12-2006

HIERARCHICAL STATE ESTIMATION FOR WIDE AREA POWER SYSTEMS

Srivatsan Lakshminarasimhan
Clemson University, slakshm@osii.com

HIERARCHICAL STATE ESTIMATION APPLIED TO A WIDE AREA POWER SYSTEMS

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

---

by
Srivatsan Lakshminarasimhan
December 2006

Accepted by:
Dr. Adly Girgis, Committee Chair
Dr. Elham Makram
Dr. Carl Baum

ABSTRACT

This thesis presents the application of hierarchical state estimation techniques to consolidate the state output of a wide area power system network. In a wide area network a large number of interconnections exist between various utilities of the wide area. Power transactions between areas occur over large distances and hence for better security there is a need to monitor the state of the entire wide area systems. Hierarchical state estimation is preferred over integrated state estimation, due to the reduced computational time.

Using existing state estimators of the member utilities of the wide area in the bottom level of hierarchical state estimation proves to be economical. The coordination level alone needs to be done using the state estimation output of the member areas for obtaining the overall state estimate. In this thesis a modified coordination technique derived from hierarchical state estimation is proposed to consolidate the state outputs of all the individual entities in the wide area. Issues due to heterogeneity of the estimators in each member utility of the wide area have been identified and addressed. A modification to the traditional hierarchical state estimation approach has been proposed which overcomes issues of delay and loss of state outputs from the estimators in the wide area. Use of synchronized phasor measurements in the hierarchical structure has been studied.

The coordination algorithms have been tested on an IEEE 118 bus system by splitting the system into smaller areas. The results of the algorithms have been analyzed on the basis of accuracy and speed. The results of these algorithms have been compared

to integrated state estimation of the wide area. The results show that the coordination algorithm is four times faster than the integrated state estimator without sacrificing the level of accuracy. To account for the issues regarding the delay of state output arrivals and the absence of state outputs from an area, the coordination algorithm of the hierarchical state estimation technique has been modified. As might be expected, the modified coordination algorithm decreases the accuracy of the overall state estimate. In contrast, the use of synchronized phasor measurements in all the levels of the hierarchical state estimator increases the confidence of the overall estimate apart from increasing the performance of the estimation process.

ACKNOWLEDGMENTS

I owe the greatest debts to Dr. Adly Girgis for providing me an opportunity to work under him and for his invaluable guidance during the entire course of my research at Clemson University. Without his patience and support, the research for this thesis would have never been possible.

I extend my sincere thanks and gratitude to my committee members Dr. Elham Makram and Dr. Carl Baum for providing the needed insight and time without which the thesis would have never been possible.

I would also like to extend my thanks to my colleagues Manish Patel and Aftab Alam for their continued encouragement and constructive criticism. Finally I like to thank my Parents and My Sister for their constant encouragement and moral support.

TABLE OF CONTENTS

Table of Contents (Continued)

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Motivation

Close monitoring of power system transactions occurring over a wide area is required to facilitate secure and reliable operation. Due to the complexity of the system, many vital power system operations such as contingency analysis, transmission capability studies, transient stability analysis etc. needs to be studied over the whole area comprising many interconnected utilities. For all of these operations, the main starting point is state estimation. Since the vital power systems operation is done over a wide area, state estimation also needs to be done for the wide area. State estimation for a wide area comprising of number of utilities is not straightforward as it involves many issues.

The volume of interconnections and number areas make state estimation difficult. To reduce the computational time of state estimation of the wide area, often a hierarchical structure of operation is preferred. In a hierarchical structure the individual areas perform the local operations, and in the higher levels the operations involving more than one area in a hierarchy are performed.

Hierarchical state estimation is not new, but the main focus has been mostly on reduction in computational time, accuracy and reduction of problem size [1-4]. Detailed study of the hierarchical state estimation is presented in the following chapters.

In this thesis focus has been given on the use of existing state estimators in a hierarchical structure, and the issues interest include the effects of inconsistencies of transfer of data upwards in the hierarchy, the effects of lack of data, and the use of synchronized phasor measurements.

Objective

There is a need for monitoring the state of consolidated power systems comprising of multiple areas. Hierarchical state estimation has been used for consolidation of state outputs. Hierarchical state estimation comprises of multiple estimation levels. The higher estimation levels are called coordination levels because the outputs from lower levels are consolidated into the total output of the hierarchy. The main advantage of the application of hierarchical state estimation for multi-area state estimation is that the existing local state estimators can be used. Hence, for overall visibility of wide area systems, coordination of the local estimates is all that is required. In this thesis, the focus is on how the coordination can be done effectively, identifying issues associated with the implementation of the coordinator and providing solutions to overcome these issues. The issues arise due to the dissimilarity of the local state estimators. These issues have been identified as asynchronism of data arrival, delay of data arrival, and loss of state outputs.

Contribution

In this thesis hierarchical state estimation is applied to achieve wide area state estimation. Issues of incoherencies of data arrival, delay in state output arrival and

absence of data specifically on a wide area system have been studied. The hierarchical state estimation algorithm has been modified to accommodate these issues. Finally, the role of synchronized phasor measurements in hierarchical state estimation structure has been studied. It is seen that hierarchical state estimation is faster than integrated state estimation for application in a large area. Furthermore, synchronized phasor measurements increase the performance of the hierarchical state estimator. In particular, use of synchronized phasor measurements improves the confidence of the coordination stage. Hence, identification of the major issues in application of hierarchical state estimation to wide area power systems and alleviation of these issues with and without synchronized phasor measurements are considered as the contribution of this work.

Weighted least square method

This section explains the basics of weighted least square method used for power system state estimation. The commonly available measurements for state estimation are power flows, voltage magnitudes, and power injections. For state estimation the measurements are collected using supervisory control and data acquisition (SCADA). SCADA measurements are not free from errors. The errors can be in the form of noise in measurements, bad measurements, and wrong circuit connection information (more often called topology information).

State estimation involves following major functions:

1. Topology processing: this function involves obtaining the model of the system based on status of the circuit breaker, tap positions of transformers, parameters of transmission lines, etc.

2. Observability analysis: this function involves checking if the available SCADA measurements are sufficient to find the state of the system. If the SCADA measurements are not sufficient, pseudo measurements can be used based on forecasted data or previous state data to make the system observable.
3. State estimation: this function obtains the best estimate of the system state using the SCADA measurements and the topology information.
4. Bad data processing: this function checks for the possible bad measurements. If any bad measurements are detected, they are removed from the measurement set and the state estimation is repeated again.

The best estimate of the system is obtained using weighted least squares (WLS). In state estimation, voltage magnitudes and voltage angles are estimated using the measurements and topology information obtained from the SCADA. The typical measurements used for state estimation are power flows (both active and reactive), voltage magnitudes at the buses, power injections at each bus both active and reactive, and current injections.

Mathematical formulation of state estimation using wls

The redundancy of measurements is very important in state estimation. The better the redundancy of good measurements, the more accurate the state estimate and the faster the convergence of the estimator. The relationship between the measurements and the state vector is a nonlinear function. Also the errors are assumed to have a Gaussian distribution. Hence the measurement equation can be represented as follows [5]

$$z_m = h_m(x) + e_m$$

where, Z_m is the measurement vector, X is the state vector, $h_m(x)$ is the measurement function, and e_m is the random measurement error. The state vector includes voltage magnitudes and angles and hence can be taken as follows

$$x = [\delta_1, \delta_2, ..., \delta_N, V_1, V_2, V_3, ..., V_N]$$

where δ_1, $\delta_2 \ldots \delta_N$ are voltage angles of buses and V_1, $V_2 \ldots V_N$ are voltage magnitudes of all the buses and N is maximum number of buses.

The Jacobian matrix for the measurement vector is formed which is denoted here as H. Now the common assumption in the WLS method is that the measurement errors are independent. This makes the measurement error covariance matrix a diagonal matrix. The main idea behind WLS estimation is the square of the measurement deviation from the initial estimate is minimized to obtain the best estimate. Hence the objective function is of the form [5]

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

$$= [z - h(x)]^T R^{-1} [z - h(x)]$$

where R is the measurement error covariance matrix and R_{ij} is the ith row and jth coloumn of the matrix

In order to solve the above equation, the first order optimality conditions must be satisfied. These can be expressed as follows [5]:

$$g(x) = \frac{\partial J(x)}{\partial x} = -H^T(x)R^{-1}[z - h(x)] = 0$$

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

The above non-linear equation can be solved via an iterative Gauss-Newton method as shown below [5]:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^{k} - [\boldsymbol{G}(\boldsymbol{x}^{k})]^{-1} * \boldsymbol{g}(\boldsymbol{x}^{k})$$

$$G(x^k) = \frac{\partial g(x^k)}{\partial x} = H^T(x^k)R^{-1}H(x^k)$$

$$g(x^k) = -H^T(x^k)R^{-1}[z - h(x^k)]$$

The gain matrix is *G(x)*. If the system if fully observable, the gain matrix will be positive definite and symmetric. For each iteration, the gain matrix is decomposed into its triangular factors and the forward/backward substitution method is used to solve for following linear set of equations, giving

$$[\boldsymbol{G}(\boldsymbol{x}^{k})]\Delta\boldsymbol{x}^{k+1} = \boldsymbol{H}^{T}(\boldsymbol{x}^{k})\boldsymbol{R}^{-1}[\boldsymbol{z} - \boldsymbol{h}(\boldsymbol{x}^{k})]$$

$$\Delta x^{k+1} = x^{k+1} - x^{k}$$

References [5-10] describe the state estimation methods of solution.

State estimation methodologies

State estimation is a vital part of power system operation as it is often the starting point for many applications. But in its implementation an important challenge is the increasing size and complexity of the power system. For this reason, state estimation tends to be slow and hence, for almost four decades, research has been conducted on how to improve the computational speed of the state estimation process. Communication requirements are also one of the important constraints in the implementation of the state estimators. This section gives a detailed review of the past research work in this area.

State estimation was first introduced by Schweppe and Wilde [11] to power systems. The method used is the regular static state estimation as descirbed in the previous section. This method has been fundamental for all other algorithms and the majority of the state estimators still use weighted least squares.

An important point to be noted here is that the state estimation is run repeatedly and it is better that the state vector always has the state of the system in the previous state, as the initial value, rather than using a flat start for every run.

The idea of using dynamic state estimation was examined shortly after static state estimation was introduced by Schweppe [12]. Many of the dynamic state estimation techniques follow a prediction correction method. Prediction is very useful tool as it gives the operator a rough estimate of one state estimation step in the future. One of the methods by J.F de Queiroz, M.B. Do Coutto Filho et al [13], deals with state forecasting. This method uses statistical techniques such as Holt's two parametric linear exponential smoothing to predict the state of the systems as the load changes. Hence the generation always follows a rough cyclic pattern.

After the advent of various algorithmic enhancements such as parallel computation, distributed processing, hierarchical methods etc., techniques were examined to apply these methods to state estimation algorithm for better computational speed of state estimation for large systems. Some of these methods are explained here. Most of these methods compromise on the accuracy to obtain a sub-optimal solution for the state of the system which is faster to obtain than the optimal solution. One of the main approaches for this has been parallel and distributed processing. A decentralized approach was proposed by Van Custem et al [1,2] as early as 1981. The decentralized approach actually splits the system into smaller independent areas and estimates the state of each segment and then finally obtains the optimal estimate of the entire area from the smaller areas. Parallel execution deals with the running of the estimation process simultaneously in different systems so as to arrive at the optimal estimate faster. Hierarchical estimation technique is a hybrid of both, in a sense that problem is split into smaller areas and distributed among many estimators (and hence, a distributed algorithm) and all the lower level estimators run in parallel. Hierarchical state estimation procedures were studied by Van custem [3] mainly for computation speed gain. This technique will be reviewed later in detail in this chapter.

Distributed processing has also been exploited, primarily for enhancing the speed of state estimation. Y. Lin and H. Lin [14,15] proposed a scheme for distributed state estimation. The power system under consideration is partitioned into many areas, and areas are governed by the local control centers. Their propagation method is based on a tier structure. The tier structure for the areas of power system is formed by first designating the area containing the swing bus to be Area 1 and the areas are labeled with

tier numbers. The area with the swing bus starts the state estimation procedure. At the completion of the state estimation in Area 1, the local computer in Area 1 sends the values of the states of the external reference boundary buses to adjacent areas in next tier. Once the computer system in Area *i* of Tier *j* receives the values of the state vector of all the external reference boundary buses from adjacent areas in Tier (*j-1*), the computer performs the state estimation for its area. Since each local computer system in the network can perform its own computations independently, the processing speed of the distributed state estimator carried out by computer network should be fast.

Carvalho and Barbosa [16,17] proposed distributed processing for state estimation using conventional algorithms like standard WLS and standard decoupled minimum distance estimation (MDE). The power network is decomposed into areas connected through boundary buses which belong simultaneously to both adjacent areas. Each area shares the state of the boundary buses at the end of iteration.

The method of Baldick and Ebrahimian [18] involves the application of the auxiliary problem principle to develop a distributed state estimator. The measurements of each area are telemetered to the computer of that area only, and only the computed border variables are exchanged between the adjacent areas. The neighboring areas interchange the border variables at each iteration and calculate the updated variables for the next iteration. With each iteration, a central computer will be informed of the status of each area for convergence. The communication transfer requirement in this method is much less than that of integrated state estimation.

Review of hierarchical state estimation

As discussed earlier the hierarchical state estimation technique was first introduced shortly after state estimation was applied to power systems. In hierarchical state estimation, the power system is split into smaller areas; the constraints for the size of smaller areas may differ. After splitting the power systems into smaller areas, state estimation for each smaller area is done using separate estimators. These estimators form one hierarchy. Now the outputs from these estimators are sent to the next higher level for consolidating outputs. This method of consolidation from the lower hierarchy is sometimes referred to as coordination. Always in hierarchical state estimation the lower levels are general state estimations and the higher levels are coordination levels, as only the lower levels alone deal with raw measurements. An estimator with one lower level and one upper level is called two-level state estimation.

Some of the earliest work on hierarchical state estimation was done by Van custem, Ribbens-Pavella et al in [1,2,3]. The reference [3] presents a critical survey of various hierarchical methods of state estimation. It compares one of the methods similar to the lower level and coordination level discussed earlier with a method using a split local estimation and local state estimate correction. In this reference, methods of decomposition of power systems have also been examined. The methods include a decomposition procedure, a direct method for decomposition and a relaxation method.

In reference [19,20] a dynamic and hierarchical state estimation techniques has been proposed. Here in this method an extended Kalman filter has been used for filtering the errors instead of simple WLS. State prediction has also been incorporated using

dynamic load prediction schemes. Both the lower and coordination levels of the state estimation have been implemented with dynamic filters.

M. S. Kurzyn [20] proposes a hierarchical static state estimation procedure in whihc the coordination method differs from the conventional two-level estimator. The coordination consists of three steps: the angle difference across the tie lines are calculated, and then the coordination angle between the reference area and neighboring area is calculated, and finally, the voltage angles of all the buses are recalculated from the obtained coordination angles. This method is achieved iteratively to obtain the coordination of other areas, even for those not directly connected to the reference. The method claims better performance than conventional WLS.

K.L.Lo, M.M. Salem et al [21] compare various techniques of hierarchical state estimation including the two-level state estimation technique. In large power systems it is enough to have one lowest state and one coordination level. The reference consists of two parts: one dealing with the logical analysis of the algorithms in consideration, and other dealing with computation experience obtained during the testing of these algorithms. Comparison of the algorithms led to the identification of various advantages and disadvantages of one algorithm over the other.

I.O. Habiballah [22] examines the effect of the application of two level state estimation on the accuracy of estimates, observability, etc. In this reference various scenarios were considered such as the absence or presence of boundary injection measurements, the absence or presence of tie line power flows, and the use of modifiable pseudo measurements. The method claims to be applicable in any partition strategy of a power system. The method was tested only on DC state estimation.

A. Bahgat, M.M.F Sakr et al [23] explore a novel possibility of two-level state estimation using non linear transformation. In this method the nonlinear relation estimation procedure is transformed in to a linear estimation problem. The transformation is applied in both the levels of the state estimation.

A.K. Sinha and J.K. Mandal [24] explored the possibility of using artificial neural networks (ANNs) to predict the load variation, and subsequently to predict the system state, in a dynamic hierarchical estimation procedure. The method claims to be fast and in this method the extended Kalman filter has been used in both the lower and upper levels of the estimation procedure. The method claims that the ANN technique accurately models load hence a better state prediction can be obtained. Furthermore, it is claimed that the EKF takes less time to correct the predicted state.

J.K. Marsh and M. Azzam [25] present a general framework for the design of two-level power systems state estimation. The main concern is the decomposition of the power system into smaller areas, the very first step of implementation of two-level state estimation. The method used here is called the α-decomposition technique. The main criterion for partition is that each individual sub-system should be individually α-observable.

In summary, it can be seen from the following review that in the past, hierarchical state estimation was viewed as a tool to improve the computation speed of the state estimation alone, due to the primitive performance capabilities of those computers.

Mathematical modeling of hierarchical state estimation

Traditional hierarchical state estimation, wherein the lowest level and one coordination level of state estimation is considered, will be discussed here. First, let us consider the lower levels. As discussed earlier a large power system will be split into smaller areas and in the lower levels of estimation, the state estimators estimate the state of the smaller subsystems.

We also need to consider some important criteria while partitioning the power system into smaller sub-systems. The first and foremost criterion is that the partitioning should be such that each individual should be independently observable. Moreover it will be better that there is enough redundancy of measurement in the sub systems. It is also a good practice to approximately match the generation and load in a particular area so that they are comparable. This will make sure that there is at least one generator bus in the sub-system. This is very important, because one of the buses is assumed to be a reference bus in the lower estimation level. Some of the references discussed in the previous section discuss specific techniques for partitioning areas.

Hence we can summarize the pre-estimation steps in hierarchical state estimation as follows.

1. Partition a large power system into smaller sub-systems conforming to certain preset decomposition criteria.
2. Identify the reference bus in each of the sub-systems.
3. Identify the tie-lines and boundary buses of the entire system.

Now since some of the basic information about the overall estimation has been established, the lower level estimation can be done. The lower level state estimation is a

regular integrated state estimation technique generally done using weighted least squares, but with respect to a local reference bus. The local reference bus is essentially a part of the sub-system. Hence the state vector of Area *m* can be defined as

$$x_m = [\delta_{im}, \delta_{jm}, \delta_{km}..., V_{im}, V_{jm}, V_{km}, V_{lm}, ...]_I$$

where x_m is the state vector of Area m, δ_{im} is the voltage angle of Bus i in Area m and V_{im} is the voltage magnitude of Bus i in Area m.

The measurements are related to the state vectors as in any general state estimation. This process of local state estimation can be carried out for all the sub-systems independently and asynchronously, and hence all the local state estimates are obtained. Next, the local state estimate outputs are sent to the coordinator. The coordination stage is the stage in which the individual state estimates are consolidated. The individual state estimates need to be consolidated because the local state estimates are with respect to the local reference, and before we do any further analysis, the state estimates of the local areas must be brought to one common reference. This common reference is generally referred to as the global reference for that hierarchy. If instead of a two-level state estimate there are multiple levels of estimation then at each level the global reference becomes the local reference for the next higher hierarchy.

The coordination process generally involves the estimation of the coordination vector. The coordination vector is a vector consisting of the slack bus angles of all the areas in a particular hierarchy. The angles for the slack bus of each local sub-system must be estimated with respect to one common global reference. As mentioned earlier, these

angles form the coordination vector. The length of the coordination vector is equal to one less than the total number of areas. Again the coordination process follows a simple weighted least square method. The coordination process is nothing but state estimation process on the tie-line power systems. The measurement for the coordination is tie-line power flows. The quantity being estimated is the coordination vector. The coordination process takes little time and few iterations because the number of measurements and the number of state vectors to be estimated are much less than that of the entire state. The local state estimates are required for the coordination process, because the coordination angles of each area add to the boundary bus angle of the areas and these angles are used to calculate the mismatch in power flow.

Mathematically the coordination process can be defined as follows. The coordination vector can be defined as

$$U = \{U_1, U_2, U_3\}^T$$

where U_i is the coordination angle of the ith area.

Now the correlation between the coordination vector and the tie-line power flow can be viewed from the following equation:

$$P_{ij} = |V_i||V_j||Y_{ij}|\cos(\theta_{ij} - \delta_i - U_A + \delta_j + U_B)$$

where, i is the ith bus in Area A, j is the jth bus in Area B, V_i and V_j are the voltage magnitude of bus i and j respectively, Y_{ij} is the magnitude of admittance of the tie-line, θij

is the angle of the admittance of the tie-line, δ_i and δ_j are the voltage angles obtained from the local state estimate of the i^{th} and j^{th} bus respecively, and U_A and U_B are the coordination angle to be estimated from each area.

From the tie-line measurements the coordination angles for each area can be estimated using WLS method

$$Z_T = h\,(U, X_b) + e$$

where, Z_T is the set of all tie-line measurements, U is the coordination vector to be estimated, X_b is the set of all boundary bus measurements used as parameter for the estimation process, and e is the measurement error

$$\Delta U = (H_U{}^T R_U{}^{-1} H_U)^{-1} H_U{}^T R_U{}^{-1}\,(Z_T - h(U, X_b))$$

where ΔU is change in coordination vector for each iteration, H_u is the Jacobian matrix for the coordination vector and R_u is the measurement error covariance matrix

From this equation, the change in coordination vector is allowed to converge and then the coordination vector can be obtained. In some methods the coordination angles are sent back to the respective areas and the state of the local area is recalculated. A variant of this is that the coordinator itself consolidates the state output with the estimated coordination angles. This method of approach is depicted in Figure 1.1.

Outline

The reminder of the thesis is organized as follows. In Chapter II the issues associated with the application of hierarchical state estimation to wide area networks are studied. The application of synchronized phasor measurements in state estimation and in hierarchical state estimation are considered in Chapter III. Chapter IV deals with the exact simulation method of the state estimators and the results of simulation. Finally, Chapter V presents conclusions and discusses possible future work.

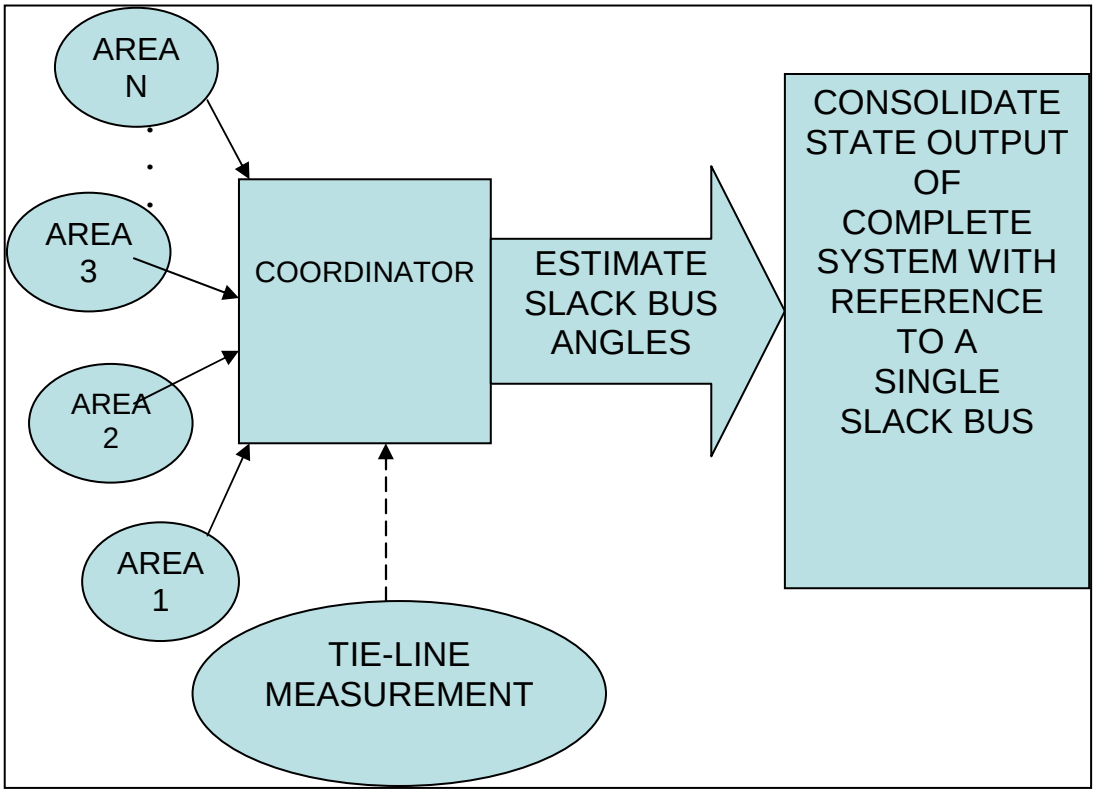

Figure 1.1: A schematic of coordination of two level state estimation

CHAPTER II

STATE ESTIMATION FOR WIDE AREA POWER SYSTEMS

Introduction

This chapter deals with a detailed review of literature on monitoring of wide area power systems, the issues associated with the application of state estimators in wide areas, and a robust modified algorithm that can be applied to wide areas.

Review on application of state estimation in large scale systems

In large scale power systems comprising multiple areas, the measurement sets and the state vector becomes extremely large. For this reason, it becomes highly uneconomical to carry out centralized state estimation. First, there is a need to transfer huge amounts of data from the measurement units to the state estimator which increases the cost requirement for the communication link. Second, the computational burden involved for state estimation is also high. Hence many decentralized schemes have been proposed. The main advantage of using a decentralized scheme for a multi-area scenario is that the existing state estimators in each area can be used to obtain the global state estimate for a multi-area power system. This has been proposed by A. Abur and L. Zhao [20] wherein all the areas are assumed to have a state estimator and the coordination is

done using the obtained state outputs from these areas. The method also uses synchronized phasor measurements in coordination stage.

Phadke, Liu et al [27] propose a different approach for wide area information sharing. The approach targets wide area state estimation. It also proposes to use the internet for communication between the local area and the coordinator.

In all of the above mentioned references some of the issues in the application of state estimation to wide area have not been considered. Chun-Lien Su and Chan-Nan Lu [28] examine the possibility of delayed arrival of measurements and propose a dynamic filter for state estimation. Extending this further to wide area state estimation, there is a possibility that the state outputs for the local state estimator and/or measurements arrive delayed to the coordinator.

Gonazález-Pérez and Wollenberg [29] analyze the possibility of massive measurement loss in power systems using a DC state estimator. The DC state estimator is a linear state estimator which is an approximate model. Extending this further to the wide area scenario, a massive loss of measurement in one of the areas will make the state estimation impossible in that area for that cycle of measurement sweep. This leads to a complete absence of state output from that area to the coordinator.

Thorpe, Phadke et al [30] examine the issues associated with the development of a wide area analysis. The issues identified were: overlap of areas, time synchronization of the coordinator, and missing state outputs from one or more of the areas. The reference also states possible solutions of which some are tested as a part of this thesis. From the above review we can clearly infer the important issues concerning the application of state estimator for a wide area network. Also it can be inferred that a hierarchical scheme is

preferred over other techniques due to the use of existing state estimators. Such an approach cuts down the cost of setup of a wide area estimator, as there is obvious reduction in the cost of setup of the communication channel and the configuration requirements of the coordinator.

The issues mentioned above can be listed as follows: (1) overlap of area, (2) asynchronism between the local state estimators, (3) delay in the arrival of state outputs to the coordinator, and (4) absence of state output from an area. The last two of these issues will be studied in detail in the following sections. Now the overlap of area can be avoided by proper indexing of the buses in the areas. The requirements are that no two areas overlap and the all the areas are connected by tie-lines. This requires a slight modification of the local state estimators.

A situation may arise in which the state estimators send their outputs at different time intervals to the coordinator. This will not be uncommon as the size of each area is different and the state estimators are very often different from one another. Thus it takes different times for the state estimators of the local area to converge. There can be additional inconsistencies due to the communication link. The delay due to the communication link depends on the type of communication and the distance to be traveled. The asynchronism will not pose a big problem if the convergence time of all the state estimators in the lower levels are comparable, in which case the coordinator can be set to stop the reception of the state outputs from the lower levels after a particular time. This time should be greater than the longest time the local state estimators need to achieve convergence.

When the convergence time of some of the areas are very large compared to the of the other state estimators, then the coordination time interval can be set to a time smaller than the convergence time of these slow areas, and the state outputs from these areas can be considered to be delayed data for that particular cycle of coordination.

Delayed arrival and loss of state output

As mentioned earlier there is a good possibility that the state output does not arrive to the central coordinator on time to do the coordination. In general, the central coordinator can be set to stop the reception of state outputs from areas at specified time intervals. If the state output is received from a particular area after this time, then the data is considered to be delayed data, and these values should be considered for the next state without interrupting the already running coordination cycle.

There are many causes for the delayed arrival of state outputs, including the following: the state estimator of the local area does not converge, the area becomes unobservable, there is a software failure of the state estimator, and there is a failure of the communication link between the area and the state estimator.

These causes can also lead to a complete absence of state outputs from an area, in which case it is said to be the loss of an area. It is possible to obtain status information through the communication link from the state estimator of an area. This status information can provide updated information about convergence, observability and other issues, from which the coordinator can identify whether the state output will be delayed or there is a loss of area.

Thus, we need to consider these two issues while implementing the coordinator for the hierarchical state estimation in a wide-area system. If the outputs are delayed from a particular area, then it is a valid assumption that the states of the area do not change considerably for a short period of time. It follows that the coordination of the areas can be done using the state of that particular area obtained from the previously sent state output. Also a reduced weight for the state output of those areas can be used. The present value of state outputs for the delayed areas either can be neglected or used in the next coordination stage. For a situation when the state outputs are delayed for more than one coordinator steps, the weights can be decreased for consecutive coordination cycles.

In a situation for which there is a complete loss of area, the coordination can be done by considering the remaining areas, and the boundary buses and tielines connected to those areas will be neglected in coordination. This will lead to a decrease in accuracy and is demonstrated in the simulations shown in the following chapters.

Modified algorithm

Hierarchical state estimation can be modified to accommodate the above state issues. The following section shows the algorithm for hierarchical state estimation and the modified algorithm which takes into consideration the issues stated above. The first algorithm is a regular static hierarchical state estimator.

The following algorithm elucidates the coordination process alone, as the local state estimation is a regular static state estimation.

1. The coordinator is ready for the reception of the state outputs from all the member area.

2. Tie-line measurements and boundary bus measurements are received.
3. From the available state estimator outputs, boundary bus states are used as parameters and the coordination vector is estimated using the measurements.
4. Using the coordination vector obtained from Step 3, the outputs of all the areas are consolidated and this output is sent to other applications and the corresponding areas.

The following algorithm is the modified algorithm of the two-level state estimator in which the issues of delay in the arrival of state output and loss of areas has been incorporated. The algorithm is as follows.

1. The coordinator waits for the reception of state output from any one of the areas.
2. A timer is started and the coordinator receives state outputs from the areas until a preset time value.
3. If some state outputs of some areas are not received, then the weights are decreased by a set amount from the previous value of weights of that particular area and the state output from the previous step for that area is used.
4. If a weight falls below a particular value, the area is removed from the coordination list and an alarm is raised untill the state output is received from that area.
5. Using the boundary measurements and the state outputs, the coordination vector for the available number of areas is estimated.

6. The state outputs of the available areas are consolidated using the coordination vector, and the consolidated output is sent to other applications and the member areas.

When the weight given to the state output of particular area falls below a particular value, it is indicative of the fact that the state outputs from an area were not available for some number of coordination steps, in which case it can be said that the area is lost. In this situation the coordination is done for the remaining areas. There is potentially a significant decrease in accuracy because the state output obtained from the previous step is used. This may not be suitable if rapid coordination steps are required and states of area fluctuate rapidly. This can be overcome by proper use of phasor measurement units in the areas.

Both algorithms have been tested on a IEEE 118 system. The simulation techniques and results are discussed in detail in Chapter IV. It is observed that the accuracy decreases as the state output arrivals are delayed. The following diagram shows a flow chart of how the modified algorithm works. The coordination stage alone has been explained and the same process is repeated for every coordination cycle. During the start of every coordination cycle the weighting matrix is reinitialized and the values are retained if the state outputs are not received; otherwise, they are brought back to their original values. Again it can been seen that there are some issues regarding the accuracy. The method consolidates the state outputs for temporary delays of state output arrivals.

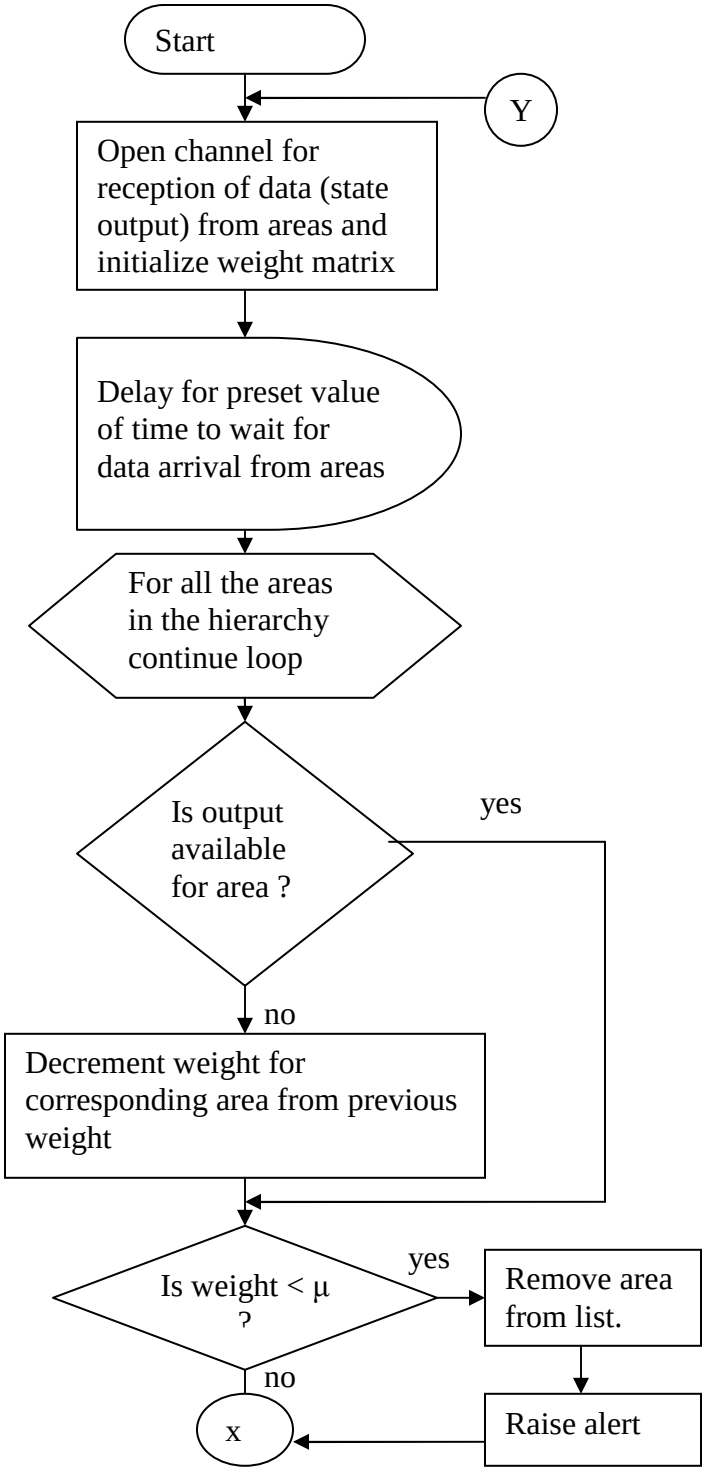

Figure 2.1: Flowchart showing the modified algorithm of hierarchical state estimation

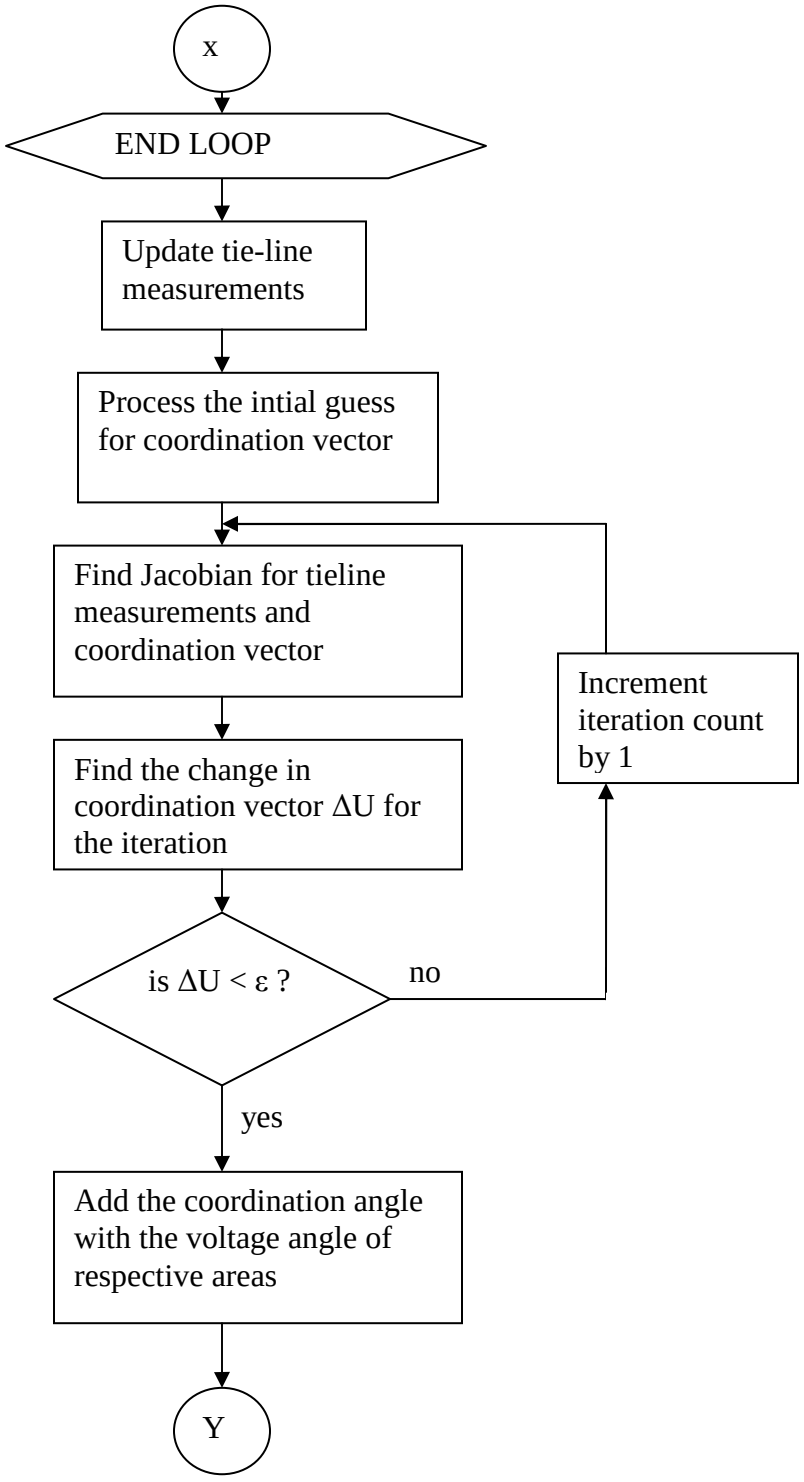

Figure 2.1: Flowchart showing the modified algorithm of hierarchical state estimation (continued)

CHAPTER III

APPLICATION OF SYNCHRONIZED PHASOR MEASUREMENTS IN WIDE AREA SYSTEMS

Introduction

This chapter deals with the study and application of phasor measurements in wide area power system state estimation. The main concern is the use of phasor measurement units (PMUs) to increase the accuracy and efficiency of lower and higher level state estimation.

Literature review on phasor measurement units in state estimation

Phasor measurement units are the direct descendent of the symmetrical component distance relay. Work on phasor measurement units started as early as 1980s in Viginia Tech. Later, PMUs became a commercial product. PMUs have found their way in many power systems operations after its commercialization. One of the important functions in which PMU have been deployed is state estimation. A. G. Phadke [31] gives an historical review on the development of synchronized phasor measurements.

One of the main advantages of phasor measurement units are that all the phasor measurements are time synchronized with a GPS clock. This means that the measurements obtained from the PMUs are obtained at the same time without any time

skewness. This is a very import advantage in its application as a monitoring device in wide area power systems.

Phasor measurement units are predicted to become a very vital part of power systems state estimation. As such, the measurements from PMUs are proven to increase the observability of power systems by strategic placement of a minimal number of phasor measurements. This has been discussed by Xu and Abur in [32] and many others. In this work the cost of installation of PMUs is taken as the objective function to be minimized with the constraint being the observability of the power systems. The observability can be defined using a matrix containing ones and zeros. If there is a PMU present on a bus or on an adjacent bus then it is given a value of one otherwise a zero is given. If there are other measurements available then these can be incorporated in the matrix which ultimately reduces the number of PMUs by reducing the cost.

Another approach for PMU placement using spanning trees of power systems graphs has been proposed by Nuqui and Phadke [33]. Here, a simulated annealing procedure has been used to add constraints on the PMU placement algorithm. One of the constraints is the reduced communication channel requirements.

It has also been studied that PMU measurements drastically improve the performance of bad data processing of the state estimator. Bad data processing is nothing but weeding out of some inconsistent data which makes the measurement residual of the state estimator very large, or sometimes makes the state estimator unable to converge to a particular estimate. J. Chen and A. Abur [34] examined the performance of PMU units with respect to bad data processing. The basic valid assumption here is that the PMU

measurements have very high reliability and accuracy. Again a strategic placement of minimal number of PMU units will improves the accuracy.

In the past, the measurement skewness has been a great problem as the measurement takes longer time to reach the control center state estimator if the Remote Terminal Unit collecting the measurements is far away. The result is that the state estimates are for the older measurements and hence inaccurate. This has been overcome in PMUs by exploiting the time synchronization of the snapshots taken by the units with the GPS clock. But again this data has to be transferred to the control center through a communication link. This delay due to communication link has been dicussed by C. Valenti, A. Feliachi et al [35].

Before the way in which PMU can be used in wide areas is examined, the basic working of PMU is discussed. PMUs record the instantaneous voltage. This instantaneous value can be referenced with any wave which gives the magnitude and phase angle with respect to the reference wave. The synchronizing clock pulse actually samples the recorded pulse and then sends this recorded pulse to the control center through a communication link.

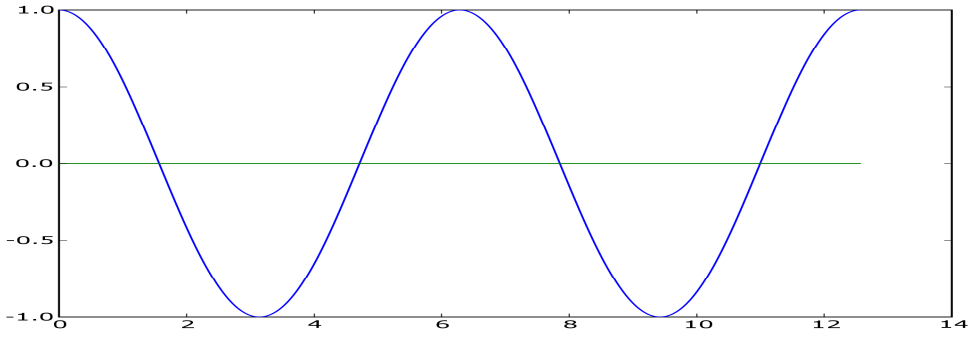

Figure 3.1 : Reference wave

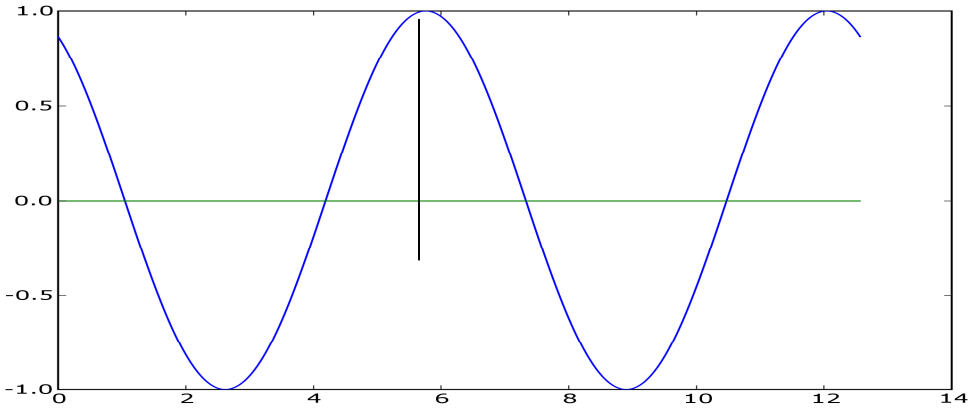

Figure 3.2 : Voltage at Bus m

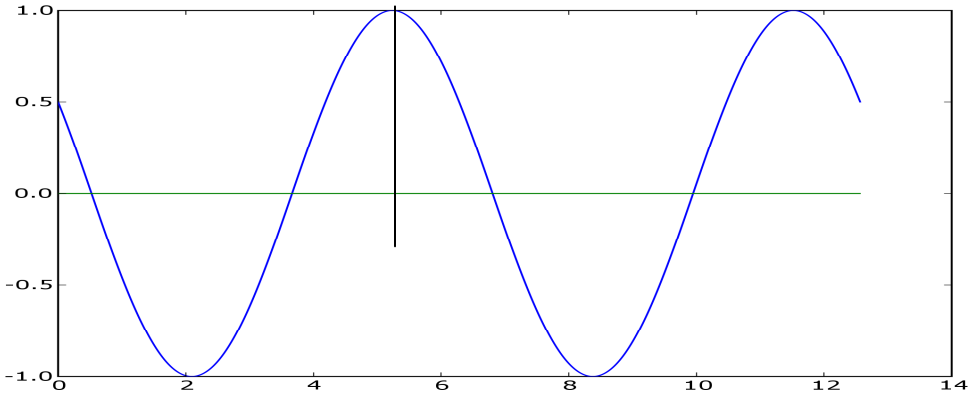

Figure 3.3: Voltage at Bus n

As we can see from the above voltage diagrams the peak of the two buses is displaced from the peak of the reference wave. Hence this is indicative of the phase angle of voltage of with respect to the reference wave shown in Figure 4.1. This is how the comparison in PMU units is made and the phase angle is calculated to a very high degree of accuracy. The measurement is then transferred to a control center and is then used for

state estimation. The priniciples of working with synchronized phasor measurement units has been explained by Moxley [36].

When using PMUs for wide area measurement phasor data concentrators are employed in different areas that collect the phasor data from their area and send the data through communication links to the central control center. Galvan [37] presents the role of PMUs in the eastern interconnection and the communication infrastructure required for achieving this. Here the role of PMUs has been extended for better data accumulation and visibility. Also, the possibility of enhancement of state estimation due to PMUs is discussed, keeping in mind the present standard of SCADA data available for state estimation.

Rice and Heydt [38] propose a method to quantify the accuracy improvement obtained by the introduction of PMUs in power systems. Using this technique, optimal PMU placement has also been studied. The method has been tested on a IEEE 14 bus system. The method basically uses the residual vector to quantify the accuracy, and the addition of PMUs reflects on the reduction of residuals. Normalized error, norm of residual vector and RMS of residuals have been used to quantify the accuracy.

Wu and Giri [39] explain the important constraints in the application of PMUs in a state estimator. The important observation made by the authors after application of PMUs on various systems is that the location of reference PMUs does not have any impact on the SE performance. The authors also address the key issues of calibration of PMUs, synchronization of PMU data and tuning of PMU data weights inside the state estimator. However all of these references examine the effect of PMUs in integrated state estimation.

Abur [40] examines the operation of distributed state estimation in mega grids. There is an emphasis on the use of PMU in the distributed state estimation in this paper, but it mainly focuses on the data sharing between the areas and the coordination algorithm. A method of using the PMU measurements in the coordination has been suggested.

Synchornized phasor measurements in lower level of hierarchical state estimation for wide area

Recently there has been an increase in the use of PMU units in many power systems in which they have been incorporated in the state estimator. To study how the PMU influences the performance of the state estimator, PMUs are employed in the local state estimator of the previously state algorithm given in Chapter III. PMU units give the exact voltage angle with respect to a specified reference. Assuming this reference to be the slack bus angle of that local area, the phase angles obtained from phasor measurements are incorporated as measurements. This assumption is valid, as only a static state estimate is required and hence the fluctuation in frequencies and distortions can be neglected as they generally die out over a moderate period of time.

The relationship between the phase measurements and the state variables is linear, as we have the state variable itself as the measurement. The phase angle of the bus having the synchronized phasor measurements is taken out of the state vector, and the value obtained from the phasor measurements can be directly used as a parameter in the estimation process. Alternatively, it can be treated as measurement with high accuracy.

It follows that the Jacobian for the measurement will contain a one at the respective bus number and zeros for other state variables. In general, the phasor

measurement units are always used as a supplement to the regular measurements, due to the sparse occurrence of PMUs in power systems. The injection measurements are generally given low weights compared to power flow measurements. During the implementation of phasor measurements in our simulations, some of the power injections were removed as they led to ill conditioned case of the diagonal measurement error covariance matrix.

Synchornized phasor measurements in coordination level of hierarchical state estimation for wide area systems

In the previous chapter some of the problems in the application of hierarchical state estimation in wide area systems have been discussed. Now some of these issues can be overcome using synchronized phasor measurements. When phasor measurements are available in local areas, this measurement data, when made available to the coordinator, can be used to effectively increase the accuracy of the coordinator.

If a good number of phasor measurements are available in the lower level, then a linear coordinator can be designed which can be used to obtain the coordination angles. This offers the additional advantage of very fast convergence of the coordinator, and the tie-line measurements are not required. Hence if there is a loss of tie-line measurements and the coordinator itself becomes unobservable this method is very effective. Again this method relies very heavily on high reliability of the phasor measurements. Also the phasor measurements can be used for bad-measurement checks in the coordination level if tie-line measurements are used. If tie-line measurements are used, then the coordinator ceases to be a linear estimator and the phasor measurements can be used by giving high weights along with tie-line measurements.

Mathematical formulation of the linear estimator using pmu measurements for coordination

The following section deals with the mathematical formulation of the coordinator wherein phasor measurements alone are used for coordination. Now if N is defined as total number of areas, then as mentioned earlier the coordination vector will be of the form

$$U = \{U_1, U_2,..., U_N\}^T$$

If the total number of phasor measurements is m in the wide area, and if each phasor measurement is numbered 1 through m then we have the measurement set from the PMUs as follows:

$$Z = \{Z_1, Z_2,\ldots, Z_m\}^T$$

The state will also be available from the areas and hence the buses having the phasor measurements can be taken and arranged in the same order as above will give a parametric bus state vector as

$$X_b = \{X_{b1}, X_{b2},\ldots, X_{bm}\}^T$$

The phasor measurements can be set to be received with respect to a global reference, and the state outputs received from the areas are with respect to local reference. Hence the difference between the two will give the coordination angle for the corresponding areas. If we have more than one number of PMU measurements in an area then we have a over-determined set. Using the Least Squares technique, we can find the coordination vector from the following measurement equation:

$$Z_m = [Z_P - X_P] = H.U + e$$

$$U = (H^T R^{-1} H)^{-1} H^T R^{-1} Z_m$$

where, Z_P is the phasor measurements of the buses in local areas, and X_P is the state output of the buses having the phasor measurements alone, R is the measurement error covariance matrix, e is the measurement error, and U is the coordination vector.

The H matrix shown above contains ones and zeros according to the areas a particular bus belongs to. If, for example, Z_1 and X_1 belongs to area three then the 3rd column corresponding to this measurement becomes 1 and everything else is zero in that row. The linear estimator has been tested on a IEEE 118 bus system and the results and method of simulation are shown in next chapter.

Though all the phasor measurements have high accuracy, the state outputs obtained from the areas may contain error and these error percentages can be determined from the information available from each area. If there is good redundancy then the error due to the use of previous states of the system can be eliminated. If there are high discrepancies then those bus states can be removed as bad data.

An important thing to note here is that the coordinator needs to send the reference wave to the PMUs to recalculate the phase angle of the voltage. This is readily possible with the newer PMUs. By doing this, the state of the bus is obtained with respect to the global reference. In addition, the time delay to reach the coordinator is not much because the newer PMUs use Ethernet, fiber optics and other sophisticated communication techniques in which the travel time for the data is negligible.

CHAPTER IV

SIMULATION OF ALGORITHMS AND RESULTS

Introduction

In this chapter the complete simulation of the algorithm described in Chapter II and Chapter III is dealt with, followed by the results of the simulation. The test system taken into consideration is the IEEE 118 bus system [41].

Partition of ieee 118 bus system

In general, to test a hierarchical or a distributed model, a test case is taken and the system is split into smaller systems. The main criterion for the partition of the areas is that the *load in the smaller partition approximately matches the generation in that area.* Moreover, it is also required that the tie-line power flows are present. In other words, all the areas should be electrically connected to one another.

It is best if the sizes of the areas are comparable so that the state estimators of the areas converge at almost the same time. For this reason, the IEEE 118 bus system has been split into smaller areas as shown in Table 4.1. The area numbers have been listed in the table along with the bus number in the original IEEE 118 bus system. The reference bus for each area was selected as the biggest generator bus in that area. The reference bus number in the areas and in the overall system have been specified.

Table 4.1: Partition Information Of The IEEE 118 Bus System

Area	Bus Number in IEEE118	Reference Bus in the areas	Reference Bus No in 118 Bus system
1	1-17,30,113,117	10	10
2	18-29,31-37,114,115	9	26
3	38-79,116,118	32	69
4	80-112	10	89

When partitioning the power system it should be kept in mind that the sub systems obtained by partitioning should be observable; i.e, the state estimator should converge independently, without any measurements being transferred from the neighboring areas. Indeed, in the above design this constraint was kept in mind and it will be later seen that the areas converge separately and therefore can be solved independently. It is important that the areas are made independent so that the hierarchical state estimator is basically a parallel, dispersed algorithm and the estimation problem for each area should be run in parallel and asynchronously.

Apart from this, for the next higher level of state estimation, if a classical measurement model is used in which there are no PMU measurements, the tie-line measurements are needed for doing the coordination. To enable this, the tie-lines between the areas and the boundary buses of each area need to be identified before implementing the algorithm. In the real world this is provided by the *coordination topology processor* which gives all information about the topology of the tie-lines. This topology information is diagrammatically represented using Figure 4.1. Each and every bubble in the figure

represents an area, and the bus numbers correspond to the bus number with respect to the whole IEEE 118 bus system and not with respect to the local areas. The lines represent the tie-lines connecting two boundary buses.

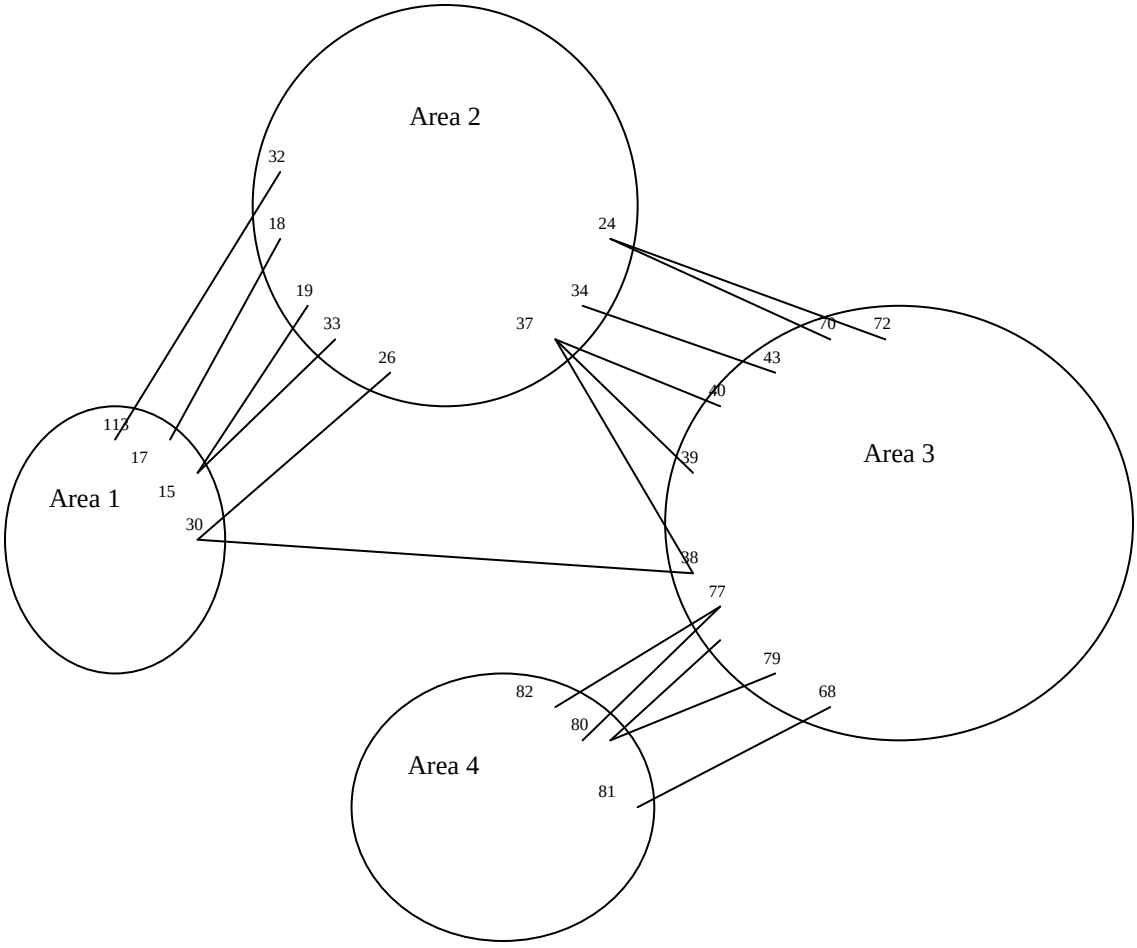

Figure 4.1: A schematic of Partition of IEEE 118 test case indicating the boundary buses and tie-lines.

To enable this partition a simple code was written to calculate the total load and total generation for a particular set of buses. The number of areas are given as input and an approximate area matching is done using the program, after which the areas are fine tuned to get an acceptable range of load-generation mismatch. Then the numbers are again checked with the topology of the system so that the buses of adjacent areas are electrically near to one another.

Use of an object oriented programming language – python

As mentioned earlier, hierarchical state estimation is a parallel, distributed algorithm, and thus, the local estimates have to run in parallel and asynchronously. This can be simulated by an object-oriented programming (OOP) concept known as multi-threading. MATLAB or C programs which are traditionally used do not support multi-threading. Python [42] offers good numerical libraries which can handle the matrix manipulation needed for the state estimation algorithm.

The Python programming language has also been used as a quick scripting tool in power systems simulator software called PSS/E developed by Siemens. Libraries for Numerical capabilities are compiled in Fortran and used in Python. Another advantage of Python is that can be interwoven with C and Fortran code.

Result of integrated state estimator

To see the difference in performance between the centralized state estimation and the hierarchical approach, the IEEE 118 bus system has been run, and the state estimator output has been noted. The same integrated state output function has been used in the two

level estimators for the lower level estimates. It was clearly seen that the integrated state estimator took a longer time for four iteration of the system. The program took about 59 sec to converge.

Result of regular hierarchical state estimator

The first algorithm specified in Chapter II an unmodified hierarchical estimator in which the coordinator waits untill the output for all the areas are received is simulated here. The local state estimator for each area is designated a separate thread and the integrated state estimator function is called with the respective arguments pertaining to the sub system. The main thread which is representative of the central coordinator waits untill all the threads finish execution and return a completed flag. This flag is captured by the main thread along with the return values which are the state outputs and then the coordination is done. The results of the execution of this program are shown below.

Area 1 Converged in 5 Iterations 5.04287099838 Seconds

Area 2 Converged in 5 Iterations 4.74246406555 Seconds

Area 4 Converged in 5 Iterations 7.89450407028 Seconds

Area 3 Converged in 5 Iterations 10.5698409081 Seconds

Recived all data

coordinating...

12.21

Coordination Done in 3 Iterration

Coordination Vector is

[[35.71426647 30.01102355 30. 39.72426094]]

13.82

The program log file has been inserted here. It can be seen that the hierarchical state estimator took about 14 sec which is much less than what would be needed with an integrated state estimator. Hence it is worthwhile to use hierarchical state estimator. The percentage error in the state variables due to the addition of randomly varying error to the measurements having a Gaussian distribution is shown in Figure 4.2. The graph presents precent error as a function of the bus number.

Moreover, it is evident from the figure that the voltage magnitude error is much less than to the phase angle errors. This is due to the high weight given to voltage measurements compared to power flows and injection measurements.

Simulation of wide area scenario

In order to properly study the effect of the issues stated in Chapter III, the wide area scenario has also been simulated. The delay in the arrival of state output can be simulated by introducing artificial delay in the local state estimation thread. The delays occurring in the local state estimate are assumed to be random, making the setup more realistic. A preset timeout value for the main thread corresponding to the coordinator is used. This is equivalent to timeout of reception of state output in the modified algorithm explained in Chapter II. The random variable determining whether the state output is delayed or not, is modeled as a Bernoulli distribution. The reason behind this is explained as follows.

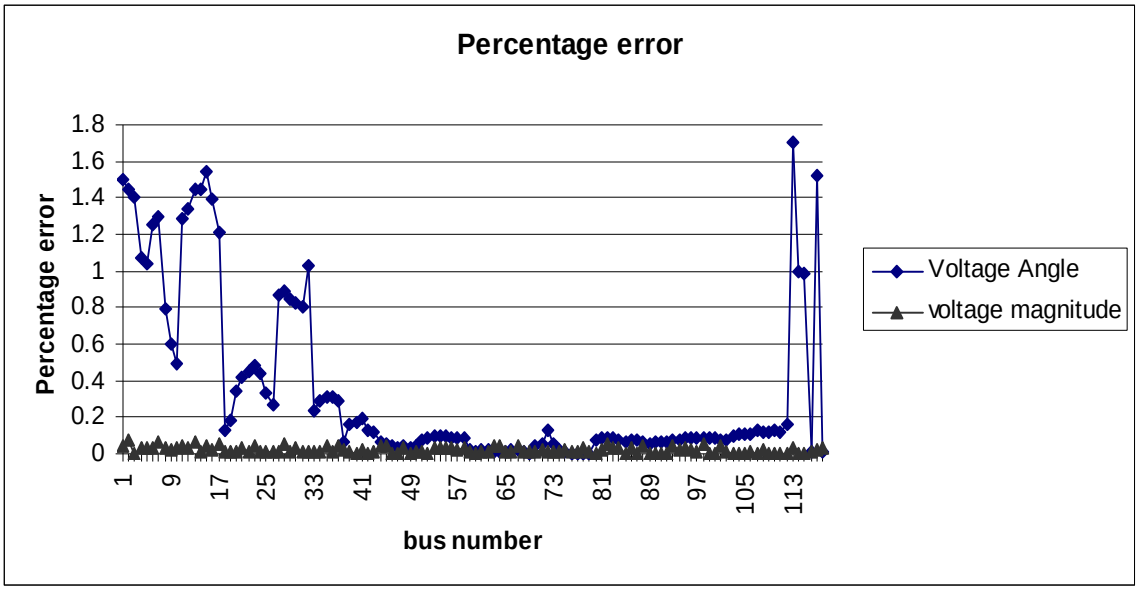

Figure 4.2: Percentage error vs State variable number for regular hierarchical state estimator.

The local state estimator can be delayed by any amount of time which varies randomly, but since the reception of state output stops at a predefined value of time, there can be only two possibilities. If the time taken by the local estimator is much less than this preset value, then the area's output is not delayed; otherwise, it is delayed. This can represented by either 1 or 0. Thus the probability that an area *i* is delayed is set as *p* then the probability that it is not delayed is *1-p*.

During the program each area is given a probability and this value is fixed. If, say, the value is 0.9, then the state output of that area is received, on average, 9 out of 10 times. A random number generator provides a uniformaly distributed floating point number between 0 and 1. To convert the uniform distribution to a Bernoulli distribution a

simple function was written. The function returns randomly 0 or 1. The occurrence of zeros or ones depends on the probability value passed as an argument. This value again is passed to the lower level state estimator which invokes a sleep function for a particular period of time, depending on the whether the output needs to be delayed or not. The value can be varied according to the expected reliability of the local state estimators.

The modified algorithm described in Chapter II uses values from the previous step. This has been incorporated by storing the state values of the previous step as a back up before doing the local state estimate for the sub-system.

If the state output does not arrive to the coordinator, with respect to this program the local state estimator thread times out, and then the values from the stored backup of that particular area is moved to the state output list. Moreover, weights of measurements pertaining to that area are reduced by a fixed amount. Then a check is made for the threshold value of these weights. If it falls below a particular value then there is a loss of area.

Results of the modified hierarchical state estimation algorithm

Now, the simulation has been run with an initial value of probability of delay as 0.95 for smaller areas and 0.9 for large areas (which is Area 3), because it is reasonable to assume that state estimation for larger areas can take more time. For a delay in area the weights of the measurement and state output were reduced by 5%. It has been observed that the accuracy is reduced for the situation in which the state output from one of the areas specifically Area 4, is delayed by one coordinator step. This is shown as a graph in Figure 4.3.

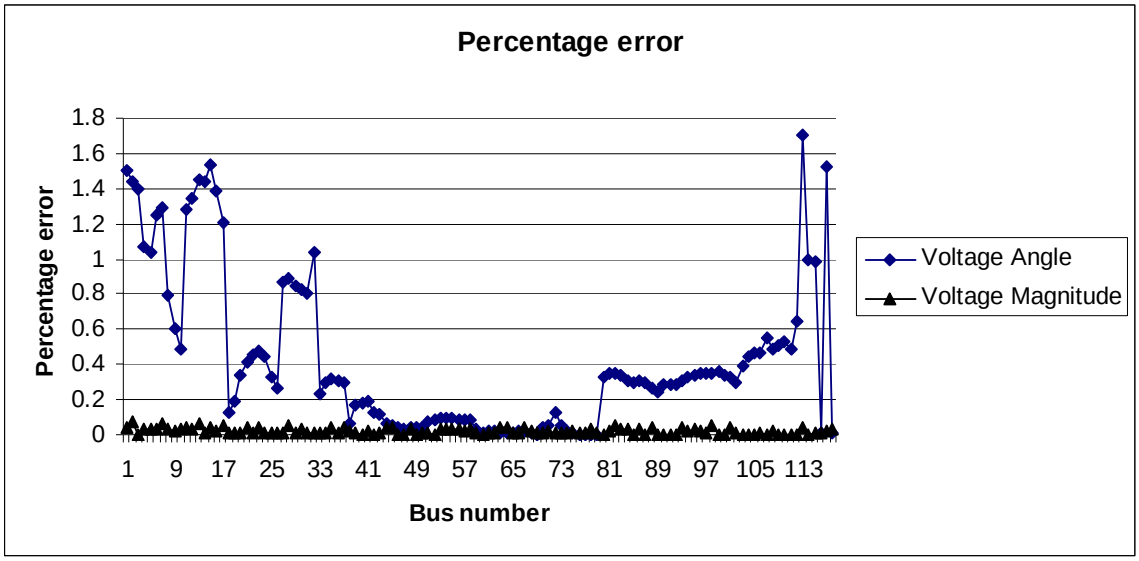

Figure 4.3: Percentage error vs state variable number for Area 4 which is delayed by one coordinator step

Now when the same area gets delayed by two coordinator steps then the accuracy deteriorates further. The weights were decreased by 15%. It can seen that the accuracy for other areas decrease as well. This is due to the fact that they are connected by tie-lines and the accuracy of coordination angle of other areas are dependent on the state outputs of this area.

If there is a loss of area then the coordination is done without taking the area into consideration according to the modified algorithm specified in Chapter III. The results for this is shown in figure 4.5. It can be noted that the state variable numbers are reduced as the Area 4 is neglected from coordination. Moreover, the accuracy has decreased very much due to the fact that the power flow in and out of the lost area is not being considered in coordination.

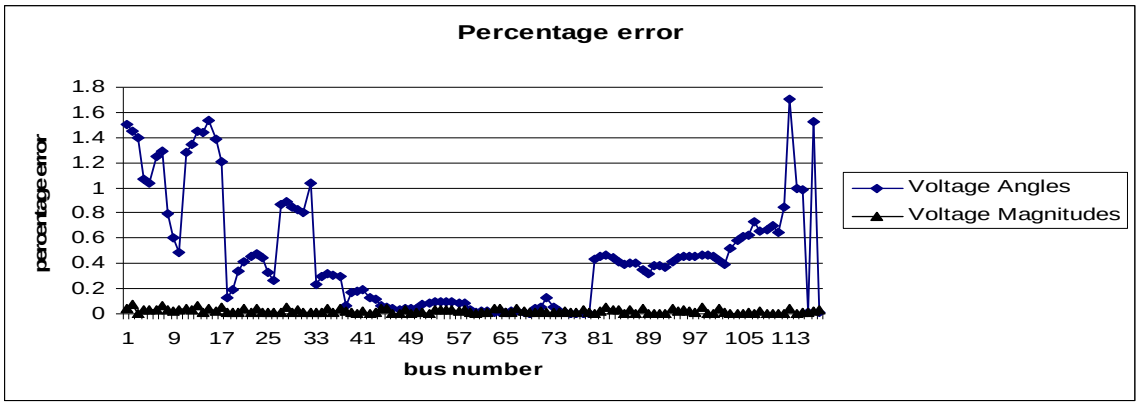

Figure 4.4: Percentage error vs state variable number for area 4 output delayed by two coordinator steps

Results of application of phasor measurements in the lowest level

The PMU measurements have been implemented and the Jacobian matrix was modified accordingly. This was incorporated in the lower level, and thus, the reference for the PMUs were the local references. PMU units were assumed in buses 1, 7, 18 - 19, 30 - 34, 68, 79 – 81, and 115 -118. Most of the buses belong to high voltage levels. Some of the vital buses have been used here. Most of the buses are also boundary buses. It is clearly seen that the accuracy and the execution speed has increased by the application of PMU measurements. The results are shown in Figure 4.6.

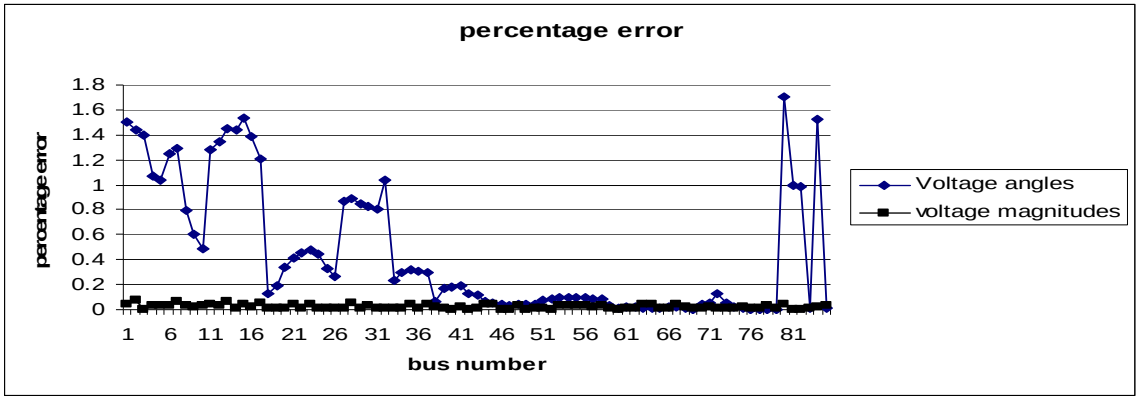

Figure 4.5: Percentage error vs state variable number for loss of area 4

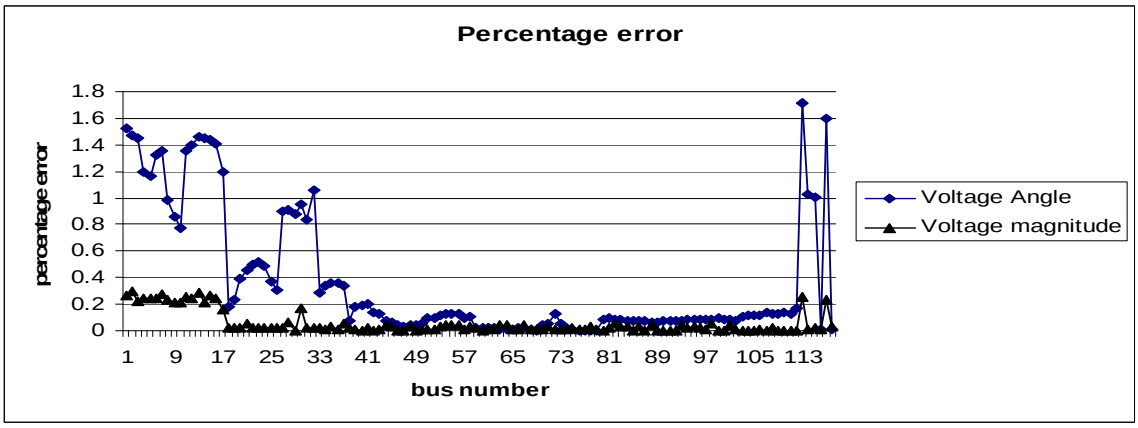

Figure 4.6: percentage error vs state variable number for incorporation PMU vs State variable number.

Also the number of iterations has decreased and the speed of convergence has increased due to the addition of PMU measurements. This can be seen from the following program log.

Area 1 Converged in 4 Iterations 2.39033699036 Seconds

Area 2 Converged in 4 Iterations 2.35015296936 Seconds

Area 4 Converged in 3 Iterations 3.8940808773 Seconds

Area 3 Converged in 4 Iterations 5.46229887009 Seconds

Recived all data

coordinating...

Coordination Done in 3 Iterration

Coordination Vector is

[[35.71770996 29.98987545 30. 39.7249489]]

Results for application of phasor measurements in coordination level of hierarchical state estimation

As mentioned in the earlier chapters, if good number of PMUs are available to the coordinator, then the coordination can be replaced by a linear estimator which can estimate the coordination angle for use with the buses which doesn't have PMU. The main advantage of this linear estimator is that *the unobservability in coordination due to the fact that the lack of tie-line measurements, can be avoided.* Here PMUs from the local estimators reference to the global slack bus is taken as one measurement, and the linear estimator is run to do coordination. The following results of the program were obtained. The accuracy margin still remain about the same, but the coordination time is drastically improved.

Area 2 Converged in 4 Iterations 2.25758385658 Seconds

Area 1 Converged in 4 Iterations 3.22476196289 Seconds

Area 4 Converged in 3 Iterations 3.7106180191 Seconds

Area 3 Converged in 4 Iterations 5.53035998344 Seconds

Recived all data

coordinating...

Coordination Vector is

[[35.87913036 29.96007795 30.00073399 39.74902474]]

The difference in the execution time of coordinator is 0.5 for the regular coordinator and 0.2 secs for the linear estimator. The margin becomes higher as the number of areas increase.

CHAPTER V

CONCLUSION

Discussion of results

The application of hierarchical state estimation for a wide area power system has been studied in this thesis. The issues associated with the implementation of the estimator to a multi-area network such as asynchronism, delay of arrival of state outputs and loss of state estimation output have been dealt with in the proposed algorithm. Accuracy of the delayed area has been sacrificed for the consolidation of the state output. The accuracy decreases by 0.5% for single step delay of an area in this particular case. It is evident from the results shown that the accuracy of the neighboring area has not decreased. Use of synchronized phasor measurements in hierarchical state estimation structure has been examined, and the use of phasor measurements in coordination level proves to be more advantageous. Results obtained by simulation show that the computational time is decreased by 40% for IEEE 118 bus system and the coordination is more reliable, assuming high reliability of phasor measurements. Problems of tie-line power system observability in the coordination stage is reduced with the use of highly reliable synchronized phasor measurements. Hence, synchronized phasor measurements increase the confidence of the overall estimate when used in all the levels of the hierarchical state estimation algorithm. Thus, it can be concluded that the above stated issues are very

critical issues affecting the proper operation of hierarchical state estimation in a wide area scenario. Additionally, the use of synchronized phasor measurements in all the levels of hierarchical state estimation improves the confidence and performance of the estimator, especially in a multi-area scenario.

Discussion of future work

Some of the critical problems addressed by this research can be vital for the improvement of wide area power system monitoring. A possible extension to this thesis is the introduction of a dynamic model for the coordination taking into consideration a state forecast for a particular area when state outputs are not available. The development of an efficient PMU placement algorithm taking into consideration the implementation of PMUs in lower levels and the upper levels to improve accuracy and observability will be a major contribution for the planning of the hierarchical state estimation structure in a wide area power system.

Exploration of other techniques such as external network modeling and hybrids of these methods with hierarchical state estimation should also be considered for application on a wide area system.

APPENDICES

Appendix A

Common library functions for all the algorithms

```
#Importing libraries (similar to include in C)
from fileread1 import *
from Numeric import *
from Matrix import *
from math import *
from LinearAlgebra import *
from time import *
from MLab import angle as arg
from random import *

#Comments always start with # symbol


def ybus_org(line1,shnt,n):
#Ybus calculation Function
#requires Linedata, shunt capacitor info and number of buses
#linedata is in IEEE cdf format,
#reference [41] has the details of cdf format
    f = open(line1,'r')
    ybus = Matrix([[complex(0,0)]*n]*n)
    lc = []
    for line in f:
        r = float(line.split()[6])
        x = float(line.split()[7])
        b = float(line.split()[8])
        start = int(float(line.split()[0]))
        end = int(float(line.split()[1]))
        tap = float(line.split()[14])
        if(tap == 0.0):
            tap = 1
        else:
            tap = 1/tap
        Y = 1/complex(r,x)
        ybus[start-1,end-1] = ybus[start-1,end-1]-(Y*tap)
        ybus[end-1,start-1] = ybus[end-1,start-1]-(Y*tap)
        ybus[start-1,start-1] = ybus[start-1,start-1]+Y*(tap**2)+complex(0,b/2)
        ybus[end-1,end-1] = ybus[end-1,end-1]+Y+complex(0,b/2)
        send_ch = (tap)*(tap-1)*Y
        rec_ch = (1-tap)*Y
        lc.append([start-1,end-1,b/2+send_ch.imag])
        lc.append([end-1,start-1,b/2+rec_ch.imag])
    lin_ch = Matrix([[0.0]*n]*n)
```

```
    for j in lc:
        lin_ch[j[0],j[1]] = lin_ch[j[0],j[1]]+j[2]
    f.close()
    for j in range(n):
        ybus[j,j] = ybus[j,j] + complex(0,shnt[j,0])
    return ybus,lin_ch


def jacob(zm,x,n,ybus,lst,lc,slack):
# function to form Jacobian matrix for given Measurement set and state vector
# Inputs required: measurement structure, state vector, number of buses, line charging
and reference bus
#Jacobian form the derivative of a measurement with respect to the particular state vector
#very simalr to the jacobiann of Newton raphson method except the matrix is a diagonal
#matrix as state estimation is always over determined
#In this algorithm the measurements are sorted according to the type
#for example injection,measurements are first processed and then the power flow
#measurements etc
#Here the jacobian and the measurement estimate is calculated in the same function
    H = Matrix([[0.0]*x.shape[1]]*zm.shape[0])
    h = Matrix([0.0]*zm.shape[0])
    for i in range(lst[1]):
        sb = int(zm[i,0])
        eb = int(zm[i,1])
        spr = []
        for y in range(n):
            if(ybus[y,sb-1] != 0):
                spr = spr + [y]
        for y in spr:
            H[i,y] = -1*x[0,n+y]*x[0,n-1+sb]*abs(ybus[y,sb-1])*sin(arg(ybus[y,sb-
1])+x[0,y]-x[0,sb-1])
        H[i,sb-1] = 0.0
        H[i,sb-1] = -1*sum(H[i,0:n],1)
        for y in spr:
            H[i,n+y] = x[0,n-1+sb]*abs(ybus[y,sb-1])*cos(arg(ybus[y,sb-1])+x[0,y]-x[0,sb-
1])
        y = sb - 1
        H[i,n+y] = 0
        for z in spr:
            if(z != y):
                H[i,n+y] = H[i,n+y] + x[0,n+z]*abs(ybus[z,sb-1])*cos(arg(ybus[z,sb-
1])+x[0,z]-x[0,sb-1])
        z = y
        tmp1 = 2*x[0,n+z]*(ybus[z,sb-1].real)
        H[i,n+y] = H[i,n+y] + tmp1
        h[0,i] = (H[i,n+sb-1] - (tmp1/2.0))*x[0,n-1+sb]
```

```
    for i in range(lst[1],lst[2]):
        sb = int(zm[i,0])
        eb = int(zm[i,1])
        spr = []
        for y in range(n):
            if(ybus[y,sb-1] != 0):
                spr = spr + [y]
        for y in spr:
            H[i,y] = -1*x[0,n+y]*x[0,n-1+sb]*abs(ybus[y,sb-1])*cos(arg(ybus[y,sb-
1])+x[0,y]-x[0,sb-1])
        H[i,sb-1] = 0.0
        H[i,sb-1] = -1*sum(H[i,0:n],1)
        for y in spr:
            H[i,n+y] = -1*x[0,n-1+sb]*abs(ybus[y,sb-1])*sin(arg(ybus[y,sb-1])+x[0,y]-
x[0,sb-1])
        y = sb-1
        H[i,n+y] = 0
        for z in spr:
            if(z != y):
                H[i,n+y] = H[i,n+y] - x[0,n+z]*abs(ybus[z,sb-1])*sin(arg(ybus[z,sb-1])+x[0,z]-
x[0,sb-1])
        z = y
        tmp1 = 2*x[0,n+z]*(ybus[z,sb-1].imag)
        H[i,n+y] = H[i,n+y] - tmp1
        h[0,i] = (H[i,n+sb-1] + tmp1/2.0)*x[0,n-1+sb]
    for i in range(lst[2],lst[3]):
        sb = int(zm[i,0])
        eb = int(zm[i,1])
        P = x[0,n-1+sb]*x[0,n-1+eb]*abs(ybus[sb-1,eb-1])*cos(arg(ybus[sb-1,eb-
1])+x[0,eb-1]-x[0,sb-1])
        Q = -1*(x[0,n-1+sb]*x[0,n-1+eb]*abs(ybus[sb-1,eb-1])*sin(arg(ybus[sb-1,eb-
1])+x[0,eb-1]-x[0,sb-1]) )
        H[i,sb-1] = -Q
        H[i,eb-1] = Q
        H[i,n+sb-1] = P/x[0,n-1+sb] - (x[0,n-1+sb]*2)*ybus[sb-1,eb-1].real
        H[i,n+eb-1] = P/x[0,n-1+eb]
        h[0,i] = P - (x[0,n-1+sb]**2)*ybus[sb-1,eb-1].real
    for i in range(lst[3],lst[4]):
        sb = int(zm[i,0])
        eb = int(zm[i,1])
        sum1 = lc[sb-1,eb-1]
        P = x[0,n-1+sb]*x[0,n-1+eb]*abs(ybus[sb-1,eb-1])*cos(arg(ybus[sb-1,eb-1])+x[0,-
1+eb]-x[0,-1+sb])
        Q = -1*(x[0,n-1+sb]*x[0,n-1+eb]*abs(ybus[sb-1,eb-1])*sin(arg(ybus[sb-1,eb-
1])+x[0,eb-1]-x[0,sb-1]))
        H[i,sb-1] = P
```

```
        H[i,eb-1] = -P
        H[i,n+sb-1] = Q/x[0,n-1+sb] - (x[0,n-1+sb]*2)*(-ybus[sb-1,eb-1].imag +sum1)
        H[i,n+eb-1] = Q/x[0,n-1+eb]
        h[0,i] = Q - ((x[0,n-1+sb]**2)*(-ybus[sb-1,eb-1].imag + sum1))
    for i in range(lst[4],zm.shape[0]):
        sb = int(zm[i,0])
        eb = int(zm[i,1])
        H[i,int(n+eb-1)] = 1.0
        h[0,i] = x[0,n-1+sb]
    H = take(H,Matrix(range(0,slack)+range(slack+1,n*2)),1)
    H = H[:,0,:]
    return h,Matrix(H)


def ise(zm,busd,fline,n,slack,ang_ref):
#function for integrated state estimate
#requires the measurements, busdata in IEEE cdf format, linedata in IEEE cdf format
#number of buses, slack bus number and angle of reference bus
    tc = time()
# This section sorts the measurement matrix according to the type of measurement

    s = argsort(zm[:,2],0)
    zm = take(zm,s)
    lst = [0]
    zm = zm[:,0,:]
    min1 = 0
# This loop locates the starting of different kinds of measurements in the Measurement
matrix

    for x in range(zm.shape[0]):
        if(min1 < zm[x,2]):
            lst = lst + [x]
            min1 = zm[x,2]
# Adding weights to each measurements from the standard deviation of the measurents
# Injection get 5 percent errors and Power flow get 1 percent errors and Voltages get 0.5
percent error

    R_diag = [.15]*(lst[2]-lst[0])+[.1]*(lst[4]-lst[2])+[.005]*(zm.shape[0]-lst[4])
    R_diag = take(R_diag,s)

#Weighting matrix is initialized to zero as the measurements are uncorrelated

    W = Matrix([[0]*R_diag.shape[0]]*R_diag.shape[0])
    for j in range(R_diag.shape[0]):
        W[j,j] = 1/R_diag[j,0]**2
```

```
    for j in range(R_diag.shape[0]):
        zm[j,3] = zm[j,3] + zm[j,3]*random()*R_diag[j,0]**2

# calculation of state vector size from Number of Buses

    dx = transpose(Matrix([1]*((2*n) -1 )))
    x = Matrix([ang_ref]*(n-1)+[1]*n)
    shnt_sus = busd[:,15]
    del busd
    [ybus,lc] = ybus_org(fline,Matrix(shnt_sus),n)
    p = 1
    m = 0
    for m in range(25):
#the state estimation loop
        x1 = concatenate((x[0,0:slack],Matrix(ang_ref),x[0,slack:(2*n-1)]),1)
        [h,H] = jacob(zm,x1,n,ybus,lst,lc,slack)
        Gi = inverse(transpose(H)*W*H)
        M = (zm[:,3]-h)
        M = transpose(H)*W*Matrix(transpose(M))
        dx = Gi*M
        x = x + transpose(dx)
        p = p + 1
        if(sum(abs(dx)) <= 0.0001):
            break
    x = concatenate((x[0,0:slack],Matrix(ang_ref),x[0,slack:(2*n-1)]),1)
    putdata('output_ise.txt',transpose(x))
    return x,p
```

Appendix B

Main function for integrated state estimator

```
def main():
# A timer to record time of execution
   t1 = time()
#bus data input from the file
   busdata = getdata('busdata.txt')
#line data input from the file
   linedata = getdata('linedata.txt')
#measurements
   zm = getdata('meas.txt')
   slack = 69
   n = 118
#state estimation function is invoked
   ise(zm,busdata,'linedata.txt',n,slack,30.0*pi/180)
   return time() - t1

print main()
```

Appendix C

Main function for traditional hierarchical state estimator

```
#Program first splits the areas into 4 which means splitting the linedata, bus data,
#measurement set etc
quit = 0
recr = 1
U1 = Matrix([30,30,30])
#endless loop, break to quit
while(quit == 0):
   t1 = time()
   busdata = getdata('busdata.txt')
   linedata = getdata('linedata.txt')
   #Number of areas to be split this has to predetermined..
#for this case 4 was taken as number of areas
   N =4
   area = []
   #define area and buses belonging to areas
   area.append(range(1,18)+[30,113,117])
   area.append(range(18,30)+range(31,38)+[114,115])
   area.append(range(38,80)+[116,118])
   area.append(range(80,113))
   #copy of area info with different number system i.e 1 – N-1
   area_1 = []
   area_1.append(range(17)+[29,112,116])
   area_1.append(range(17,29)+range(30,37)+[113,114])
   area_1.append(range(37,79)+[115,117])
   area_1.append(range(79,112))

   #Define slack bus of each bus with global bus numbering
   area_slk = [10,26,69,89]
   dict = []
   for i in range(N):
      dict.append({})
      cnt = 1
      for j in area[i]:
         dict[i][j] = cnt
         cnt = cnt+1
   line_area = []
   for i in range(N):
      line_area.append([])
   #Gather tie-line information
   line_tie = []
   for j in range(linedata.shape[0]):
```

```
    sb = linedata[j,0]
    eb = linedata[j,1]
    for ar in range(N):
        if(dict[ar].get(sb) != None):
            if(dict[ar].get(eb) != None):
                line_area[ar].append(((linedata[j,:]).asarray())[0,:])
            else:
                line_tie.append(((linedata[j,:]).asarray())[0,:])
            break
for i in range(N):
    line_area[i] = Matrix(line_area[i])



line_tie   = Matrix(line_tie)
tie_lines = line_tie

for i in range(N):
    for j in range(line_area[i].shape[0]):
        (line_area[i])[j,0] = dict[i][int((line_area[i])[j,0])]
        (line_area[i])[j,1] = dict[i][int((line_area[i])[j,1])]
bus_area = []
for i in range(N):
    bus_area.append((take(busdata,Matrix(area_1[i])))[0,:,:])
for i in range(N):
    for j in range(len(area[i])):
        bus_area[i][j,0] = dict[i][int(bus_area[i][j,0])]

list_bb = []

#identify the boundary buses
for j in range(tie_lines.shape[0]):
    if(int(tie_lines[j,0])  not in list_bb):
        list_bb.append(int(tie_lines[j,0]))
    else:
        pass

for j in range(tie_lines.shape[0]):
    if(int(tie_lines[j,1])  not in list_bb):
        list_bb.append(int(tie_lines[j,1]))
    else:
        pass

k = len(list_bb)
x_b = Matrix([[0.0]*k+[1.0]*k])
k = 1
```

```
    dict_bb = {}
    dict_bb_rev = {}
    list_bb.sort()
    quit = 0
    recr = 1

    #get measurements
    zm = getdata('Measure1.txt')
    zm_ = []
    zm_t = []
    for i in range(N):
        zm_.append([])


    for j in range(zm.shape[0]):
        if(zm[j,2] in [0,1,4]):
            if((zm[j,0] in list_bb) and (zm[j,2] != 4)):
                continue
            for i in range(N):
                if(zm[j,0] in area[i]):
                    zm_[i].append([dict[i][zm[j,0]],dict[i][zm[j,1]],zm[j,2],zm[j,3]])
                    break
        else:
            if((zm[j,0] in list_bb) and (zm[j,1] in list_bb)):
                lst_b = [zm[j,0],zm[j,1]]
                lst_revb = [zm[j,1],zm[j,0]]
                cond = concatenate([sum(tie_lines[:,0:2] == lst_b,1),sum(tie_lines[:,0:2] ==
lst_revb,1)])
                if(2 in cond):
                    zm_t.append([zm[j,0],zm[j,1],zm[j,2],zm[j,3]])
                    continue
            for i in range(N):
                if(zm[j,0] in area[i]):
                    zm_[i].append([dict[i][zm[j,0]],dict[i][zm[j,1]],zm[j,2],zm[j,3]])
    str_line = []
    for i in range(N):
        zm_[i] = Matrix(zm_[i])
        str_line.append('line_area'+str(i)+'.txt')
        putdata(str_line[i],line_area[i])

    x_ = []
    p = []
#local state estimation starts here
#Threads are started to run the local areas simultaneously
#thread signifies one process
```

```
    #alpha = [0.98,0.95,0.99,0.5]
    alpha = [1,1,1,1]
    for i in range(N):
        x_.append([])
        p.append(0)
    thr = []
    for i in range(N):
        thr.append(0)
        thr[i] =
Thread(None,ise,None,(zm_[i],Matrix(bus_area[i]),str_line[i],bus_area[i].shape[0],dict[i]
[area_slk[i]]-1,0,i,alpha[i]))
        thr[i].start()
    for i in range(N):
        thr[i].join()

#program itself is the main coordinator
#program waits till all the threads finish operation – characteristic of traditional
hierarchical state estimator
    print str( time()-t1)+'\n'
    x_ = []
    flag_delay = 0
    for i in range(N):
        x_.append(0)
        str1 = 'output_ar'+str(i+1)+'.txt'
        x_[i] = getdata(str1)
        if(min(x_[i].shape) == 0):
            print "Area "+str(i+1)+"'s Data Not Arrived... \n"
            print "Using Data Recived in previous step \n"
            flag_delay = flag_delay+1
            x_[i] = getdata('output_old'+str(i+1)+'.txt')

    if(flag_delay == 0):
        print "Recived all data \n"
#coordination started
    print  "coordinating...\n"

    zm_t = Matrix(zm_t)
    for j in list_bb:
        dict_bb[k] = j
        dict_bb_rev[j] = k
        k = k + 1
    dict_area = {}
    bbus = Matrix([[0.0]*bus_area[0].shape[1]]*(k-1))
    for j in range(1,k):
        for i in range(N):
            if(dict_bb[j] in area[i]):
```

```
            x_b[0,j-1] = x_[i][0,dict[i][dict_bb[j]]-1]
            x_b[0,k+j-2] = x_[i][0,dict[i][dict_bb[j]]+bus_area[i].shape[0]-1]
            dict_area[j] = i+1

tie_new = tie_lines
for j in range(tie_lines.shape[0]):
   tie_new[j,0] = int(dict_bb_rev[int(tie_new[j,0])])
   tie_new[j,1] = int(dict_bb_rev[int(tie_new[j,1])])
bbus = Matrix([[0.0]*busdata.shape[1]]*len(list_bb))
cnt = 0
for j in list_bb:
   bbus[cnt,:] = busdata[j-1,:]
   bbus[cnt,0] = dict_bb_rev[j]
   cnt = cnt + 1
putdata('tie_new.txt',tie_new)
shnt_sus = bbus[:,15]
[ybus,lc] = ybus_org('tie_new.txt',shnt_sus,bbus.shape[0])
for j in range(zm_t.shape[0]):
   zm_t[j,0] = dict_bb_rev[zm_t[j,0]]
   zm_t[j,1] = dict_bb_rev[zm_t[j,1]]
s = argsort(zm_t[:,2],0)
zm_t = take(zm_t,s)
lst = [0]
zm_t = zm_t[:,0,:]
min1 = 2

for x in range(zm_t.shape[0]):
   if(min1 < zm_t[x,2]):
      lst = x
      break


U = U1
U = U * pi/180
p = 1
area_recv = [1.0,1.0,1.0,1.0]
for m in range(25):
   u = concatenate([U[0,0:2],Matrix([30*pi/180]),Matrix(U[0,2])],1)
   [h,H,Ru] = jacob_coord(zm_t,lst+1,x_b,len(list_bb),4,ybus,lc,u,dict_area,area_recv)
   Gi = inverse(Matrix(transpose(H))*Ru*H)
   M = (zm_t[:,3]-h)
   M = transpose(H)*(Matrix(inverse(Ru))*Matrix(transpose(M)))
   dx = Gi*M
   U = U + transpose(dx)
   p = p + 1
   if(sum(abs(dx)) <= 0.0001):
```

```
        print "Coordination Done in "+str(m+1)+" Iterration"
        break
print "Coordination Vector is"
print u*180/pi
n = 118
for i in range(N):
    x_[i][0,0:bus_area[i].shape[0]] = x_[i][0,0:bus_area[i].shape[0]] + u[0,i]
x_lf = getdata('output1.txt')
x_con = Matrix([[0.0]*2*n])
for m in range(n):
    for i in range(N):
        if(m+1 in area[i]):
            x_con[0,m] = x_[i][0,dict[i][m+1]-1]
            x_con[0,n+m] =  x_[i][0,dict[i][m+1]-1+bus_area[i].shape[0]]
err = x_con - x_lf
err = err / x_lf
putdata('error.txt',abs(err)*100)
putdata('consolidate.txt',x_con)
print str( time() - t1)+'\n'
recr = recr + 1
print recr
U1 = U
```

Appendix D

Main function for modified coordination algorithm

```
#program very similar to traditional
#has separate routine to identify the weigts and probability of delay of area and loss of
#area
#starts by splitting areas
quit = 0
recr = 1
U1 = Matrix([36,30,39])
while(quit == 0):
   t1 = time()
   busdata = getdata('busdata.txt')
   linedata = getdata('linedata.txt')
   N =4

   area = []
   area.append(range(1,18)+[30,113,117])
   area.append(range(18,30)+range(31,38)+[114,115])
   area.append(range(38,80)+[116,118])
   area.append(range(80,113))
   area_1 = []
   area_1.append(range(17)+[29,112,116])
   area_1.append(range(17,29)+range(30,37)+[113,114])
   area_1.append(range(37,79)+[115,117])
   area_1.append(range(79,112))

   area_slk = [10,26,69,89]
   dict = []
   for i in range(N):
      dict.append({})
      cnt = 1
      for j in area[i]:
         dict[i][j] = cnt
         cnt = cnt+1
   line_area = []
   for i in range(N):
      line_area.append([])

   line_tie = []
   for j in range(linedata.shape[0]):
      sb = linedata[j,0]
      eb = linedata[j,1]
      for ar in range(N):
         if(dict[ar].get(sb) != None):
```

```
            if(dict[ar].get(eb) != None):
               line_area[ar].append(((linedata[j,:]).asarray())[0,:])
            else:
               line_tie.append(((linedata[j,:]).asarray())[0,:])
            break
for i in range(N):
   line_area[i] = Matrix(line_area[i])



line_tie   = Matrix(line_tie)
tie_lines = line_tie

for i in range(N):
   for j in range(line_area[i].shape[0]):
      (line_area[i])[j,0] = dict[i][int((line_area[i])[j,0])]
      (line_area[i])[j,1] = dict[i][int((line_area[i])[j,1])]
bus_area = []
for i in range(N):
   bus_area.append((take(busdata,Matrix(area_1[i])))[0,:,:])
for i in range(N):
   for j in range(len(area[i])):
      bus_area[i][j,0] = dict[i][int(bus_area[i][j,0])]

list_bb = []

#identify the boundary buses
for j in range(tie_lines.shape[0]):
   if(int(tie_lines[j,0])  not in list_bb):
      list_bb.append(int(tie_lines[j,0]))
   else:
      pass

for j in range(tie_lines.shape[0]):
   if(int(tie_lines[j,1])  not in list_bb):
      list_bb.append(int(tie_lines[j,1]))
   else:
      pass

k = len(list_bb)
x_b = Matrix([[0.0]*k+[1.0]*k])
k = 1
dict_bb = {}
dict_bb_rev = {}
list_bb.sort()
quit = 0
```

```
    zm = getdata('Measure1.txt')
    zm_ = []
    zm_t = []
    for i in range(N):
        zm_.append([])


    for j in range(zm.shape[0]):
        if(zm[j,2] in [0,1,4]):
            if((zm[j,0] in list_bb) and (zm[j,2] != 4)):
                continue
            for i in range(N):
                if(zm[j,0] in area[i]):
                    zm_[i].append([dict[i][zm[j,0]],dict[i][zm[j,1]],zm[j,2],zm[j,3]])
                    break
        else:
            if((zm[j,0] in list_bb) and (zm[j,1] in list_bb)):
                lst_b = [zm[j,0],zm[j,1]]
                lst_revb = [zm[j,1],zm[j,0]]
                cond = concatenate([sum(tie_lines[:,0:2] == lst_b,1),sum(tie_lines[:,0:2] ==
lst_revb,1)])
                if(2 in cond):
                    zm_t.append([zm[j,0],zm[j,1],zm[j,2],zm[j,3]])
                    continue
            for i in range(N):
                if(zm[j,0] in area[i]):
                    zm_[i].append([dict[i][zm[j,0]],dict[i][zm[j,1]],zm[j,2],zm[j,3]])
    str_line = []
    for i in range(N):
        zm_[i] = Matrix(zm_[i])
        str_line.append('line_area'+str(i)+'.txt')
        putdata(str_line[i],line_area[i])

    x_ = []
    p = []
    alpha = [0.98,0.95,0.99,0.5]
    for i in range(N):
        x_.append([])
        p.append(0)
    thr = []
    for i in range(N):
        thr.append(0)
        thr[i] =
Thread(None,ise,None,(zm_[i],Matrix(bus_area[i]),str_line[i],bus_area[i].shape[0],dict[i]
[area_slk[i]]-1,0,i,alpha[i]))
```

```
    thr[i].start()
for i in range(N):
    thr[i].join()

print str( time()-t1)+'\n'
x_ = []
flag_delay = 0
for i in range(N):
    x_.append(0)
    str1 = 'output_ar'+str(i+1)+'.txt'
    x_[i] = getdata(str1)
    if(min(x_[i].shape) == 0):
        print "Area "+str(i+1)+"'s Data Not Arrived... \n"
        print "Using Data Recived in previous step \n"
        flag_delay = flag_delay+1
        x_[i] = getdata('output_old'+str(i+1)+'.txt')

if(flag_delay == 0):
    print "Recived all data \n"

print  "coordinating...\n"

zm_t = Matrix(zm_t)
for j in list_bb:
    dict_bb[k] = j
    dict_bb_rev[j] = k
    k = k + 1
dict_area = {}
bbus = Matrix([[0.0]*bus_area[0].shape[1]]*(k-1))
for j in range(1,k):
    for i in range(N):
        if(dict_bb[j] in area[i]):
            x_b[0,j-1] = x_[i][0,dict[i][dict_bb[j]]-1]
            x_b[0,k+j-2] = x_[i][0,dict[i][dict_bb[j]]+bus_area[i].shape[0]-1]
            dict_area[j] = i+1

tie_new = tie_lines
for j in range(tie_lines.shape[0]):
    tie_new[j,0] = int(dict_bb_rev[int(tie_new[j,0])])
    tie_new[j,1] = int(dict_bb_rev[int(tie_new[j,1])])
bbus = Matrix([[0.0]*busdata.shape[1]]*len(list_bb))
cnt = 0
for j in list_bb:
    bbus[cnt,:] = busdata[j-1,:]
    bbus[cnt,0] = dict_bb_rev[j]
    cnt = cnt + 1
```

```
    putdata('tie_new.txt',tie_new)
    shnt_sus = bbus[:,15]
    [ybus,lc] = ybus_org('tie_new.txt',shnt_sus,bbus.shape[0])
    for j in range(zm_t.shape[0]):
        zm_t[j,0] = dict_bb_rev[zm_t[j,0]]
        zm_t[j,1] = dict_bb_rev[zm_t[j,1]]
    s = argsort(zm_t[:,2],0)
    zm_t = take(zm_t,s)
    lst = [0]
    zm_t = zm_t[:,0,:]
    min1 = 2

    for x in range(zm_t.shape[0]):
        if(min1 < zm_t[x,2]):
            lst = x
            break


    U = U1
    U = U * pi/180
    p = 1
    u = concatenate([U[0,0:2],Matrix([30*pi/180]),Matrix(U[0,2])],1)
    [h,H] = jacob_coord(zm_t,lst+1,x_b,len(list_bb),4,ybus,lc,u,dict_area)
    if(recr == 1):
        Pmin = Matrix([[1e2,0,0],[0,1e2,0],[0,0,1e2]])
    M = (zm_t[:,3]-h)
    K = Pmin*transpose(H)*inverse(H*Pmin*transpose(H) +
Matrix(identity(zm_t.shape[0])*1e-4))

    dx = K*transpose(M)
    U = U + transpose(dx)
    p = p + 1
    Pmin = Matrix((identity(U.shape[0]) - K*H)*Pmin)
    print "Coordination Vector is"
    u = concatenate([U[0,0:2],Matrix([30*pi/180]),Matrix(U[0,2])],1)
    print u*180/pi
    n = 118
    for i in range(N):
        x_[i][0,0:bus_area[i].shape[0]] = x_[i][0,0:bus_area[i].shape[0]] + u[0,i]
    x_lf = getdata('output1.txt')
    x_con = Matrix([[0.0]*2*n])
    for m in range(n):
        for i in range(N):
            if(m+1 in area[i]):
                x_con[0,m] = x_[i][0,dict[i][m+1]-1]
                x_con[0,n+m] =  x_[i][0,dict[i][m+1]-1+bus_area[i].shape[0]]
```

```
err = x_con - x_lf
err = err / x_lf
putdata('error.txt',abs(err)*100)
putdata('consolidate.txt',x_con)
print str( time() - t1)+'\n'
recr = recr + 1
print recr
U1 = U
```

Appendix E

Libraries and main function for inclusion of phasor measurements

in lower level of estimation

```
#main function and libraries are changed to accomadate newer measurement type i.e
#voltage angle measurement
#Check Jacob – function (jacobian) for updated phasor measurement handling routines
from fileread1 import *
from Numeric import *
from Matrix import *
from math import *
from LinearAlgebra import *
from time import *
from MLab import angle as arg
from random import *
from threading import *

def bernoulli(p):
    x = random()
    if(x <= p):
        return 1
    else:
        return 0

def filecopy(str1,str2):
    file1 = open(str1,'r')
    file2 = open(str2,'w')
    for i in file1:
        file2.write(i)
    file1.close()
    file2.close()
    return 0

def filedel(str1):
    file1 = open(str1,'w')
    file1.close()

def jacob(zm,x,n,ybus,lst,lc,slack):
    H = Matrix([[0.0]*x.shape[1]]*zm.shape[0])
    h = Matrix([0.0]*zm.shape[0])
    for i in range(lst[1]):
                    sb = int(zm[i,0])
                    eb = int(zm[i,1])
```

```
                H[i,int(eb-1)] = 1.0
                h[0,i] = x[0,-1+sb]
        #for i in range(lst[1],lst[2]):
        #    sb = int(zm[i,0])
        #    eb = int(zm[i,1])
        #    H[i,int(eb-1)] = 1.0
        #    h[0,i] = x[0,-1+sb]
    for i in range(lst[1],lst[2]):
        sb = int(zm[i,0])
        eb = int(zm[i,1])
        P = x[0,n-1+sb]*x[0,n-1+eb]*abs(ybus[sb-1,eb-1])*cos(arg(ybus[sb-1,eb-
1])+x[0,eb-1]-x[0,sb-1])
        Q = -1*(x[0,n-1+sb]*x[0,n-1+eb]*abs(ybus[sb-1,eb-1])*sin(arg(ybus[sb-1,eb-
1])+x[0,eb-1]-x[0,sb-1]) )
        H[i,sb-1] = -Q
        H[i,eb-1] = Q
        H[i,n+sb-1] = P/x[0,n-1+sb] - (x[0,n-1+sb]*2)*ybus[sb-1,eb-1].real
        H[i,n+eb-1] = P/x[0,n-1+eb]
        h[0,i] = P - (x[0,n-1+sb]**2)*ybus[sb-1,eb-1].real
    for i in range(lst[2],lst[3]):
        sb = int(zm[i,0])
        eb = int(zm[i,1])
        sum1 = lc[sb-1,eb-1]
        P = x[0,n-1+sb]*x[0,n-1+eb]*abs(ybus[sb-1,eb-1])*cos(arg(ybus[sb-1,eb-1])+x[0,-
1+eb]-x[0,-1+sb])
        Q = -1*(x[0,n-1+sb]*x[0,n-1+eb]*abs(ybus[sb-1,eb-1])*sin(arg(ybus[sb-1,eb-
1])+x[0,eb-1]-x[0,sb-1]))
        H[i,sb-1] = P
        H[i,eb-1] = -P
        H[i,n+sb-1] = Q/x[0,n-1+sb] - (x[0,n-1+sb]*2)*(-ybus[sb-1,eb-1].imag +sum1)
        H[i,n+eb-1] = Q/x[0,n-1+eb]
        h[0,i] = Q - ((x[0,n-1+sb]**2)*(-ybus[sb-1,eb-1].imag + sum1))
    for i in range(lst[3],zm.shape[0]):
        sb = int(zm[i,0])
        eb = int(zm[i,1])
        H[i,int(n+eb-1)] = 1.0
        h[0,i] = x[0,n-1+sb]
    H = take(H,Matrix(range(0,slack)+range(slack+1,n*2)),1)
    H = H[:,0,:]
    return h,Matrix(H)



def jacob_coord(zm,lst,x,n,N,ybus,lc,u,dict_area,area_recv):
    H = Matrix([[0.0]*N]*zm.shape[0])
```

```
    h = Matrix([0.0]*zm.shape[0])
    R = Matrix([[0.0]*zm.shape[0]]*zm.shape[0])
    for i in range(lst):
        sb = int(zm[i,0])
        eb = int(zm[i,1])
        P = x[0,n-1+sb]*x[0,n-1+eb]*abs(ybus[sb-1,eb-1])*cos(arg(ybus[sb-1,eb-
1])+x[0,eb-1]+u[0,((dict_area[eb])-1)]-u[0,((dict_area[sb])-1)]-x[0,sb-1])
        Q = -1*x[0,n-1+sb]*x[0,n-1+eb]*abs(ybus[sb-1,eb-1])*sin(arg(ybus[sb-1,eb-
1])+x[0,eb-1]+u[0,((dict_area[eb])-1)]-u[0,((dict_area[sb])-1)]-x[0,sb-1])
        H[i,dict_area[sb]-1] = -Q
        H[i,dict_area[eb]-1] = Q
        h[0,i] = P - (x[0,n-1+sb]**2)*ybus[sb-1,eb-1].real
        R[i,i] = area_recv[max(dict_area[sb],dict_area[eb])-1]
    for i in range(lst,zm.shape[0]):
        sb = int(zm[i,0])
        eb = int(zm[i,1])
        sum1 = lc[sb-1,eb-1]
        P = x[0,n-1+sb]*x[0,n-1+eb]*abs(ybus[sb-1,eb-1])*cos(arg(ybus[sb-1,eb-
1])+x[0,eb-1]+u[0,dict_area[eb]-1]-u[0,dict_area[sb]-1]-x[0,sb-1])
        Q = -1*x[0,n-1+sb]*x[0,n-1+eb]*abs(ybus[sb-1,eb-1])*sin(arg(ybus[sb-1,eb-
1])+x[0,eb-1]+u[0,dict_area[eb]-1]-u[0,dict_area[sb]-1]-x[0,sb-1])
        H[i,dict_area[sb]-1] = P
        H[i,dict_area[eb]-1] = -P
        h[0,i] = Q - ((x[0,n-1+sb]**2)*(-ybus[sb-1,eb-1].imag + sum1))
        R[i,i] = area_recv[max(dict_area[sb],dict_area[eb])-1]
    H = take(H,[0,1,3],1)
    return h,Matrix(H),R

def ybus_org(line1,shnt,n):
    f = open(line1,'r')
    ybus = Matrix([[complex(0,0)]*n]*n)
    lc = []
    for line in f:
        r = float(line.split()[6])
        x = float(line.split()[7])
        b = float(line.split()[8])
        start = int(float(line.split()[0]))
        end = int(float(line.split()[1]))
        tap = float(line.split()[14])
        if(tap == 0.0):
            tap = 1
        else:
            tap = 1/tap
        Y = 1/complex(r,x)
        ybus[start-1,end-1] = ybus[start-1,end-1]-(Y*tap)
        ybus[end-1,start-1] = ybus[end-1,start-1]-(Y*tap)
```

```
        ybus[start-1,start-1] = ybus[start-1,start-1]+Y*(tap**2)+complex(0,b/2)
        ybus[end-1,end-1] = ybus[end-1,end-1]+Y+complex(0,b/2)
        send_ch = (tap)*(tap-1)*Y
        rec_ch = (1-tap)*Y
        lc.append([start-1,end-1,b/2+send_ch.imag])
        lc.append([end-1,start-1,b/2+rec_ch.imag])
    lin_ch = Matrix([[0.0]*n]*n)
    for j in lc:
        lin_ch[j[0],j[1]] = lin_ch[j[0],j[1]]+j[2]
    f.close()
    for j in range(n):
        ybus[j,j] = ybus[j,j] + complex(0,shnt[j,0])
    return ybus,lin_ch
```

```
def ise(zm,busd,fline,n,slack,ang_ref,area,alpha):
    old_fl = 'output_old'+str(area+1)+'.txt'
    org_fl = 'output_ar'+str(area+1)+'.txt'
    filecopy(org_fl,old_fl)
    filedel(org_fl)
    tc = time()
# This section sorts the measurement matrix according to the type of measurement

    s = argsort(zm[:,2],0)
    zm = take(zm,s)
    lst = [0]
    zm = zm[:,0,:]
    min1 = 0
# This loop locates the starting of different kinds of measurements in the Measurement
matrix

    for x in range(zm.shape[0]):
        if(min1 < zm[x,2]):
            lst = lst + [x]
            min1 = zm[x,2]
    print lst
# Adding weights to each measurements from the standard deviation of the measurents
# Injection get 5 percent errors and Power flow get 1 percent errors and Voltages get 0.5
percent error

    R_diag = [.005]*(lst[1]-lst[0])+[0.01]*(lst[3]-lst[1])+[.005]*(zm.shape[0]-lst[3])
```

```
   R_diag = take(R_diag,s)

#Weighting matrix is initialized to zero as the measurements are uncorrelated

   W = Matrix([[0]*R_diag.shape[0]]*R_diag.shape[0])
   for j in range(R_diag.shape[0]):
      W[j,j] = 1/R_diag[j,0]**2

   for j in range(R_diag.shape[0]):
      zm[j,3] = zm[j,3] + zm[j,3]*random()*R_diag[j,0]**2

#Number of Buses

   dx = transpose(Matrix([1]*((2*n) -1 )))
   x = Matrix([ang_ref]*(n-1)+[1]*n)
   shnt_sus = busd[:,15]
   del busd
   [ybus,lc] = ybus_org(fline,Matrix(shnt_sus),n)
   p = 1
   m = 0
   for m in range(25):
      x1 = concatenate((x[0,0:slack],Matrix(ang_ref),x[0,slack:(2*n-1)]),1)
      [h,H] = jacob(zm,x1,n,ybus,lst,lc,slack)
      Gi = inverse(transpose(H)*W*H)
      M = (zm[:,3]-h)
      M = transpose(H)*W*Matrix(transpose(M))
      dx = Gi*M
      x = x + transpose(dx)
      p = p + 1
      if(sum(abs(dx)) <= 0.0001):
         break
   x = concatenate((x[0,0:slack],Matrix(ang_ref*pi/180),x[0,slack:(2*n-1)]),1)
   delay_synth = random()*5*(1+bernoulli(alpha))
   print delay_synth
   ##sleep(delay_synth)
   putdata('output_ar'+str(area+1)+'.txt',x)
   print 'Area '+str(area+1)+' Converged in '+str(p-1)+' Iterations '+str(time() - tc)+'
Seconds \n'
   return x,p

quit = 0
recr = 1
U1 = Matrix([30,30,30])
while(quit == 0):
   t1 = time()
   busdata = getdata('busdata.txt')
```

```
linedata = getdata('linedata.txt')
N =4

area = []
area.append(range(1,18)+[30,113,117])
area.append(range(18,30)+range(31,38)+[114,115])
area.append(range(38,80)+[116,118])
area.append(range(80,113))
area_1 = []
area_1.append(range(17)+[29,112,116])
area_1.append(range(17,29)+range(30,37)+[113,114])
area_1.append(range(37,79)+[115,117])
area_1.append(range(79,112))

area_slk = [10,26,69,89]
dict = []
for i in range(N):
   dict.append({})
   cnt = 1
   for j in area[i]:
      dict[i][j] = cnt
      cnt = cnt+1
line_area = []
for i in range(N):
   line_area.append([])

line_tie = []
for j in range(linedata.shape[0]):
   sb = linedata[j,0]
   eb = linedata[j,1]
   for ar in range(N):
      if(dict[ar].get(sb) != None):
         if(dict[ar].get(eb) != None):
            line_area[ar].append(((linedata[j,:]).asarray())[0,:])
         else:
            line_tie.append(((linedata[j,:]).asarray())[0,:])
         break
for i in range(N):
   line_area[i] = Matrix(line_area[i])



line_tie   = Matrix(line_tie)
tie_lines = line_tie

for i in range(N):
```

```
    for j in range(line_area[i].shape[0]):
        (line_area[i])[j,0] = dict[i][int((line_area[i])[j,0])]
        (line_area[i])[j,1] = dict[i][int((line_area[i])[j,1])]
bus_area = []
for i in range(N):
    bus_area.append((take(busdata,Matrix(area_1[i])))[0,:,:])
for i in range(N):
    for j in range(len(area[i])):
        bus_area[i][j,0] = dict[i][int(bus_area[i][j,0])]

list_bb = []

#identify the boundary buses
for j in range(tie_lines.shape[0]):
    if(int(tie_lines[j,0])  not in list_bb):
        list_bb.append(int(tie_lines[j,0]))
    else:
        pass

for j in range(tie_lines.shape[0]):
    if(int(tie_lines[j,1])  not in list_bb):
        list_bb.append(int(tie_lines[j,1]))
    else:
        pass

k = len(list_bb)
x_b = Matrix([[0.0]*k+[1.0]*k])
k = 1
dict_bb = {}
dict_bb_rev = {}
list_bb.sort()
quit = 0
recr = 1

zm = getdata('Meas_pmu.txt')
zm_ = []
zm_t = []
for i in range(N):
    zm_.append([])


for j in range(zm.shape[0]):
    if(zm[j,2] in [0,1,4]):
        if((zm[j,0] in list_bb) and (zm[j,2] != 4)):
            continue
        for i in range(N):
```

```
            if(zm[j,0] in area[i]):
               zm_[i].append([dict[i][zm[j,0]],dict[i][zm[j,1]],zm[j,2],zm[j,3]])
               break
      else:
         if((zm[j,0] in list_bb) and (zm[j,1] in list_bb)):
            lst_b = [zm[j,0],zm[j,1]]
            lst_revb = [zm[j,1],zm[j,0]]
            cond = concatenate([sum(tie_lines[:,0:2] == lst_b,1),sum(tie_lines[:,0:2] ==
lst_revb,1)])
            if(2 in cond):
               zm_t.append([zm[j,0],zm[j,1],zm[j,2],zm[j,3]])
               continue
         for i in range(N):
            if(zm[j,0] in area[i]):
               zm_[i].append([dict[i][zm[j,0]],dict[i][zm[j,1]],zm[j,2],zm[j,3]])
   str_line = []
   for i in range(N):
      zm_[i] = Matrix(zm_[i])
      str_line.append('line_area'+str(i)+'.txt')
      putdata(str_line[i],line_area[i])

   x_ = []
   p = []
   #alpha = [0.98,0.95,0.99,0.5]
   alpha = [1,1,1,1]
   for i in range(N):
      x_.append([])
      p.append(0)
   thr = []
   for i in range(N):
      thr.append(0)
      thr[i] =
Thread(None,ise,None,(zm_[i],Matrix(bus_area[i]),str_line[i],bus_area[i].shape[0],dict[i]
[area_slk[i]]-1,0,i,alpha[i]))
      thr[i].start()
   for i in range(N):
      thr[i].join()

   print str( time()-t1)+'\n'
   x_ = []
   flag_delay = 0
   for i in range(N):
      x_.append(0)
      str1 = 'output_ar'+str(i+1)+'.txt'
      x_[i] = getdata(str1)
      if(min(x_[i].shape) == 0):
```

```
        print "Area "+str(i+1)+"'s Data Not Arrived... \n"
        print "Using Data Recived in previous step \n"
        flag_delay = flag_delay+1
        x_[i] = getdata('output_old'+str(i+1)+'.txt')

if(flag_delay == 0):
    print "Recived all data \n"

print  "coordinating...\n"

zm_t = Matrix(zm_t)
for j in list_bb:
    dict_bb[k] = j
    dict_bb_rev[j] = k
    k = k + 1
dict_area = {}
bbus = Matrix([[0.0]*bus_area[0].shape[1]]*(k-1))
for j in range(1,k):
    for i in range(N):
        if(dict_bb[j] in area[i]):
            x_b[0,j-1] = x_[i][0,dict[i][dict_bb[j]]-1]
            x_b[0,k+j-2] = x_[i][0,dict[i][dict_bb[j]]+bus_area[i].shape[0]-1]
            dict_area[j] = i+1

tie_new = tie_lines
for j in range(tie_lines.shape[0]):
    tie_new[j,0] = int(dict_bb_rev[int(tie_new[j,0])])
    tie_new[j,1] = int(dict_bb_rev[int(tie_new[j,1])])
bbus = Matrix([[0.0]*busdata.shape[1]]*len(list_bb))
cnt = 0
for j in list_bb:
    bbus[cnt,:] = busdata[j-1,:]
    bbus[cnt,0] = dict_bb_rev[j]
    cnt = cnt + 1
putdata('tie_new.txt',tie_new)
shnt_sus = bbus[:,15]
[ybus,lc] = ybus_org('tie_new.txt',shnt_sus,bbus.shape[0])
for j in range(zm_t.shape[0]):
    zm_t[j,0] = dict_bb_rev[zm_t[j,0]]
    zm_t[j,1] = dict_bb_rev[zm_t[j,1]]
s = argsort(zm_t[:,2],0)
zm_t = take(zm_t,s)
lst = [0]
zm_t = zm_t[:,0,:]
min1 = 2
```

```
for x in range(zm_t.shape[0]):
   if(min1 < zm_t[x,2]):
      lst = x
      break


U = U1
U = U * pi/180
p = 1
area_recv = [1.0,1.0,1.0,1.0]
for m in range(25):
   u = concatenate([U[0,0:2],Matrix([30*pi/180]),Matrix(U[0,2])],1)
   [h,H,Ru] = jacob_coord(zm_t,lst+1,x_b,len(list_bb),4,ybus,lc,u,dict_area,area_recv)
   Gi = inverse(Matrix(transpose(H))*Ru*H)
   M = (zm_t[:,3]-h)
   M = transpose(H)*(Matrix(inverse(Ru))*Matrix(transpose(M)))
   dx = Gi*M
   U = U + transpose(dx)
   p = p + 1
   if(sum(abs(dx)) <= 0.0001):
      print "Coordination Done in "+str(m+1)+" Iterration"
      break
print "Coordination Vector is"
print u*180/pi
n = 118
for i in range(N):
   x_[i][0,0:bus_area[i].shape[0]] = x_[i][0,0:bus_area[i].shape[0]] + u[0,i]
x_lf = getdata('output1.txt')
x_con = Matrix([[0.0]*2*n])
for m in range(n):
   for i in range(N):
      if(m+1 in area[i]):
         x_con[0,m] = x_[i][0,dict[i][m+1]-1]
         x_con[0,n+m] =  x_[i][0,dict[i][m+1]-1+bus_area[i].shape[0]]
err = x_con - x_lf
err = err / x_lf
putdata('error.txt',abs(err)*100)
putdata('consolidate.txt',x_con)
print str( time() - t1)+'\n'
recr = recr + 1
print recr
U1 = U
```

Appendix F

Main function for phasor measurements

implementation in coordination level

```
#This program is for synchronized phasor measurements in coordination level
#the function has linear coordinator
quit = 0
recr = 1
U1 = Matrix([30,30,30])
while(quit == 0):
    t1 = time()
    busdata = getdata('busdata.txt')
    linedata = getdata('linedata.txt')
    N =4

    area = []
    area.append(range(1,18)+[30,113,117])
    area.append(range(18,30)+range(31,38)+[114,115])
    area.append(range(38,80)+[116,118])
    area.append(range(80,113))
    area_1 = []
    area_1.append(range(17)+[29,112,116])
    area_1.append(range(17,29)+range(30,37)+[113,114])
    area_1.append(range(37,79)+[115,117])
    area_1.append(range(79,112))

    area_slk = [10,26,69,89]
    dict = []
    for i in range(N):
        dict.append({})
        cnt = 1
        for j in area[i]:
            dict[i][j] = cnt
            cnt = cnt+1
    line_area = []
    for i in range(N):
        line_area.append([])

    line_tie = []
    for j in range(linedata.shape[0]):
        sb = linedata[j,0]
        eb = linedata[j,1]
        for ar in range(N):
```

```
        if(dict[ar].get(sb) != None):
            if(dict[ar].get(eb) != None):
                line_area[ar].append(((linedata[j,:]).asarray())[0,:])
            else:
                line_tie.append(((linedata[j,:]).asarray())[0,:])
            break
for i in range(N):
    line_area[i] = Matrix(line_area[i])



line_tie   = Matrix(line_tie)
tie_lines = line_tie

for i in range(N):
    for j in range(line_area[i].shape[0]):
        (line_area[i])[j,0] = dict[i][int((line_area[i])[j,0])]
        (line_area[i])[j,1] = dict[i][int((line_area[i])[j,1])]
bus_area = []
for i in range(N):
    bus_area.append((take(busdata,Matrix(area_1[i])))[0,:,:])
for i in range(N):
    for j in range(len(area[i])):
        bus_area[i][j,0] = dict[i][int(bus_area[i][j,0])]

list_bb = []

#identify the boundary buses
for j in range(tie_lines.shape[0]):
    if(int(tie_lines[j,0])  not in list_bb):
        list_bb.append(int(tie_lines[j,0]))
    else:
        pass

for j in range(tie_lines.shape[0]):
    if(int(tie_lines[j,1])  not in list_bb):
        list_bb.append(int(tie_lines[j,1]))
    else:
        pass

k = len(list_bb)
x_b = Matrix([[0.0]*k+[1.0]*k])
k = 1
dict_bb = {}
dict_bb_rev = {}
list_bb.sort()
```

```
    quit = 0
    recr = 1

    zm = getdata('Meas_pmu.txt')
    zm_ = []
    zm_t = []
    for i in range(N):
        zm_.append([])


    for j in range(zm.shape[0]):
        if(zm[j,2] in [0,1,4]):
            if((zm[j,0] in list_bb) and (zm[j,2] != 4)):
                continue
            for i in range(N):
                if(zm[j,0] in area[i]):
                    zm_[i].append([dict[i][zm[j,0]],dict[i][zm[j,1]],zm[j,2],zm[j,3]])
                    break
        else:
            if((zm[j,0] in list_bb) and (zm[j,1] in list_bb)):
                lst_b = [zm[j,0],zm[j,1]]
                lst_revb = [zm[j,1],zm[j,0]]
                cond = concatenate([sum(tie_lines[:,0:2] == lst_b,1),sum(tie_lines[:,0:2] ==
lst_revb,1)])
                if(2 in cond):
                    zm_t.append([zm[j,0],zm[j,1],zm[j,2],zm[j,3]])
                    continue
            for i in range(N):
                if(zm[j,0] in area[i]):
                    zm_[i].append([dict[i][zm[j,0]],dict[i][zm[j,1]],zm[j,2],zm[j,3]])
    str_line = []
    for i in range(N):
        zm_[i] = Matrix(zm_[i])
        str_line.append('line_area'+str(i)+'.txt')
        putdata(str_line[i],line_area[i])

    x_ = []
    p = []
    #alpha = [0.98,0.95,0.99,0.5]
    alpha = [1,1,1,1]
    for i in range(N):
        x_.append([])
        p.append(0)
    thr = []
    for i in range(N):
        thr.append(0)
```

```
        thr[i] =
Thread(None,ise,None,(zm_[i],Matrix(bus_area[i]),str_line[i],bus_area[i].shape[0],dict[i]
[area_slk[i]]-1,0,i,alpha[i]))
        thr[i].start()
    for i in range(N):
        thr[i].join()

    print str( time()-t1)+'\n'
    x_ = []
    flag_delay = 0
    for i in range(N):
        x_.append(0)
        str1 = 'output_ar'+str(i+1)+'.txt'
        x_[i] = getdata(str1)
        if(min(x_[i].shape) == 0):
            print "Area "+str(i+1)+"'s Data Not Arrived... \n"
            print "Using Data Recived in previous step \n"
            flag_delay = flag_delay+1
            x_[i] = getdata('output_old'+str(i+1)+'.txt')

    if(flag_delay == 0):
        print "Recived all data \n"

    print  "coordinating...\n"

    zm_t = Matrix(zm_t)
    for j in list_bb:
        dict_bb[k] = j
        dict_bb_rev[j] = k
        k = k + 1
    dict_area = {}
    bbus = Matrix([[0.0]*bus_area[0].shape[1]]*(k-1))
    for j in range(1,k):
        for i in range(N):
            if(dict_bb[j] in area[i]):
                x_b[0,j-1] = x_[i][0,dict[i][dict_bb[j]]-1]
                x_b[0,k+j-2] = x_[i][0,dict[i][dict_bb[j]]+bus_area[i].shape[0]-1]
                dict_area[j] = i+1

    tie_new = tie_lines
    for j in range(tie_lines.shape[0]):
        tie_new[j,0] = int(dict_bb_rev[int(tie_new[j,0])])
        tie_new[j,1] = int(dict_bb_rev[int(tie_new[j,1])])
    bbus = Matrix([[0.0]*busdata.shape[1]]*len(list_bb))
    cnt = 0
    for j in list_bb:
```

```
        bbus[cnt,:] = busdata[j-1,:]
        bbus[cnt,0] = dict_bb_rev[j]
        cnt = cnt + 1
    putdata('tie_new.txt',tie_new)
    shnt_sus = bbus[:,15]
    [ybus,lc] = ybus_org('tie_new.txt',shnt_sus,bbus.shape[0])
    for j in range(zm_t.shape[0]):
        zm_t[j,0] = dict_bb_rev[zm_t[j,0]]
        zm_t[j,1] = dict_bb_rev[zm_t[j,1]]
    s = argsort(zm_t[:,2],0)
    zm_t = take(zm_t,s)
    lst = [0]
    zm_t = zm_t[:,0,:]
    min1 = 2

    for x in range(zm_t.shape[0]):
        if(min1 < zm_t[x,2]):
            lst = x
            break

#coordination starts here
#linear coordinator has been used
#availability of synchronized phasor measurements

    U = U1
    U = U * pi/180
    p = 1
    area_recv = [1.0,1.0,1.0,1.0]
    sel_pmu = [30,33,68,80]
    x_lf = getdata('output1.txt')

    u = concatenate([U[0,0:2],Matrix([30*pi/180]),Matrix(U[0,2])],1)
    for pmno in range(N):
                    #print pmno,dict[pmno]
            u[0,pmno] = x_lf[0,sel_pmu[pmno]-1] -
x_[pmno][0,dict[pmno][sel_pmu[pmno]]-1]

    print "Coordination Vector is"
    print u*180/pi
    n = 118
    for i in range(N):
        x_[i][0,0:bus_area[i].shape[0]] = x_[i][0,0:bus_area[i].shape[0]] + u[0,i]
    x_con = Matrix([[0.0]*2*n])
    for m in range(n):
        for i in range(N):
            if(m+1 in area[i]):
```

```
            x_con[0,m] = x_[i][0,dict[i][m+1]-1]
            x_con[0,n+m] =  x_[i][0,dict[i][m+1]-1+bus_area[i].shape[0]]
err = x_con - x_lf
err = err / x_lf
putdata('error.txt',abs(err)*100)
putdata('consolidate.txt',x_con)
print str( time() - t1)+'\n'
recr = recr + 1
print recr
U1 = U
quit = 1
```

Appendix G

Parameters used in the implementation of the algorithms

Measurement error – Power flows – 5%
Measurement error – Power injections – 7%
Measurement error – Voltage Magnitudes – 1%

Modified coordination algorithm: - IEEE 118
Weights for area delayed by single step – 90%
Weights for area delayed by two steps – 75%

Probabilities of Delay of areas – IEEE 118
Area 1 90%
Area 2 90%
Area 3 95%
Area 4 80%

Justification for use of specific value is specified in chapter IV

REFERENCES

[1] Van Cutsem, Howard, J. L., Ribbens Pavela, M. and El-Fattah, Y. M., "A two level static state estimator for electric power systems," IEEE Transaction on PAS, Vol. PAS-100, No.8, Aug 1981.

[2] Van Cutsem, T; Ribbens-Pavella, M 'Hierarchical methods for state estimation of electric power systems.' IEEE TRANS. POWER APPAR. SYS. Vol. PAS-102, no. 10, pp. 3415-3424. 1983

[3] Van Cutsem and Ribbens Pavela, M., 'Critical survey of Hierarchical methods for state estimation of electric power systems,' IEEE Transaction on PAS, Vol. PAS-102, No.10, Oct 1983.

[4] M.S. Kurzyn "Real Time State estimation For lage scale power system" IEEE transaction on power apparatus and systems, vol PAS-102 No 7 July 1983.

[5] A. Monticelli, State Estimation in Electric Power Systems, Boston, Kluwer Academic Publishers, 1999.

[6] A. J. Wood, B. F. Wollenberg, Power Generation Operation & Control, New York, Wiley, 1984.

[7] A. Abur, A. G. Exposito, Power System State Estimation: Theory & Implementation, New York, Marcel Decker, 2004

[8] J. J. Grainger, W. D. Stevenson, Power System Analysis, McGraw Hill, 1994.

[9] A. Monticelli, "Electric Power System State Estimation", Proceedings of the IEEE, Vol. 88, No. 2, Feb. 2000 pp. 262-282.

[10] O. Alsac, N. Vempati, B. Stott, A. Monticelli, "Generalized State Estimation," IEEE transactions on Power Systems, Vol. 13, No. 3, Aug. 1998.

[11] Schweppe, F. C., Wildes, J., and Rom, D. B. 'Power system static state estimation I, II, III', IEEE Transaction on PAS, Vol. PAS-89, No.1, Jan 1970.

[12] Schweppe, F. C. , 'Recursive state estimation: Unknown but bounded errors and system inputs' IEEE transactions on Automatic control vol 13, Feb 1968

[13] AML da Silva, MB do Coatto Filho JF de Querioz, “State forecasting in Electric power systems” IEE proc-generation transmission and dist. 1983

[14] Lin, S. Y., and Lin, C. H., ‘An implementable distributed state estimator and distributed bad data processing schemes for electric power systems’, IEEE Transaction on Power Systems, Vol. 9, No.3, Aug 1994.

[15] Lin, S.Y., ‘A distributed state estimator for electric power systems’, IEEE Transactions for Power Systems, Vol. 7, No 2, May 1992.

[16] Barbosa, F. M., Carvalho, J. B., ‘Parallel and Distributed processing in state estimation of power system energy’.

[17] Barbosa, F. M., Carvalho, J. B., ‘Distributed processing in power system state estimation,’ 10th Mediterranean Electrotechnical Conference, MEleCon 2000, Vol III.

[18] Baldick, R., Ebrahimian, R., ‘State estimation Distributed Processing’, IEEE Transactions on Power Systems, Vol. 15, No. 4, Nov. 2000.

[19] P. Rousseaux , Van Cutsem and Ribbens Pavela, M.’Multi level dynamic state estimation for electric power systems’, 8 th power systems computation conf. Helsinki Finland 19-24 Aug 1984

[20] P. Rousseaux, D. Mallieu, Van Cutsem and Ribbens Pavela, M. “Dynamic state prediction and Hierarchical filtering for power systems state estimation” Automatica, IFAC vol 24 No 5 pp 595 – 618 1988

[21] K.L.Lo, M.M. Salem , R.D Mc COll, A.M.moffatt “Two-level state estimation for large power systems”. IEE proceedings vol 135, pt C no 4 July 1988

[22] I.O.Habiballah “ Modified two level state estimation approach”, IEE proc-generation transmission and dist. Vol 143 no 2 march 1996

[23] Bahgat, A. Sakr, M.M.F. El-Shafei, A.R. “Two level dynamic state estimator for electric power systems basedon nonlinear transformation” IEE proc-generation transmission and dist. Vol 136, no 1 Jan 1989

[24] A.K.Sinha and J.K. Mandal “Hierarchical dynamic state estimator using ANN-based dynamic load prediction” IEE proc. Generation, trans. And distri. Vol 146, No 6 Nov 1999

[25] Marsh, J.F. Azzam, M. “MCHSE: a versatile framework for the design of two-level powersystem state estimators” IEE proc. Generation, trans. And distri Volume: 135, Issue: 4, Jul 1988

[26] A. Abur, L. Zhao, "Multiarea State Estimation Using Synchronized Phasor Measurements,", IEEE Transactions of Power Systems, Vol. 20, No. 2, May 2005.

[27] A. Phadke, Y. Liu, J. D. La Ree, L, Mili, K. A. Rahman, 'Internet based Wide Area Information Sharing and its Role in Power System State Etimation.' IEEE Power Engineering Society Winter Meeting, Vol: 2, Feb 2001

[28] Chun-lien Su, Chan-Nan Lu "Interconnected network state estimation using randomly delayed measurement" IEEE transaction on Power systems vol 16 No 4 Nov 2001

[29] Gonzalez-Perez, C. Wollenberg, B.F. 'Analysis of massive measurement loss in large-scale power systemstate estimation' IEEE trans. on power systems, Vol 16, No 4, Nov 2001

[30] John P. Stovall*, Brendan J. Kirby* Thomas J. Overbye James S. Thorp, Arun G. Phadke, 'Issues Associated with the Development of a Wide-Area Analysis and Visualization Environment' Proceedings of the 39th Hawaii International Conference on System Sciences – 2006

[31] Arun G. Phadke "Synchronized Phasor Measurements A historical Review" IEEE/PES Transmission and Distribution Conference and Exhibition 2002: Asia Pacific. Volume: 1 Oct 2002

[32] Xu, B. Abur, A. 'Observability analysis and measurement placement for systems with PMUs' IEEE PES Power Systems Conference and Exposition, 2004. vol.2 Oct 2004

[33] R.F. Nuqui and A.G. Phadke 'Phasor Measurement Unit Placement based on Incomplete Observability' IEEE power engineering society summer meeting 2002, vol 2

[34] Jian Chen, Ali Abur 'Improved Bad Data processing via Strategic placement of PMUs' IEEE power engineering society general meeting Jun 2005

[35] Naduvathuparambil, B. Valenti, M.C. Feliachi, A. 'Communication delays in wide area measurement systems' System Theory, 2002. Proceedings of the Thirty-Fourth Southeastern Symposium

[36] Roy Moxely "Synchrophasors in Real World" Technical paper on PMU Schweitzer Engineering laboratory www.selinc.com

[37] Floyd Galvan, "The Eastern Interconnect Phasor Project" IEEE PES conference and exposition Dallas, Texas May 2006

[38] Mark J Rice and Gerald T. Heydt "Power System state estimatin accuracy enhancement through the use of PMU measurements" IEEE PES conference and exposition Dallas, Texas May 2006

[38] Hongxia Wu and Jay Giri “PMU Impact on State estimation Reliabilityfor Improved Grid Security” IEEE PES conference and exposition Dallas, Texas May 20069

[39] Ali Abur “Distributed State estimation for mega grids” 15th PSCC Liege 22-26 Aug 2006

[41] http://www.ee.washington.edu/research/pstca/pf118/pg_tca118bus.htm

[42] http://pti-us.com/pti/company/brochures/SS274_PSSE30.pdf