIED: 10/18/10
%
function [OUTPUT_1,OUTPUT_2,OUTPUT_3,OUTPUT_4,OUTPUT_5,OUTPUT_6] = ...
    getDVPtopology_data_v3()

disp(' ');
disp('----------- Begin getDVPtopology_data_v3 -----------')
disp(' ');

% Input the number of line current measurements
% This input represents a set of data that will be input from openPDC.
% However, this could potentially be a static variable depending on what we
% want.
number_of_current_measurements = max(max(current_lookup_table));
OUTPUT_1 = number_of_current_measurements;

fprintf('%s %d %s', 'There are', number_of_current_measurements, 'current measurements in the network.' );
disp(' ');

% Input the number of breakers
% This input represents a set of data that will be input from openPDC.
% However, this could potentially be a static variable depending on what we
% want.
number_of_breakers = max(max(breaker_lookup_table));
OUTPUT_2 = number_of_breakers;

fprintf('%s %d %s', 'There are', number_of_breakers, 'breaker statuses in the network' );
disp(' ');

% Input the full network
% This input represents the network data when all lines are in service and
% should be stored in memory and easily accessed by the topology processor.
number_of_lines = max(size(busLine_List));
full_network = busLine_List();
OUTPUT_3 = number_of_lines;
OUTPUT_4 = full_network;

fprintf('%s %d %s', 'There are', number_of_lines, ... 'lines in the network');
disp(' ');
disp(' ');

disp('--- Full Network ---');
disp(' Line From To Status');
disp(full_network);
disp(' ');

% Input the current measurement look-up table
% This input represents the information that relates current measurements to
% lines and should be stored in memory and easily accessed by the topology
% processor.
current_measurement_lookup_table = current_lookUp_table();
OUTPUT_5 = current_measurement_lookup_table;

disp('--- Current Measurement Lookup Table ---');
disp(' Meas Line');
disp(current_measurement_lookup_table);
disp(' ');

% Input the breaker statuses look-up table
% This input represents the information that relates the breaker statuses
% to which lines are out of service and should be stored in memory and
% easily accessed by the topology processor.
breaker_status_lookup_table = breaker_lookUp_table();
OUTPUT_6 = breaker_status_lookup_table;

disp('--- Breaker Status Lookup Table ---');
disp(' Line BS1 BS2 BS3 BS4');
disp(breaker_status_lookup_table);
disp(' ');

disp('------------ End of getDVPtopology_data_v3 --------------')
disp(' ');
end
function [OUTPUT_1, OUTPUT_2, OUTPUT_3] = checkDVPline_flows_v3(current_measurements, ...
     number_of_current_measurements, current_measurement_lookup_table)

    disp('');
    disp('------------------------- Begin checkDVPline_flows_v3 -------------------------')
    disp('');

    zero_current_threshold = 0.1;
    number_of_zero_currents = 0;
    zero_current_measurements = zeros(number_of_current_measurements,1);
    potential_out_of_service_lines = zeros(number_of_current_measurements,1);
    out_of_service_lines = zeros(number_of_current_measurements,1);

    %------------------- Search through line currents -------------------
    for i = 1:1:number_of_current_measurements
        if current_measurements(i,2) <= zero_current_threshold
            number_of_zero_currents = number_of_zero_currents + 1;
            zero_current_measurements(i) = current_measurements(i,1);
            potential_out_of_service_lines(i) = current_measurement_lookup_table(i,1);
            out_of_service_lines(i) = current_measurement_lookup_table(i,1);
        end
    end

    OUTPUT_1 = number_of_zero_currents;
    OUTPUT_2 = potential_out_of_service_lines;
    OUTPUT_3 = out_of_service_lines;
% What is the associated measurement number of the zero current measurement?
zero_current_measurements(number_of_zero_currents,1) = ...
current_measurements(i,1);

% What is the associated line number of the zero current measurement?
potential_out_of_service_lines(number_of_zero_currents,1) = ...
current_measurement_lookup_table(i,2);

% Print to the MATLAB window
% 'Current Measurement X through line Y is ZERO'
fprintf('%s %d %s %d %s','Current Measurement',current_measurements(i,1),...
' through line',current_measurement_lookup_table(i,2), 'is ZERO');
disp(' ');
end
end

% Another redundancy check to see if all of the current measurements on the potential out of service lines are zero
for i = 1:1:number_of_current_measurements

% Search through the potential out of service lines. Redundancy comes from multiple current measurements on each line.
check_line = potential_out_of_service_lines(i,1);

% A counter to count instances of matching lines in potential_out_of_service_lines
counter1 = 0;

% A counter to count instances of matching lines in current_measurement_lookup_table
counter2 = 0;

% Only check actual lines
if check_line ~= 0

    for j = 1:1:number_of_current_measurements

        % Count the redundancy of potential out of service lines
        if potential_out_of_service_lines(j,1) == check_line
            counter1 = counter1 + 1;
        end

        % Count the redundancy of lines in the current measurement lookup table
        if current_measurement_lookup_table(j,2) == check_line
            counter2 = counter2 + 1;
        end

    end
end
end

% If the number of times the line in question appears in
% potential_out_of_service_lines is equal to the number of times that
% it appears in the current measurement lookup table
if counter1 == counter2

  % Then add the line to the list of out of service lines because
  % the current measurements fully indicate that the line is out of
  % service.
  out_of_service_lines(i,1) = check_line;

  % Eliminate this line number from potential_out_of_service_lines
  % so that it will not be checked again since it has already been
  % verified.
  for k = 1:number_of_current_measurements
    if potential_out_of_service_lines(k,1) == check_line
      potential_out_of_service_lines(k,1) = 0;
    end
  end

else

  % Indicates that there is a line that appears to be out but has
  % current measurements that disagree with each other.
  out_of_service_lines(i,1) = 9999;
end
end

end % Output the number of current measurements that show zero line flow
OUTPUT_1 = number_of_zero_currents;
end % Output the current measurement numbers that show zero line flow
OUTPUT_2 = zero_current_measurements;
end % Output the lines that have zero line flow
OUTPUT_3 = out_of_service_lines;

disp(' '); disp('--------- End of checkDVPline_flows_v3 ---------'); disp(' '); end
function [OUTPUT_1, OUTPUT_2] = checkDVPbreaker_status_v3(number_of_breakers, ...  
    breaker_statuses)
    disp(' ');  
    disp('---------- Begin checkDVPbreaker_status_v3 ----------');
    disp(' ');  
    % Zero number_of_open_breakers so that we can count in a loop
    number_of_open_breakers = 0;
    
    % Create a space to list the breakers that are open; sized for all breakers
    open_breakers = zeros(number_of_breakers,1);
    
    %------------------- Search through breaker statuses ---------------------%
    for i = 1:1:number_of_breakers
        % If the breaker is open (0 represents an open breaker and 1 represents
        % a closed breaker. However, in practice this may be reversed).
        if breaker_statuses(i,2) == 0
            % Count the number of open breakers in the network
            number_of_open_breakers = number_of_open_breakers + 1;
            
            % Keep track of which breakers are open
            open_breakers(number_of_open_breakers,1) = breaker_statuses(i,1);
            
            % Print information to the MATLAB window
            fprintf('%s %d %s', 'Breaker ', breaker_statuses(i,1), ' is open');
            disp(' ');  
        end % if
    end % for

    % If the breaker is not open, then it is closed
else
    % Print information to the MATLAB window
    %...
fprintf('%s %d %s', 'Breaker ', breaker_statuses(i,1), ' is closed');
disp(' ');
end

% Print information to the MATLAB window
disp(' ');
if number_of_open_breakers ~= 1
    fprintf('%s %d %s', 'There are', number_of_open_breakers, 'breakers open');
    disp(' ');
else
    fprintf('%s %d %s', 'There is', number_of_open_breakers, 'breaker open');
    disp(' ');
end

OUTPUT_1 = number_of_open_breakers;
OUTPUT_2 = open_breakers;

disp(' ');
disp('----------- End of checkDVPbreaker_status_v3 -----------')
disp(' ');
end
% Function: checkDVPnetwork_topology_v3
% Description: Compares line flows with breaker statuses to determine which lines are actually out of service
% Arguments: number_of_zero_currents - Number of current measurements that show zero line current
%   number_of_lines - Number of lines in the network
%   number_of_breakers - Number of breakers in the network
%   open_breakers - A column vector of the numbers of the open breakers
%   zero_current_measurements - A column vector of the current measurement numbers who show zero line current
%   current_measurement_lookup_table - A lookup table relating current measurements to the line that they measure
%   breaker_status_lookup_table - A lookup table relating line numbers to the breaker numbers that must be open for the line to be out of service
% Output: OUTPUT - A column vector of the lines that are out of service
% Author: Kevin D. Jones
% Virginia Tech
% Last Modified: 3/14/11

function [OUTPUT] = checkDVPnetwork_topology_v3(open_breakers,number_of_breakers,...
  number_of_lines, potential_out_of_service_lines,...
  breaker_status_lookup_table,number_of_current_measurements)

% Create a space for the list of actual out of service lines
actual_out_of_service_lines = zeros(number_of_lines,1);
% Create a space for the breakers that correspond to the potential out % of service lines
check_breaker_vectors = zeros(number_of_current_measurements,5);

% Set the loop index
index = 0;

for i = 1:1:number_of_current_measurements
  % Only check the potential out of service lines
  if potential_out_of_service_lines(i,1) ~= 0
    % Increment the Index
    index = index + 1;
    % For clarity
    line = potential_out_of_service_lines(i,1);
    % Populate a set of vectors that contain the potential out of

% service line and the breakers that must be open for that line
% to actually be out of service.
check_breaker_vectors(index,1) =
potential_out_of_service_lines(i,1);
check_breaker_vectors(index,2) =
braker_status_lookup_table(line,2);
check_breaker_vectors(index,3) =
braker_status_lookup_table(line,3);
check_breaker_vectors(index,4) =
braker_status_lookup_table(line,4);
check_breaker_vectors(index,5) =
braker_status_lookup_table(line,5);
end
end

for i = 1:1:number_of_current_measurements
  % Only check the vectors which have potentially out of service lines
  if check_breaker_vectors(i,1) ~= 0
    % Initialize the count
    breaker_count = 0;
    % Check the breaker statuses to see if all of the breakers that
    % are required to be open are actually open.
    for j = 2:1:5
      breaker = check_breaker_vectors(i,j);
      for k = 1:1:number_of_breakers
        if open_breakers(k,1) == breaker
          breaker_count = breaker_count + 1;
        end
      end
    end
    % If all of the required breakers are open then output the given
    % line as an actual out of service line.
    if breaker_count == 4
      actual_out_of_service_lines(i,1) = check_breaker_vectors(i,1);
    else
      % Otherwise, output some error message to the command window to
      % indicate where the discrepancy is located.
      fprintf('%s %d %s', 'Line flows indicate line', ...'
is out of service.');
disp(' ');
      fprintf('%s %d %s %d %s %d %s', 'However, breakers', ...'
check_breaker_vectors(i,2),',','...
check_breaker_vectors(i,3),',','...
check_breaker_vectors(i,4),', and',...,check_breaker_vectors(i,5),...
'are not all open.');
disp(' ');
    end
  end
end

% Set the function output
OUTPUT = actual_out_of_service_lines;
End
function [OUTPUT] = updateDVPnetwork_topology_v3(network_topology,...
number_of_lines,out_of_service_lines)

% Cycle through the list of out of service lines and update the status in
% the network topology accordingly
for i = 1:1:number_of_lines

    line = out_of_service_lines(i,1);
    if line ~= 0
        % Make the status of the out of service line 0
        network_topology(line,4) = 0;
    end

end

OUTPUT = network_topology;
end
function [full_network] = busLine_List()

full_network = [
    % Line Number  From Bus  To Bus  Status
    1             1         2       1;
    2             2         3       1;
    3             1         4       1;
    4             2         5       1;
    5             3         5       1;
    6             4         5       1];

function [lookUp_table] = breaker_lookUp_table()

lookUp_table = [
    % Line Number  BS1  BS2  BS3  BS4
    1             1    2    4    5;
    2             4    7    8    9;
    3             2    3    11   12;
    4             6    7    14   15;
    5             9   10   14   16;
    6            12   13   15   17];

function [lookUp_table] = current_lookUp_table()

lookUp_table = [
    % Current Meas  Line Number
    1             1;
    2             1;
    3             2;
    4             2;
    5             3;
    6             3;
    7             4;
    8             4;
    9             5;
   10             5;
   11             6;
   12             6];
function [topologyProcessor_Input] = breakerStatuses()

topologyProcessor_Input = [
    % Breaker Number     Status
    1             1;
    2             0;
    3             0;
    4             1;
    5             1;
    6             1;
    7             1;
    8             1;
    9             1;
    10            1;
    11            0;
    12            0;
    13            1;
    14            1;
    15            1;
    16            1;
    17            1];

function [topologyProcessor_Input] = currentMeasurements()

topologyProcessor_Input = [
    % Current Measurement Number     Magnitude Value
    1                            1;
    2                            1;
    3                            1;
    4                            1;
    5                            0;
    6                            0;
    7                            1;
    8                            1;
    9                            1;
    10                           1;
    11                           1;
    12                           1];
Appendix C

WLS State Estimation Matlab Code

% Traditional WLS State Estimation Example
clear
clc

measurements = measurement_data_2();
network = network_data();
measurement_type = measurements(:,2);
measurement_locations = [measurements(:,3),measurements(:,4)];
z = measurements(:,5);
variances = measurements(:,6);
[number_of_measurements,temp] = size(measurements);

% Create a space for the Covariance Matrix
R = zeros(number_of_measurements);

% Assume a flat start
x = [0,0,0,0,1,1,1,1,1];
% Use the flat start to identify the angles and magnitudes
variable_type = x;

[number_of_state_variables,temp] = size(x');
number_of_state_magnitudes = (number_of_state_variables + 1)/2;
number_of_state_angles = ...
    number_of_state_variables - number_of_state_magnitudes;

[number_of_lines,temp] = size(network);
number_of_busses = max(max(network(:,1),network(:,2)));

% Display Measurement Information
disp('--------------------------------- Measurement Information ---------------');
disp(' Measurement Type From To Value Variance');
disp(measurements);

% Display Network Information
disp('------------------------------- Network Information -------------------');
disp(' From To Resistance Reactance Susceptance');
disp(network);

% Populate the Admittance Matrix
ybus = populate_ybus(network,number_of_busses,number_of_lines);
disp('----------------------------- Admittance Matrix ------------------------');
disp(ybus);

% Populate the Covariance Matrix
for i = 1:1:number_of_measurements
R(i,i) = variances(i)*variances(i);
end

% Initialize the tolerance value
tolerance = 100;

% Start the WLS Algorithm
while tolerance > 0.0001

% Calculate the Measurement Function
h = update_measurement_function(...
  measurement_type, measurement_locations, ybus, ...
  x_old, number_of_state_magnitudes, number_of_state_angles);

% Populate the Jacobian Matrix
H = update_jacobian_matrix(...
  variable_type, measurement_type, measurement_locations, ybus, ...
  x_old, number_of_state_magnitudes, number_of_state_angles, ...
  number_of_measurements);

% Calculate the Gain Matrix
G = transpose(H)*(R\eye(size(R)))*H;

x_new = x_old + inv(transpose(H)*inv(R)*H)*(transpose(H)*inv(R)*(z-h));

tolerance = max(abs(x_old-x_new));

x_old = x_new;
end

% Display the state variable
disp('-------------------Solution-------------------');
disp('Angles are in radians');
disp('State Vector');
disp(x_old);
function [ybus] = populate_ybus(network_data,...
    number_of_busses,number_of_lines)

from_busses = network_data(:,1);
to_busses = network_data(:,2);
resistances = network_data(:,3);
reactances = network_data(:,4);
%susceptances = network_data(:,5);

% Calculate Admittances from Impedances
for i = 1:1:number_of_lines
    series_admittances(i) = 1/complex(resistances(i),reactances(i));
end

%Create a space for the Ybus
ybus = zeros(number_of_busses);

%Populate Off-Diagonal Elements
% Each off diagonal entry is the negative of the admittance connecting the
% two associated busses. If there is no connection then the entry is zero
for m = 1:1:number_of_lines
    ybus((from_busses(m)),(to_busses(m))) = -series_admittances(m);
    ybus((to_busses(m)),(from_busses(m))) = -series_admittances(m);
end

%Populate Diagonal Elements
nextDiagonalEntry = 0;
% The diagonal entry is the sum of all of the admittances connected to the
% associated bus.
for n = 1:1:number_of_busses
    for o = 1:1:number_of_lines
        if ((from_busses(o) == n) || (to_busses(o) == n))
            nextDiagonalEntry = nextDiagonalEntry + series_admittances(o);
        end
    end
    ybus((n),(n))= nextDiagonalEntry;
    nextDiagonalEntry = 0;
end
end
function [output] = update_measurement_function(...
    measurement_type, measurement_location, ybus, ...
    state_vector, number_of_state_magnitudes, number_of_state_angles)

is_a_real_power_injection = 1;
is_a_real_power_flow = 2;
is_a_reactive_power_injection = 3;
is_a_reactive_power_flow = 4;
is_a_current_injection = 5;
is_a_voltage_magnitude = 6;

number_of_state_variables = ...
    number_of_state_angles + number_of_state_magnitudes;
G = real(ybus);
B = imag(ybus);
V = state_vector(number_of_state_magnitudes:number_of_state_variables);
O = state_vector(1:number_of_state_angles);
O = [0; O];

[number_of_measurements, temp] = size(measurement_type);

h = zeros(number_of_measurements, 1);

for i = 1:1:number_of_measurements

    if measurement_type(i) == is_a_real_power_injection
        m = measurement_location(i, 1);

        for k = 1:1:number_of_state_magnitudes
            h(i) = h(i) + V(k) * (G(m, k) * cos(O(m) - O(k)) + B(m, k) * sin(O(m) - O(k)));
        end
        h(i) = h(i) * V(m);
    elseif measurement_type(i) == is_a_real_power_flow
        m = measurement_location(i, 1);
        n = measurement_location(i, 2);

        h(i) = V(m) * V(m) * (-G(m, n)) - V(m) * V(n) * ...
                (-G(m, n) * cos(O(m) - O(n)) - B(m, n) * sin(O(m) - O(n)));
    elseif measurement_type(i) == is_a_reactive_power_injection
        m = measurement_location(i, 1);

        for k = 1:1:number_of_state_magnitudes
            h(i) = h(i) + V(k) * (G(m, k) * sin(O(m) - O(k)) - B(m, k) * cos(O(m) - O(k)));
        end
        h(i) = h(i) * V(m);
    elseif measurement_type(i) == is_a_reactive_power_flow
        m = measurement_location(i, 1);
        n = measurement_location(i, 2);

        h(i) = -V(m) * V(m) * (-B(m, n)) - V(m) * V(n) * ...
(-G(m,n) \sin(0(m) - 0(n)) + B(m,n) \cos(0(m) - 0(n)));

elseif measurement_type(i) == is_a_current_injection

elseif measurement_type(i) == is_a_voltage_magnitude
    m = measurement_location(i);
    h(i) = V(m);
end

end

output = h;
function [output] = update_jacobian_matrix(...
    state_variable_type, measurement_type, measurement_location, ybus, ...
    state_vector, number_of_state_magnitudes, number_of_state_angles, ...
    number_of_measurements)

is_a_real_power_injection = 1;
is_a_real_power_flow = 2;
is_a_reactive_power_injection = 3;
is_a_reactive_power_flow = 4;
is_a_current_injection = 5;
is_a_voltage_magnitude = 6;

number_of_state_variables = ...
    number_of_state_angles + number_of_state_magnitudes;

G = real(ybus);
B = imag(ybus);
V = state_vector(number_of_state_magnitudes: number_of_state_variables);
O = state_vector(1: number_of_state_angles);
O = [0; O];

H = zeros(number_of_measurements, number_of_state_variables);

for i = 1:1: number_of_measurements
    for j = 1:1: number_of_state_angles
        if measurement_type(i) == is_a_real_power_injection
            m = measurement_location(i, 1);
            if m == j + 1
                for k = 1:1: number_of_state_magnitudes
                    H(i, j) = H(i, j) + V(m) * V(k) * (...  
                        -G(m, k) * sin(O(m) - O(k)) + B(m, k) * cos(O(m) - O(k)));  
                end
                H(i, j) = H(i, j) - V(m) * V(m) * B(m, m);
            else
                H(i, j) = V(m) * V(j + 1) * (...  
                        G(m, j + 1) * sin(O(m) - O(j + 1)) - B(m, j + 1) * cos(O(m) - O(j + 1)));  
            end
        elseif measurement_type(i) == is_a_real_power_flow
            m = measurement_location(i, 1);
            n = measurement_location(i, 2);
            if m == j + 1
                H(i, j) = V(m) * V(n) * (...  
                        -G(m, n) * sin(O(m) - O(n)) + B(m, n) * cos(O(m) - O(n)));  
            elseif n == j + 1
                H(i, j) = -V(m) * V(n) * (...  
                        -G(m, n) * sin(O(m) - O(n)) + B(m, n) * cos(O(m) - O(n)));  
            end
        elseif measurement_type(i) == is_a_reactive_power_injection
            m = measurement_location(i, 1);
            if m == j + 1
                for k = 1:1: number_of_state_angles + 1
                    H(i, j) = H(i, j) + V(m) * V(k) * (...  
                        G(m, k) * cos(O(m) - O(k)) + B(m, k) * sin(O(m) - O(k)));  
                end
                H(i, j) = H(i, j) - V(m) * V(m) * G(m, m);
            end
        end
    end
end
else
    H(i,j) = V(m)*V(j+1)*
        (-G(m,j+1)*cos(O(m)-O(j+1))-B(m,j+1)*sin(O(m)-
        O(j+1)));
end

elseif measurement_type(i) == is_a_reactive_power_flow
    m = measurement_location(i,1);
    n = measurement_location(i,2);
    if m == j+1
        H(i,j) = -V(m)*V(n)*
            (G(m,n)*cos(O(m)-O(n))+B(m,n)*sin(O(m)-O(n)));
    elseif n == j+1
        H(i,j) = -V(m)*V(n)*
            (G(m,n)*cos(O(m)-O(n))-B(m,n)*sin(O(m)-O(n)));
    end
elseif measurement_type(i) == is_a_current_injection
elseif measurement_type(i) == is_a_voltage_magnitude
    H(i,j) = 0;
end
end
end

for i = 1:1:number_of_measurements
    for j = 1:1:number_of_state_magnitudes
        s = j+number_of_state_angles;
        if measurement_type(i) == is_a_real_power_injection
            m = measurement_location(i,1);
            if m == j
                for k = 1:1:number_of_state_magnitudes
                    H(i,s) = H(i,s) + V(k)*
                        (G(m,k)*cos(O(m)-O(k))+B(m,k)*sin(O(m)-O(k)));
                end
                H(i,s) = H(i,s) + V(m)*G(m,m);
            else
                H(i,s) = V(m)*
                    (G(m,j)*cos(O(m)-O(j))+B(m,j)*sin(O(m)-O(j)));
            end
        elseif measurement_type(i) == is_a_real_power_flow
            m = measurement_location(i,1);
            n = measurement_location(i,2);
            if m == j
                H(i,s) = -V(n)*
                    (-G(m,n)*cos(O(m)-O(n))-B(m,n)*sin(O(m)-O(n)))+...
                    2*(-G(m,n))*V(m);
            elseif n == j
                H(i,s) = -V(m)*
                    (-G(m,n)*cos(O(m)-O(n))-B(m,n)*sin(O(m)-O(n)));
            end
        elseif measurement_type(i) == is_a_reactive_power_injection
            m = measurement_location(i,1);
            if m == j
                for k = 1:1:number_of_state_angles+1
                    H(i,s) = H(i,s) + V(k)*
                        (G(m,k)*sin(O(m)-O(k))-B(m,k)*cos(O(m)-O(k)));
                end
                H(i,s) = H(i,s) - V(m)*B(m,m);
            else

            end
end
end
\[ H(i, s) = \frac{V(m) \times (G(m, j) \times \sin(\theta(m) - \theta(j)) - B(m, j) \times \cos(\theta(m) - \theta(j)))}{2} \]

else if \( \text{measurement\_type}(i) == \text{is\_a\_reactive\_power\_flow} \)

\[ m = \text{measurement\_location}(i, 1); \]
\[ n = \text{measurement\_location}(i, 2); \]
\[ \text{if } m == j \]
\[ H(i, s) = -V(n) \times \left( -G(m, n) \times \sin(\theta(m) - \theta(n)) + B(m, n) \times \cos(\theta(m) - \theta(n)) \right) - 2 \times V(m) \times (-B(m, n)); \]
\[ \text{elseif } n == j \]
\[ H(i, s) = -V(m) \times \left( -G(m, n) \times \sin(\theta(m) - \theta(n)) + B(m, n) \times \cos(\theta(m) - \theta(n)) \right); \]
\[ \text{end} \]

else if \( \text{measurement\_type}(i) == \text{is\_a\_current\_injection} \)

else if \( \text{measurement\_type}(i) == \text{is\_a\_voltage\_magnitude} \)

\[ m = \text{measurement\_location}(i, 1); \]
\[ \text{if } m == j \]
\[ H(i, s) = 1; \]
\[ \text{else} \]
\[ H(i, s) = 0; \]
\[ \text{end} \]
\[ \text{end} \]

end

end

output = H;

end
function [output] = network_data()
% From  To  R    jX  2B
output = [ 1  2  0.02  0.06 0.0;
           1  4  0.01  0.08 0.0;
           2  3  0.05  0.10 0.0;
           2  5  0.02  0.07 0.0;
           3  5  0.01  0.07 0.0;
           4  5  0.03  0.09 0.0];
end

function [output] = measurement_data_2()
% Measurement Type
% 1 - Real Power Injection
% 2 - Real Power Flow
% 3 - Reactive Power Injection
% 4 - Reactive Power Flow
% 5 - Current Injection
% 6 - Voltage Magnitude
% Measurement  Type  From  To   Value  Covariance
output = [ 1  2  1  2  0.3132  0.008;
           2  2  1  4  0.037  0.008;
           3  2  3  5  0.0127  0.008;
           4  2  4  5  0.0004  0.008;
           5  1  1  1  -0.0321  0.010;
           6  1  2  2  0.052  0.010;
           7  1  3  3  0.018  0.010;
           8  4  1  2  0.9396  0.008;
           9  4  1  4  0.2960  0.008;
          10  4  3  5  0.0888  0.008;
          11  4  4  5  0.0011  0.008;
          12  3  3  3  0.0080  0.010;
          13  3  4  4  -0.20592  0.010;
          14  3  5  5  -0.004  0.010;
          15  6  1  1  1.0120  0.004;
          16  6  2  2  1.003  0.004;
          17  6  5  5  1.001  0.004];
end
function [output] = measurement_data()

% Measurement Type

% 1 - Real Power Injection
% 2 - Real Power Flow
% 3 - Reactive Power Injection
% 4 - Reactive Power Flow
% 5 - Current Injection
% 6 - Voltage Magnitude

output = [
    1  2  1  2  0.3132  0.008;
    2  2  1  4  0.037   0.008;
    3  2  2  3  0.0001  0.008;
    4  2  2  5  0.0274  0.008;
    5  2  3  5  0.0127  0.008;
    6  2  4  5  0.0004  0.008;
    7  1  1  1 -0.0321  0.010;
    8  1  2  2  0.052   0.010;
    9  1  3  3  0.018   0.010;
   10 1  4  4  0.0000  0.010;
   11 1  5  5 -0.037   0.010;
   12 4  1  2  0.9396  0.008;
   13 4  1  4  0.2960  0.008;
   14 4  2  3  0.0002  0.008;
   15 4  2  5  0.0959  0.008;
   16 4  3  5  0.0888  0.008;
   17 4  4  5  0.0011  0.008;
   18 3  1  1  0.3370  0.010;
   19 3  2  2 -0.13125 0.010;
   20 3  3  3  0.0080  0.010;
   21 3  4  4 -0.20592 0.010;
   22 3  5  5 -0.004   0.010;
   23 6  1  1  1.0120  0.004;
   24 6  2  2  1.003   0.004;
   25 6  3  3  1.0022  0.004;
   26 6  4  4  0.9980  0.004;
   27 6  5  5  1.001   0.004];
end