Detecting Circuit Breaker Status Errors in Substations

Part II of Final Project Report

Power Systems Engineering Research Center
A National Science Foundation Industry/University Cooperative Research Center since 1996
Detecting Circuit Breaker Status Errors in Substations

Final Project Report

Part II

Report Authors

Jun Zhu
Ali Abur
Texas A&M University

PSERC Publication 05-57

October 2005
Information about this project

For information about this project contact:

Ali Abur, Ph.D.
Texas A&M University
Department of Electrical Engineering
College Station, TX 77843-3128
Phone: 979-845-1493
FAX: 979-845-9887
Email: abur@ee.tamu.edu

Power Systems Engineering Research Center

This is a project report from the Power Systems Engineering Research Center (PSERC). PSERC is a multi-university Center conducting research on challenges facing a restructuring electric power industry and educating the next generation of power engineers. More information about PSERC can be found at the Center’s website:

For additional information, contact:

Power Systems Engineering Research Center
Cornell University
428 Phillips Hall
Ithaca, New York 14853
Phone: 607-255-5601
Fax: 607-255-8871

Notice Concerning Copyright Material

PSERC members are given permission to copy without fee all or part of this publication for internal use if appropriate attribution is given to this document as the source material. This report is available for downloading from the PSERC website.

©2005 Texas A&M University. All rights reserved.
Acknowledgements

The Power Systems Engineering Research Center sponsored the research project titled “Enhanced Reliability of Power System Operation Using Advanced Algorithms and IEDs for On-Line Monitoring” (PSERC project T-17). This is Part II of the final report on that project.

We express our appreciation for the support provided by PSERC’s industrial members and by the National Science Foundation under grant NSF EEC-0002917 received under the Industry / University Cooperative Research Center program.

The authors thank PSERC members for their technical advice on the project, especially M. Ingram (TVA), D. Sevcik (CenterPoint Energy), M. Subramanian (ABB), D. Krummen (AEP), A. Mander (Tri-State G&T Association), L. Priez (Entergy), J. Bell (TXU), and P. Kaptain (WAPA) who are industry advisors for the project. The authors also acknowledge Professors A. P. Sakis Meliopoulos of Georgia Tech and Rahmat Shoureshi of Colorado School of Mines who provided technical advice to this work. Professor Kezunovic is the project leader.
Executive Summary

State estimation is a crucial function for reliable and secure operation of power systems, and for efficient power market operations. Successful use of state estimators requires minimization of possible errors that would lead to inaccurate estimates of the power system condition and, as a result, to inaccurate information for power system operators. Circuit breaker status errors can produce such errors because the state estimator would be estimating the system condition for a power system whose topology is not the same as the actual system topology.

Recently a new method, based on a reduced system model and Lagrange multipliers, was proposed for topology error analysis. In this method, the size of a detailed substation model is reduced by applying Kirchhoff’s law and by implicitly considering topological constraints. This model reduction is achieved without losing any capability to detect and identify topology errors. The method has an important advantage over existing topological analysis techniques in that the user does not have to specify the suspect substation ahead of time because all substations can be modeled by using a small number of extra state variables.

Only simulation results for one substation have been shown in the literature for this new method. Extensive testing on practical test systems has not yet been reported. This project investigates the method and its implementation to evaluate its possible performance for practical, multi-substation power systems. The method is implemented and tested on the IEEE 14, 30 and 57 bus test systems using simulated data and topology error scenarios.

The main two purported advantages of the implemented method are its ability to detect status errors associated with substation breakers without significantly increasing the size of the network model, and to differentiate between analog measurement errors and breaker status errors. Both advantages were validated using simulated cases. As a result, the method is recommended as a new feature for state estimation software.
# Table of Contents

1. Introduction ................................................................. 1
2. Method of Implicit Model ................................................. 2
   2.1 Building the Tree .................................................. 2
   2.2 Computing Lagrange Multipliers ............................... 3
   2.3 Normalization of Lagrange Multipliers ...................... 6
3. Topology Error Identification Algorithm ...................... 7
   3.1 Overall Process .................................................. 7
   3.2 Flow Chart ....................................................... 7
4. Simulation Results ...................................................... 10
   4.1 Topology or Measurement Error ............................... 10
   4.2 Simultaneously Occurring Measurement and Topology Errors .............................. 12
   4.3 Computation Time .............................................. 13
5. Program Data Structure .............................................. 14
   5.1 Input Data Files ................................................. 14
      5.1.1 topoaninput.dat ........................................... 14
      5.1.2 topomeasure.dat .......................................... 15
      5.1.3 vinput.dat .................................................. 17
   5.2 Output Data Files ............................................... 17
6. Conclusions and Future Work ...................................... 20
REFERENCES ............................................................... 20
APPENDIX 1. Matlab Code of the Method ............................ 22
APPENDIX 2. Input Data Files Generating Program ................ 35
# Table of Figures

| Figure 1.1 | Substation graph | .............................................................................. | 2 |
| Figure 1.2 | Proper tree | .............................................................................. | 2 |
| Figure 3.1 | Flow chart | .............................................................................. | 8 |
| Figure 4.1 | 4-CB substation model | .............................................................................. | 12 |
| Figure 8.1 | One-line substation model | .............................................................................. | 35 |
| Figure 8.2 | Two-line substation model | .............................................................................. | 36 |
| Figure 8.3 | Three-line substation model | .............................................................................. | 36 |
| Figure 8.4 | Four-line substation model | .............................................................................. | 37 |
| Figure 8.5 | Five-line substation model | .............................................................................. | 38 |
| Figure 8.6 | Six-line substation model | .............................................................................. | 39 |
| Figure 8.7 | Seven-line substation model | .............................................................................. | 40 |
Table of Tables

Table 4.1 Simulated Topology and Measurement Errors ..................................................10
Table 4.2 Results of Error Identification - 14-bus System..............................................11
Table 4.3 Results of Error Identification - 30-bus System..............................................11
Table 4.4 Results of Error Identification - 57-bus System..............................................11
Table 4.5 Multiple Error Identification Results - 14-bus System .................................13
Table 4.6 Comparison of Computation Time.................................................................13
1. Introduction

Topology error analysis is one of the important functions which assure the validity of the data used in power system state estimation. Normally, topology of the power system refers to the topology between and within the substations in the system, that is, the status of the circuit breakers in the substation. Traditional state estimation is carried out assuming that the status of the circuit breakers are all known, then all the inconsistencies resulted will be blamed on the analog measurements. However, the wrong status of a circuit breaker, which will typically lead to the outage of a line, will cause much more severe errors in the state estimation results.

Considering computational limits, detailed substation models including circuit breakers are only involved for small sub networks surrounding the suspect area in traditional topology analysis method. As a result, the suspected area has to be determined a priori state estimation by other techniques [1,2,3,4,5].

Recently, a new method based on reduced system model and Lagrange multipliers is proposed [6]. In this method, the size of the detailed substation model is reduced by applying Kirchhoff’s law and implicitly considering the topological constraints. This is achieved without losing any capability to detect and identify topology errors. However, only the simulation results within one substation are shown in that paper.

The objective of this report is to testify the validity of the topology analysis method stated above in a realistic power system. The report is organized as follows. Section II briefly presents the theory and formulation of the method. Section III gives the flow chart of the method. The simulation results are given in section IV. The input and output data structure is introduced in section V. Section VI concludes the report, followed by the Appendix.
2. Method of Implicit Model

In this section, a brief presentation of the implicit model method will be given. It is composed of three parts: (1) how to build the tree of the substation, (2) how to compute the Lagrange multipliers, and (3) how to normalize them.

2.1 Building the Tree

In order to simplify the detailed substation model, a tree is built. This way, any of the power flows and voltage drops in the substation graph could be expressed in terms of the power flows of the tree branches and the voltage drops across the tree links, respectively. Moreover, a proper tree is a useful structure to determine the constraints of the circuit breakers.

The proper tree is defined as follows.
1) Exclude all nonzero impedance branches.
2) Include as many closed CBs as possible.
3) Select, for every electrical bus, a nonzero injection node that will be termed the base node. Include the respective injection branch in the tree. An electrical bus exclusively connected to nonzero impedance branches constitutes a special case which is handled by adding a virtual null-injection branch to any of its elemental buses. This extra branch, also included in the tree, irresponsible for the null-injection constraints conventionally used by SEs.
4) If the case arises, complete the tree with CBs the status of which is unknown.
5) Exclude as many open CBs as possible.

Figure 1.2 shows the proper tree of the substation of Figure 1.1.
This is the rule of tree building of a substation. For a whole system, the non-zero impedance branches will be connected to each other and there will be a common reference picked for all the injections.

### 2.2 Computing Lagrange Multipliers

From the proper tree built, the proper model can be written as follows:

\[
\begin{align*}
    z &= h(x_I, x_{CB}) + e \\
    c(x_I, x_{CB}) &= 0 \\
    x_{CB} &= 0
\end{align*}
\]

(2.1) (2.2) (2.3)

where the following notation is adopted:

- \(x_I\): Component of the state vector containing:
  - Voltage magnitude and phase angle of all base nodes (each one representing an electrical node).
  - Voltage magnitude and phase angle across unknown and open CBs belonging to the proper tree.
  - Power flows through links of the proper tree corresponding to injections, as well as unknown and closed CBs.

- \(x_{CB}\): Component of the state vector comprising:
  - Voltage magnitude and phase angle across closed CBs included in the proper tree.
  - Power flows through open CBs excluded from the proper tree.

For example, for substation shown in Fig. 1:

\[
\begin{align*}
    x_I &= [\theta_1, p_2]^T \\
    x_{CB} &= [\theta_{23}, \theta_{23}, \theta_{14}, p_{12}, p_{24}]^T
\end{align*}
\]

For equation (2.1), the measurement vector \(z\) is expressed as a sum of \(h(x_I, x_{CB})\) and the vector \(e\). \(h(x_I, x_{CB})\) is the nonlinear function of both \(x_I\) and \(x_{CB}\), which are the state vectors, and \(e\) is the vector of measurement errors. Equation (2.2) gives the equality constraints contributed by zero-injections on buses. Topology errors are normally assumed to be zero by state estimator. Therefore, for error free operation, the equality constraints of CBs can be written as shown in equation (2.3). For closed CBs in the tree and open CBs in the co-tree, the constraint equations can be written as \(\theta_{ab} = 0\) and \(p_{cd} = 0\), respectively. As mentioned in the previous section, a proper tree will make equation (2.3) as simple as possible.
The weighted least square state estimation problem in the presence of network topology errors and equality constraints can then be formulated as the following optimization problem:

\[
\text{Minimize} \quad J(x_i, x_{cb}) = \frac{1}{2} r' W r \\
\text{Subject to} \quad c(x_i, x_{cb}) = 0, \quad x_{cb} = 0 
\]  

(2.4)

where:

\( r = z - h(x_i, x_{cb}) \) is the measurement residual vector,

\( W \) is the diagonal matrix whose inverse is the measurement error covariance matrix, \( \text{cov}(e) \).

Applying the method of Lagrange multipliers, the Lagrangian of proper model becomes

\[
L = \frac{1}{2} r' W r - \mu ' c(x_i, x_{cb}) - \lambda ' x_{cb} 
\]  

(2.5)

and the following first-order optimality conditions can be written as

\[
\frac{\partial L}{\partial x_i} = H_i' W r + C_i' \mu = 0 
\]  

(2.6)

\[
\frac{\partial L}{\partial x_{cb}} = H_{cb}' W r + C_{cb}' \mu + \lambda = 0 
\]  

(2.7)

\[
\frac{\partial L}{\partial \mu} = c(x_i, x_{cb}) = 0 
\]  

(2.8)

\[
\frac{\partial L}{\partial \lambda} = x_{cb} = 0 
\]  

(2.9)

where:

\( H_i, H_{cb} \) and \( C_i, C_{cb} \) are the gradients of \( H \) and \( C \) with respect to \( x_i \) and \( x_{cb} \), respectively.

\( \mu \) and \( \lambda \) are the Lagrange multipliers for the equality constraints (2.1) and (2.2).

Equation (2.7) allows the multiplier \( \lambda \) to be expressed in terms of \( r \) and \( \mu \).

\[
\lambda = T \begin{bmatrix} r \\ \mu \end{bmatrix} 
\]  

(2.10)
where
\[
T = - \left[ \begin{array}{c} WH_{CB} \\ C_{CB} \end{array} \right]^t
\]  

(2.11)

is the topological sensitivity matrix.

Equation (2.3) allows the substitution of \( x_{cb} \) in equations (2.6)-(2.8). Denoting \( h(x_i,0) \) and \( c(x_i,0) \) by \( h_0(x_i) \), \( c_0(x_i) \) respectively, the proper model becomes the implicit model
\[
z = h_0(x_i) + e  
\]  

(2.12)
\[
c_0(x_i) = 0  
\]  

(2.13)

Note that equation (2.12) and (2.13) provides exactly the same results as the original model without explicitly handling topological constraints.

Substituting the first order Taylor approximations for \( h_0(x_i) \) and \( c_0(x_i) \), the following linear equations will be obtained:
\[
H_i \cdot \Delta x_i + r = \Delta z  
\]  

(2.14)
\[
C_i \cdot \Delta x_i = -c_0(x_{i0})  
\]  

(2.15)

where:
\[
\Delta x_i = x_i - x_{i0},  
\]
\[
\Delta z = z - h_0(x_{i0}).  
\]

\( x_{i0} \) being the initial guess for the system state vector.

Using (2.6), (2.14), and (2.15), the following equation will be obtained:
\[
\begin{bmatrix}
0 & H_i^tW & C_i^t \\
H_i & I & 0 \\
C_i & 0 & 0
\end{bmatrix}
\begin{bmatrix}
\Delta x_i \\
r \\
\mu
\end{bmatrix}
= 
\begin{bmatrix}
0 \\
\Delta z \\
-c_0(x_{i0})
\end{bmatrix}
\]  

(2.16)

Equation (2.16) is similar to the equation used in conventional WLS state estimation except for the fact that state variables are not exactly the same. The measurement residuals \( r \) and the Lagrange multipliers for the zero injections \( \mu \) can be obtained first by iteratively solving (2.16). Once the state estimation algorithm successfully converges, (2.10) can be used to recover the Lagrange multiplier vector associated with the topology errors.
2.3 Normalization of Lagrange Multipliers

To identify topology errors, the validity of the constraints (8) has to be tested. This can be done by checking the Lagrange multiplier vector $\lambda$ associated with the topology error vector $x_{CB}$. However, it has to be normalized by its covariance matrix $\text{cov}(\lambda)$ before testing of its significance.

Letting $u = [r \ \mu]^{T}$ and using (2.10):

$$\Lambda = \text{cov}(\lambda) = T \cdot \text{cov}(u) \cdot T^{t}$$  \hspace{1cm} (2.17)

The covariance of $u$, $\text{cov}(u)$ can be calculated by first expressing $r$ and $\mu$ in terms of the measurement mismatch. To do that, let the inverse of the coefficient matrix in (2.16) be given in partitioned form as follows:

$$\begin{bmatrix}
0 & H_{i}W & C_{i}^{t} \\
H_{i} & I & 0 \\
C_{i} & 0 & 0
\end{bmatrix}^{-1} = \begin{bmatrix}
E_{1} & E_{2} & E_{3} \\
E_{4} & E_{5} & E_{6} \\
E_{7} & E_{8} & E_{9}
\end{bmatrix}$$  \hspace{1cm} (2.18)

Noting that $c_{0}(x) = 0$ at the solution, (2.16) will yield the following expressions for $r$ and $\mu$:

$$r = E_{5} \cdot \Delta z$$  \hspace{1cm} (2.19)
$$\mu = E_{6} \cdot \Delta z$$  \hspace{1cm} (2.20)

Let $\Psi = [E_{5} \ E_{8}]^{T}$, then:

$$u = \Psi \cdot \Delta z$$  \hspace{1cm} (2.21)
$$\text{cov}(u) = \Psi^{-1} \cdot W \cdot \Psi$$  \hspace{1cm} (2.22)

The Lagrange multipliers for the topology errors can then be normalized using the diagonal elements of the covariance matrix $\Lambda$ defined in (2.19):

$$\lambda_{i}^{N} = \frac{\lambda_{i}}{\sqrt{\Lambda(i,i)}}$$  \hspace{1cm} (2.23)

for all $i = 1 \ldots k$, where $k$ is the total number of CBs whose errors are to be identified.
3. Topology Error Identification Algorithm

3.1 Overall Process

Using the method presented in section 2, a topology error detection and identification can be carried out as described below:

**Step 1. Topology Processing**
First a proper tree is selected for each substation. Based on the tree, the state variables are categorized into two groups, $x_i$ and $x_{CB}$. Then the Jacobian matrix $H$ is partitioned into two sub matrices, $H_i$ and $H_{CB}$. This step constitutes the backbone of the algorithm.

**Step 2. State Estimation**
Except for the size of the state and measurement vectors, this procedure is essentially identical to traditional state estimation carried out by existing methods. The solution will yield both the measurement residual vector $r$ and the Lagrange multiplier vector $\mu$ of zero injections if they are treated as equality constraints in the state estimation formulation.

**Step 3. Normalization of $r^N$ and $\lambda^N$**
Compute the normalized residuals $r^N$ for the measurements and the normalized Lagrange multipliers $\lambda^N$ for the topology errors using the procedure outlined in section II above.

**Step 4. Bad Data and Topology Error Identification**
Choose the larger one between the largest normalized residual and the largest normalized Lagrange multiplier.

- If the chosen value is below the identification threshold, then no bad data or parameter error will be suspected. A statistically reasonable threshold to use is 3.0, which is the one used in all simulations presented in the next section.
- Else, the measurement or the parameter corresponding to the chosen largest value will be identified as the source of error in estimation.

Note that the bad data and topology error analysis are conducted simultaneously. That means this algorithm can identify the topology error even with the interference of the bad data, and clear them one by one. Furthermore, only a small subset of substations should be suspicious in practice, as topology errors are not frequent. In this algorithm, the detailed substation data of those suspicious substations can be combined with the normal bus data. Since the entire substation blocks are decoupled in T matrix, the computational burden will be largely decreased this way.

3.2 Flow Chart

The flow chart of the algorithm is shown below.
1. Build proper

2. Build Jacobian matrix

3. Build C matrix for zero injection constraints

3. Divide the Jacobian matrices into two parts: $H_i$ and $H_{CB}$, $C_i$ and $C_{CB}$

4. Compute $\Delta z$ and $r$ by eqn. (2.16)

5. $\text{max}(\Delta z) < \text{threshold}$?

5. Y

6. Compute $\lambda$ by eqn. (2.10)

7. Normalize $\lambda$ and $r$

7. continue

Update the state variables

Figure 3.1 Flow Chart
Figure 3.1 Flow Chart (continued)

```
8. max(\(\lambda\)) < max(\(r\))?

Y

N

8. max(\(\lambda\)) < 3.0?

N

Y

Topological Error

No Error

Measurement Error

continue
```
4. Simulation Results

The above described topology error identification procedure is implemented and tested on IEEE 14, 30 and 57 bus test systems. Different cases are simulated where errors are introduced in the circuit breakers and analog measurements. Both single errors and simultaneously occurring errors in analog measurements and circuit breaker status are simulated.

4.1 Topology or Measurement Error

This case presents single errors in circuit breaker status or analog measurement. The method is shown to differentiate between these different types of errors and to correctly identify the error. The simulated errors for the three test systems are listed in Table 1, where the tests A and B are carried out as follows:

Test A: The status of the circuit breaker listed in Table 1 is wrong; all analog measurements are error free. There is a disconnected line as a result of that circuit breaker’s operation.

Test B: No topology errors are introduced; all measurements are error free except for the listed flows in Table 4.1. Tables 4.2-4.4 show the sorted normalized residuals $r_N$ and normalized Lagrange multipliers $\lambda_N$, obtained during the tests of Table 1, only the 5 largest ones are listed.

<table>
<thead>
<tr>
<th>Test System</th>
<th>Wrong status CB &amp; Outaged Line/Meas.</th>
<th>Test A</th>
<th>Test B</th>
</tr>
</thead>
<tbody>
<tr>
<td>14-bus</td>
<td>$CB_{34-32}$</td>
<td>Line 5-6</td>
<td>$p_{34-32}$</td>
</tr>
<tr>
<td>30-bus</td>
<td>$CB_{74-68}$</td>
<td>Line 10-22</td>
<td>$p_{74-68}$</td>
</tr>
<tr>
<td>57-bus</td>
<td>$CB_{89-78}$</td>
<td>Line 4-6</td>
<td>$p_{89-78}$</td>
</tr>
</tbody>
</table>
### Table 4.2 Results of Error Identification - 14-bus System

<table>
<thead>
<tr>
<th>Measurement/Topology</th>
<th>Test A</th>
<th>Test B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CB_{34-32}$</td>
<td>1008.06</td>
<td>$P_{34-32}$</td>
</tr>
<tr>
<td>$CB_{36-32}$</td>
<td>1008.06</td>
<td>$CB_{33-34}$</td>
</tr>
<tr>
<td>$CB_{5-33}$</td>
<td>944.46</td>
<td>$CB_{36-32}$</td>
</tr>
<tr>
<td>$CB_{5-35}$</td>
<td>944.46</td>
<td>$CB_{34-32}$</td>
</tr>
<tr>
<td>$CB_{33-34}$</td>
<td>942.43</td>
<td>$CB_{35-36}$</td>
</tr>
</tbody>
</table>

### Table 4.3 Results of Error Identification - 30-bus System

<table>
<thead>
<tr>
<th>Measurement/Topology</th>
<th>Test A</th>
<th>Test B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CB_{74-68}$</td>
<td>212.30</td>
<td>$P_{74-68}$</td>
</tr>
<tr>
<td>$CB_{70-68}$</td>
<td>199.09</td>
<td>$CB_{74-68}$</td>
</tr>
<tr>
<td>$CB_{10-71}$</td>
<td>160.95</td>
<td>$CB_{73-74}$</td>
</tr>
<tr>
<td>$CB_{105-104}$</td>
<td>157.04</td>
<td>$CB_{36-34}$</td>
</tr>
<tr>
<td>$CB_{106-104}$</td>
<td>157.04</td>
<td>$CB_{33-31}$</td>
</tr>
</tbody>
</table>

### Table 4.4 Results of Error Identification - 57-bus System

<table>
<thead>
<tr>
<th>Measurement/Topology</th>
<th>Test A</th>
<th>Test B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CB_{80-78}$</td>
<td>371.63</td>
<td>$P_{80-78}$</td>
</tr>
<tr>
<td>$CB_{82-78}$</td>
<td>371.63</td>
<td>$P_{128-124}$</td>
</tr>
<tr>
<td>$CB_{81-82}$</td>
<td>282.65</td>
<td>$CB_{80-78}$</td>
</tr>
<tr>
<td>$CB_{76-75}$</td>
<td>212.03</td>
<td>$CB_{82-78}$</td>
</tr>
<tr>
<td>$CB_{77-75}$</td>
<td>212.03</td>
<td>$CB_{79-80}$</td>
</tr>
</tbody>
</table>
Note that in Tables 4.2 and 4.4, there are two equal largest normalized Lagrange multipliers. That is because these two circuit breakers form a critical pair like the circuit breakers 2-3 and 2-4 shown in Figure 4.1. By changing the status from closed to open, either CB 2-3 or 2-4 can disconnect the line at bus 4. Such errors can be detected but not identified. By the simulation results shown above, single circuit breaker status error or single measurement error can be detected and identified based on the specific topology of that substation.

4.2 Simultaneously Occurring Measurement and Topology Errors

This case shows the identification of multiple errors occurring simultaneously in the 14-bus system. Errors are simulated for the status of CB 31-26, and for the power flow measurement p28-26 in substation 4. Note that the two errors are highly correlated. Two tests are carried out as follows:

Test A: The bias in the measurement is 50% of the true value.

Test B: The bias in the measurement is 100% of the true value.

Table 4.5 shows the sorted normalized residuals $r_N$ and normalized Lagrange multipliers $\lambda_N$ of the two tests, only the 5 largest ones are listed.

When there are multiple errors in the CB status as well as the measurement, the program can identify the most influential error among all the errors as shown in Table 4.5. Similar to the case of the multiple interacting and conforming bad data, strongly interacting topology and analog measurement errors may not be identified due to error masking. Such cases are however rare and cannot be handled by this method.
Table 4.5 Multiple Error Identification Results - 14-bus System

<table>
<thead>
<tr>
<th>Test A</th>
<th>Test B</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Measurement/Topology</strong></td>
<td><strong>Normalized residual (\lambda^N)</strong></td>
</tr>
<tr>
<td>(CB_{31-26})</td>
<td>278.72</td>
</tr>
<tr>
<td>(P_{31-50})</td>
<td>174.00</td>
</tr>
<tr>
<td>(P_{31-26})</td>
<td>174.00</td>
</tr>
<tr>
<td>(CB_{44-42})</td>
<td>144.64</td>
</tr>
<tr>
<td>(CB_{45-42})</td>
<td>212.03</td>
</tr>
</tbody>
</table>

4.3 Computation Time

The number of state variables of this implicit topology error identification method is less than the state variables used in traditional state estimation, since it is based only on the real power flow measurements. The computation times for the topology error analysis program and the traditional state estimation program are given in Table 4.6 for the test systems. Note that the topology error analysis method uses the detailed substation topology data for all the buses.

Table 4.6 Comparison of Computation time

<table>
<thead>
<tr>
<th>Test System</th>
<th>Computation Time (Sec.)</th>
<th>Topology Error Analysis</th>
<th>State Estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>14-bus</td>
<td>0.8750</td>
<td>0.5780</td>
<td></td>
</tr>
<tr>
<td>30-bus</td>
<td>1.1560</td>
<td>0.8120</td>
<td></td>
</tr>
<tr>
<td>57-bus</td>
<td>4.1250</td>
<td>2.682</td>
<td></td>
</tr>
</tbody>
</table>

For all three systems, the computation time of topology error analysis program is almost 150% of the traditional state estimation. Considering the complexity of the substation level system, this result is acceptable.
5. Program Data Structure

The input data files for the topology analysis program include the detailed circuit breaker connection information and the power flow measurements through the circuit breakers. The output data files give the normalized Lagrange multipliers of CBs constraints and the normalized residuals of measurements.

5.1 Input Data Files

The input data of the topology analysis is composed of three files: (1) the detailed circuit breaker topology data, (2) power flow measurement data, and (3) voltage data.

5.1.1 topoanainput.dat

1 4 6 1 15 16 17
2 6 10 2 18 19 20 21 22
3 4 6 3 23 24 25
4 7 13 4 26 27 28 29 30 31
5 6 10 5 32 33 34 35 36
6 6 10 6 37 38 39 40 41
7 5 8 7 42 43 44 45
8 1 1 8
9 6 10 9 46 47 48 49 50
10 4 6 10 51 52 53
11 4 6 11 54 55 56
12 4 6 12 57 58 59
13 5 8 13 60 61 62 63
14 4 6 14 64 65 66

-99

1 0 -1 0 1.000 1 2 16 19 0.01938 0.05917 0.00000 0.05280
2 0 -1 0 1.000 1 5 17 33 0.05403 0.22304 0.00000 0.04920
3 0 -1 0 1.000 2 5 20 34 0.05695 0.17388 0.00000 0.03400
4 0 -1 0 1.000 2 4 21 27 0.05811 0.17632 0.00000 0.03740
5 0 -1 0 1.000 2 3 22 24 0.04699 0.19797 0.00000 0.03480
6 0 -1 0 1.000 3 4 25 28 0.06701 0.17103 0.00000 0.03460
7 0 -1 0 1.000 4 5 29 35 0.01335 0.04211 0.00000 0.01280
8 0 -1 0 1.000 7 8 43 8 0.00000 0.17615 0.00000 0.00000
9 0 -1 0 1.000 7 9 44 47 0.00000 0.11001 0.00000 0.00000
10 0 -1 0 1.000 9 10 48 52 0.03181 0.08450 0.00000 0.00000
11 0 -1 0 1.000 9 14 49 65 0.12711 0.27038 0.00000 0.00000
12 0 -1 0 1.000 6 11 38 55 0.09498 0.19890 0.00000 0.00000
13 0 -1 0 1.000 6 12 39 58 0.12291 0.25581 0.00000 0.00000
14 0 -1 0 1.000 6 13 40 61 0.06615 0.13027 0.00000 0.00000
15 0 -1 0 1.000 12 13 59 62 0.22092 0.19988 0.00000 0.00000
16 0 -1 0 1.000 13 14 63 66 0.17093 0.34802 0.00000 0.00000
17 0 -1 1 0.930 6 5 41 36 0.00000 0.25202 0.00000 0.00000
18 0 -1 1 0.960 9 4 50 30 0.00000 0.55618 0.00000 0.00000
19 0 -1 1 0.970 4 7 31 45 0.00000 0.20912 0.00000 0.00000
20 0 -1 0 1.000 10 11 53 56 0.08205 0.19207 0.00000 0.00000
21 1 1 -1 -1 -1 -1 1 16 0.00000 0.00000 0.00000 0.00000
22 1 1 -1 -1 -1 -1 1 16 0.00000 0.00000 0.00000 0.00000
23 1 1 -1 -1 -1 -1 1 17 0.00000 0.00000 0.00000 0.00000
24 1 1 -1 -1 -1 -1 1 17 0.00000 0.00000 0.00000 0.00000
25 2 1 -1 -1 -1 -1 2 19 0.00000 0.00000 0.00000 0.00000
26 2 0 -1 -1 -1 -1 1 19 20 0.00000 0.00000 0.00000 0.00000
27 2 1 -1 -1 -1 -1 20 18 0.00000 0.00000 0.00000 0.00000
28 2 1 -1 -1 -1 -1 2 21 0.00000 0.00000 0.00000 0.00000
29 2 1 -1 -1 -1 -1 21 22 0.00000 0.00000 0.00000 0.00000
30 2 1 -1 -1 -1 -1 22 18 0.00000 0.00000 0.00000 0.00000

-99
The file shown above is part of the topoanainput.dat file of 14 bus system. This input file includes the detailed circuit breaker topology information. –99 is used to flag of the end of data. It is composed of two parts, the substation information data and the circuit breaker topology of the system.

**Substation Information Data**
1st column: substation number.
2nd column: total number of electrical nodes included in that substation.
3rd column: total number of branches belong to that substation. It includes all the circuit breakers and the non-zero impedance branches connected to this station.
4th column to the end: the number of columns after the 3rd column differs for each substation, it shows the detailed electrical node number of that substation.

**Circuit Breaker Topology Data**
1st column: the branch number.
2nd column: the substation number of that line. For non-zero impedance branch, the value is 0. For circuit breakers, the value is the number of the substation it belongs to.
3rd column: circuit breaker status. For non-zero impedance branch, the value is -1. For open circuit breakers, the value is 0. For closed circuit breakers, the value is 1.
4th column: the line type. For a transformer, the value is 1 and for a normal transmission line, the value is 0. For circuit breakers, the value is –1.
5th column: tap value. For a transformer, it gives the tap value of that transformer. For normal transmission line it is 1. For circuit breaker it is –1.
6th to 7th columns: the start and end substation number the non-zero impedance branch connected. For circuit breakers, the value is –1.
8th to 9th columns: the detailed electrical nodes of a branch, for both the non-zero impedance branch and circuit breaker.
10th to 13th columns: the resistance, reactance, conductance and susceptance of the non-zero impedance branch. For circuit breakers, the values will be all zeros.

<table>
<thead>
<tr>
<th>topomeasure.dat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 16 19 1.572330 0.000001</td>
</tr>
<tr>
<td>1 1 2 17 33 0.754130 0.000001</td>
</tr>
<tr>
<td>1 1 21 1 16 2.326470 0.000001</td>
</tr>
<tr>
<td>1 1 22 16 15 0.754140 0.000001</td>
</tr>
<tr>
<td>1 1 24 17 15 -0.754130 0.000001</td>
</tr>
<tr>
<td>1 2 5 22 24 0.736110 0.000001</td>
</tr>
<tr>
<td>1 2 1 19 16 -1.529160 0.000001</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

The list shown above is part of the topomeasure.dat file for the 14 bus system. Since the topology analysis method only uses the real power flow and injection data, both directions of power flows of a non-zero impedance branch and all of the bus real power injections are needed in order to give enough redundancy. It is composed of two parts.

**Power Flow Measurements**

1st column: the flag of the measurement. 1 represents it is a real power flow measurement.

2nd column: the station number that power flow belongs to, whether it is a non-zero impedance branch connected to that station or it is a circuit breaker in that station.

3rd column: the branch number of that power flow.

4th to 5th columns: the start and end node number of that power flow.

6th column: the power flow measurement.

7th column: the weight of that measurement.
**Power Injection Measurements**

1st column: the flag of the measurement. 3 represents it is a real power injection measurement.

2nd column: the substation number of that power injection belongs to.

3rd column: the node number of that power injection.

4th column: the power injection measurement.

5th column: the weight of that measurement.

5.1.3 `vinput.dat`

<table>
<thead>
<tr>
<th>Substation</th>
<th>Power Injection</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0600</td>
</tr>
<tr>
<td>2</td>
<td>1.0450</td>
</tr>
<tr>
<td>3</td>
<td>1.0100</td>
</tr>
<tr>
<td>4</td>
<td>1.0137</td>
</tr>
<tr>
<td>5</td>
<td>1.0211</td>
</tr>
<tr>
<td>6</td>
<td>1.0269</td>
</tr>
<tr>
<td>7</td>
<td>1.0109</td>
</tr>
<tr>
<td>8</td>
<td>1.0512</td>
</tr>
<tr>
<td>9</td>
<td>0.9860</td>
</tr>
<tr>
<td>10</td>
<td>0.9753</td>
</tr>
<tr>
<td>11</td>
<td>0.9687</td>
</tr>
<tr>
<td>12</td>
<td>1.0096</td>
</tr>
<tr>
<td>13</td>
<td>1.0023</td>
</tr>
<tr>
<td>14</td>
<td>0.9739</td>
</tr>
<tr>
<td>-99</td>
<td></td>
</tr>
</tbody>
</table>

This file contains the state estimation results for all the bus voltage magnitudes. They are supplied to be used in the topology analysis program since the node voltages will not be included in the state variables.

1st column: the substation number.

2nd column: the magnitude of the voltage.

5.2 **Output Data Files**

There is only one output file of the topology analysis program. A sample is shown below.

**BADDATAOTP.dat**

The maximum normalized lagrange multiplier

| 55 | 54 | 221.6456 |

Normalized lagrange multiplier

<p>| 1  | 16 | 19.2210 |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>15</td>
<td>11.4732</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>15</td>
<td>11.4732</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>0.5939</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>19</td>
<td>19.8892</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>18</td>
<td>11.0035</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>19.8892</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>18</td>
<td>11.0035</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>15.7338</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>13.7105</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>24</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>23</td>
<td>11.9863</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>23</td>
<td>11.9863</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>6.6629</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>27</td>
<td>91.8836</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>26</td>
<td>99.9550</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>28</td>
<td>94.9610</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>29</td>
<td>91.8836</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>26</td>
<td>72.7250</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>26</td>
<td>93.7364</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>30</td>
<td>72.2868</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>31</td>
<td>72.0833</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>33</td>
<td>10.3829</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>32</td>
<td>16.4818</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>35</td>
<td>10.3829</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>32</td>
<td>16.4818</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>36</td>
<td>78.7925</td>
<td></td>
</tr>
</tbody>
</table>

---

Normalized residual

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>1</td>
<td>19</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>27</td>
</tr>
<tr>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>1</td>
<td>30</td>
</tr>
</tbody>
</table>
This file gives the normalized Lagrange multipliers of the CB constraints and normalized residuals of the measurements.

**The Maximum Normalized Lagrange Multiplier**
1st to 2nd columns: the start and end node number of the circuit breaker.
3rd column: the normalized Lagrange multiplier of that CB constraint, which is the largest one of all.

**Normalized Lagrange Multiplier**
1st to 2nd columns: the start and end node number of the circuit breaker.
3rd column: the normalized Lagrange multiplier of that CB constraint.

**Normalized Residual**
1st column: the flag of the measurement. 1 represents it is a real power flow measurement. 3 represents it is a real power injection measurement.
For real power flow measurement
2nd to 3rd columns: the start and end node number of that power flow measurement.
4th column: normalized residual of that measurement.

**For Real Power Injection Measurement**
2nd column: the connected node number of that power injection measurement.
3rd column: normalized residual of that measurement.
6. Conclusions and Future Work

This project demonstrates that the presented topology error analysis method provides a viable and efficient solution for large power systems. This method can identify network topology error even in the presence of bad measurements. The topology error identification is accomplished by formulating the circuit breaker variables as zero equality constraints and then testing the significance of the associated Lagrange multipliers. These are computed from the normalized measurement residuals obtained by the process similar to the traditional state estimation. The method provides enough flexibility treating different scales of substation level topology. Only one substation or all substations can be modeled in detail at the substation level without significant computational burden. By using the implicit model, most of the variables are treated as equality constraints and thus are excluded from the estimation equations. The state variables will thus be not significantly more than the number of state variables in the traditional state estimation. The computation time increases roughly 50% compared to the traditional state estimation solution, which is quite acceptable considering the fact that the entire substation is represented in detail.

A natural extension of this work is to consider the same approach for parameter error detection and identification. Preliminary studies conducted as part of this project indicate promising results. The advantage of such an extension is that any sort of parameter errors or biases, such as those associated with sagging transmission lines, or missing transformer taps, etc. can be detected without requiring any prior knowledge or suspicion about these parameters. Furthermore, the procedure will facilitate simultaneous detection of bad data and parameter errors, enabling differentiation between occasional bad data and systematic errors due to wrong parameters.

REFERENCES


% start to input data
clear all
threshold=1e-3;

finput=fopen('vinput.dat');
ReadLine=fgets(finput);
BranchData=str2num(ReadLine);
I=1;
while BranchData(1)~=99
    busv(I) = BranchData(1);
    voltage(I) = BranchData(2);
    ReadLine=fgets(finput);
    BranchData=str2num(ReadLine);
    I=I+1;
end

finput=fopen('topoanainput.dat'); % input system network data
ReadLine=fgets(finput);
BranchData=str2num(ReadLine);
I=1;
while BranchData(1)~=99
    busnum(I) = BranchData(2)+1;
    linenum(I) = BranchData(3);
    bn=busnum(I)-1;
    for x=1:bn
        buscont(I,x)=BranchData(3+x);
    end
    ReadLine=fgets(finput);
    BranchData=str2num(ReadLine);
    I=I+1;
end
total_bus=I-1;

ReadLine=fgets(finput);
BranchData=str2num(ReadLine);
I=1;
while BranchData(1)~=99
    station(I) = BranchData(2);
    stat(I) = BranchData(3);
    linetype(I) = BranchData(4);
    linetap(I) = BranchData(5);
    startbus(I) = BranchData(6);
    endbus(I) = BranchData(7);
    startnode(I) = BranchData(8);
    endnode(I) = BranchData(9);
    liner(I)=BranchData(10);
    linex(I)=BranchData(11);
    con(I)=BranchData(12);
    sup(I)=BranchData(13);
ReadLine=fgets(finput);
BranchData=str2num(ReadLine);
I=I+1;
end

total_line=I-1;
fclose(finput);

finput=fopen('topomeasure.dat'); % input measurement data
ReadLine=fgets(finput);
BranchData=str2num(ReadLine);
I=1;
while BranchData(1)~=-99
    stationpflow(I)=BranchData(2);
    linenumpflow(I) = BranchData(3);
    startbuspflow(I) = BranchData(4);
    endbuspflow(I) = BranchData(5);
    measurepflow(I) = BranchData(6);
    measureweightpflow(I) = BranchData(7);
    ReadLine=fgets(finput);
    BranchData=str2num(ReadLine);
    I=I+1;
end

total_pflow=I-1;

ReadLine=fgets(finput);
BranchData=str2num(ReadLine);
I=1;
while BranchData(1)~=-99
    stationpinj(I)=BranchData(2);
    buspinj(I) = BranchData(3);
    measurepinj(I) = BranchData(4);
    measureweightpinj(I) = BranchData(5);
    ReadLine=fgets(finput);
    BranchData=str2num(ReadLine);
    I=I+1;
end

total_pinj=I-1;

[a,bussort]=sort(busv);
V=ones(sum(busnum)-total_bus,1);
for p=1:total_bus
    for x=1:busnum(p)-1
        V(buscont(p,x))=voltage(bussort(p));
    end
end

linez=zeros(total_line,1); % build line parameter matrix
zthet=zeros(total_line,1);
for x=1:total_line
    if linetype(x)==1
        linez(x)=linex(x)*linetap(x);
        zthet(x)=pi/2;
    else
        % code continues
    end

23
linez(x)=sqrt(liner(x)^2+linex(x)^2);
if liner(x)~=0
   zthet(x)=atan(linex(x)/liner(x));
else
   if linex(x)>0
      zthet(x)=pi/2;
   else
      zthet(x)=-pi/2;
   end
end
end
end
fclose(finput);

for p=1:total_bus
   temp=find(stationpflow==p);
   [a,b]=size(temp);
   flownum(p)=b;
   temp=find(stationpinj==p);
   [a,b]=size(temp);
   injnum(p)=b;
   linknum(p)=linenum(p)+injnum(p)-busnum(p)+1;
end
Plcom1=zeros(max(flownum)+max(injnum),max(linknum),total_bus);
Plcom2=zeros(max(flownum)+max(injnum),max(linknum),total_bus);
thetacom=zeros(max(linknum),max(busnum)-1,total_bus);
close=zeros(total_bus,1);
open=zeros(total_bus,1);
thresh=zeros(total_bus,1);
varnum=zeros(total_bus,1);
flag=0;
tree=zeros(1,2);
bp=find(stat==-1);
[a,bpsize]=size(bp);
for p=1:total_bus
   % build tree for each substation
   if busnum(p)>2
      % for substations with more than 2 out lines
      A=zeros(busnum(p),linenum(p)+total_pinj);
      m=1;
      for x=1:total_pinj
         if stationpinj(x)==p
            A(1,m)=1;
            for y=1:busnum(p)-1
               if buscont(p,y)==buspinj(x)
                  A(y+1,m)=-1;
                  tree(p,m)=total_line+x;
                  m=m+1;
               end
            end
         end
      end
   end
end
for x=1:total_line
    if station(x)==p & stat(x)==1
        for y=1:busnum(p)-1
            if buscont(p,y)==startnode(x)
                A(y+1,m)=1;
            end
            if buscont(p,y)==endnode(x)
                A(y+1,m)=-1;
            end
            tree(p,m)=x;
        end
        close(p)=close(p)+1;
        m=m+1;
    end
end
thresh(p)=m-1;
for x=1:total_line
    if station(x)==p & stat(x)==0
        for y=1:busnum(p)-1
            if buscont(p,y)==startnode(x)
                A(y+1,m)=1;
            end
            if buscont(p,y)==endnode(x)
                A(y+1,m)=-1;
            end
            tree(p,m)=x;
        end
        open(p)=open(p)+1;
        m=m+1;
    end
end
varnum(p)=m-1;

ground=0;
for x=1:busnum(p)-1
    if A(x,x)==0
        for y=x+1:m
            if A(x,y)==0
                A(:,x:y)=[A(:,y) A(:,x:y-1)];
                tree(p,x:y)=[tree(p,y) tree(p,x:y-1)];
                if ground==0
                    if y>x & y<=x+injnum(p)-1
                        A(:,x+1:thresh(p))=[A(:,x+injnum(p):thresh(p)) A(:,x+1:x+injnum(p)-1)];
                        tree(p,x+1:thresh(p))=[tree(p,x+injnum(p):thresh(p))
                        A(:,x+1:x+injnum(p)-1)];
                        tree(p,x+1:x+injnum(p)-1));
                    end
                end
            end
        end
    end
end
else
    if ground==0

\[ A(:,x+1:thresh(p)) = [A(:,x+injnum(p):thresh(p)) \\
A(:,x+1:x+injnum(p)-1)] ; \]
\[ \text{tree}(p,x+1:thresh(p)) = [\text{tree}(p,x+injnum(p):thresh(p)) \\
\text{tree}(p,x+1:x+injnum(p)-1)] ; \]
ground=1;
end
end
A(x,x+1:m) = A(x,x+1:m) / A(x,x);
for y=x+1:busnum(p)
for z=x+1:m
A(y,z) = A(y,z) - A(x,z) * A(y,x);
end
end
AA=zeros(busnum(p),linenum(p)+injnum(p));
for x=1:m-1
y=tree(p,x);
if y>total_line
y=y-total_line;
AA(1,x)=1;
for z=1:busnum(p)-1
if buscont(p,z)==buspinj(y)
AA(z+1,x)=-1;
end
end
else
for z=1:busnum(p)-1
if buscont(p,z)==startnode(y)
AA(z+1,x)=1;
end
if buscont(p,z)==endnode(y)
AA(z+1,x)=-1;
end
end
end
x=m;
for y=1:total_line
if (stat(y)==-1) & (startbus(y)==p | endbus(y)==p)
node=0;
for z=1:busnum(p)-1
if buscont(p,z)==startnode(y)
AA(z+1,x)=1;
node=1;
end
if buscont(p,z)==endnode(y)
AA(z+1,x)=-1;
node=-1;
end
end
if node==1
AA(1,x)=-1;
else
AA(1,x)=1;
end
end
end
    tree(p,x)=y;
    x=x+1;
end
end
AA=AA(1:busnum(p)-1,:);
At=AA(:,1:busnum(p)-1);
Al=AA(:,busnum(p):linenum(p)+injnum(p));
Dl=inv(At)*Al;
Bt=-Dl';
B=[Bt eye(linknum(p))];
nonzerobranch(p)=linenum(p)+injnum(p)-m+1;
z=1;
for y=1:bpsize
    if startbus(y)==p | endbus(y)==p
        nonbp(p,z)=y;
        z=z+1;
    end
end
Pcom=B';
[a,b]=size(Pcom);
Pflowcom=zeros(flownum(p)+injnum(p),b);
y=1;
for x=1:total_pflow
    if stationpflow(x)==p
        if startbuspflow(x)==startnode(linenumpflow(x))
            Pflowcom(y,:)=Pcom(find(tree(p,:)==linenumpflow(x)),:);
        else
            Pflowcom(y,:)=-Pcom(find(tree(p,:)==linenumpflow(x)),:);
        end
        y=y+1;
    end
end
for x=1:total_pinj
    z=x+total_line;
    if stationpinj(x)==p
        Pflowcom(y,:)=Pcom(find(tree(p,:)==z),:);
        y=y+1;
    end
end
Plcom1(1:flownum(p)+injnum(p),1:linknum(p)-nonzerobranch(p),p)=Pflowcom(:,1:linknum(p)-nonzerobranch(p));
Plcom2(1:flownum(p)+injnum(p),1:nonzerobranch(p),p)=Pflowcom(:,linknum(p)-nonzerobranch(p)+1:linknum(p));
    thetacom(1:nonzerobranch(p),1:busnum(p)-1,p)=-Bt(linknum(p)-nonzerobranch(p)+1:linknum(p),:);
else
    [a,b]=size(tree); % for substation with only 1 out lines
    tree(p,:)=zeros(1,b);
    for x=1:total_pflow
        if stationpflow(x)==p
            Pflowcom(y,1:nonzerobranch(p))=Pcom(find(tree(p,:)==y),1:nonzerobranch(p));
        else
            Pflowcom(y,1:nonzerobranch(p))=-Pcom(find(tree(p,:)==y),1:nonzerobranch(p));
        end
        y=y+1;
    end
end
end
if startbuspflow(x)==p | endbuspflow(x)==p
    flag=1;
end
end
if flag==1
    y=0;
    for x=1:total_line
        if station(x)==0
            if startbus(x)==p
                thetacom(y+1,1,p)=-1;
                y=y+1;
            end
            if endbus(x)==p
                thetacom(y+1,1,p)=1;
                y=y+1;
            end
        end
    end
    nonzerobranch(p)=y;
    z=1;
    for y=1:bpsize
        if startbus(y)==p | endbus(y)==p
            nonbp(p,z)=y;
            z=z+1;
        end
    end
    y=0;
    for x=1:total_pflow
        if stationpflow(x)==p
            if startbuspflow(x)==startnode(linenumpflow(x))
                a=linenumpflow(x);
                for z=1:nonzerobranch(p)
                    if nonbp(p,z)==a
                        Plcom2(1+y,z,p)=1;
                    end
                end
                y=y+1;
            else
                a=linenumpflow(x);
                for z=1:nonzerobranch(p)
                    if nonbp(p,z)==a
                        Plcom2(1+y,z,p)=-1;
                    end
                end
                y=y+1;
            end
        end
    end
    for x=1:total_pinj
        if stationpinj(x)==p
            temp=cumsum(Plcom2(1:y,:,p));
            Plcom2(1+y,:,p)=temp(y,:);
        end
    end
    varnum(p)=1;
flag=0;
end

for p=1:total_bus
    for x=1:nonzerobranch(p)
        for y=1:busnum(p)-1
            if thetacom(x,y,p)==0
                if tree(p,y)<total_line & tree(p,y)==0
                    if stat(tree(p,y))==0
                        thetacom(x,:,p)=zeros(1,max(busnum)-1);
                    end
                end
            end
        end
    end
end

variable=zeros(sum(varnum),1);
delta=1;
threshold=1e-3;

H=zeros(total_pflow+total_pinj,sum(varnum));

while delta>=threshold
    % state estimation part
    startrow=0;
    startcol=0;
    x=0;
    y=0;
    deltaz=zeros(sum(flownum)+sum(injnum),1);
    for p=1:total_bus
        deltaz(1+x+y:x+y+flownum(p))=measurepflow(1+x:x+flownum(p));
        x=x+flownum(p);
        deltaz(1+x+y:x+y+injnum(p))=measurepinj(1+y:y+injnum(p));
        y=y+injnum(p);
    end
    for p=1:total_bus
        % build Jacobian matrix
        for x=1:flownum(p)+injnum(p)
            for y=1:nonzerobranch(p)
                temp1=Plcom2(x,y,p)*thetacom(y,1:busnum(p)-1,p);
                n=nonbp(p,y);
                if startbus(n)==p
                    q=endbus(n);
                else
                    q=startbus(n);
                end
                for v=1:nonzerobranch(q)
                    if nonbp(q,v)==n
                        node=v;
                    end
                end
                if Plcom2(x,y,p)==1
                    temp2=thetacom(node,1:busnum(q)-1,q);
                end
            end
        end
    end
end
from=startnode(n);
to=endnode(n);
elseif Plcom2(x,y,p)==-1
    temp2=(-1)*thetacom(node,1:busnum(q)-1,q);
    from=endnode(n);
to=startnode(n);
else
    temp2=0*thetacom(node,1:busnum(q)-1,q);
end
startq=0;
for v=1:q-1
    startq=startq+varnum(v);
end
if nnz(temp1)~=0 & nnz(temp2)~=0
    for z=1:busnum(p)-1
        H(startrow+x,startcol+z)=H(startrow+x,startcol+z)+temp1(z)*V(from)*V(to)*s
in(temp1*variable(startcol+1:startcol+busnum(p)-
    1)+temp2*variable(startq+1:startq+busnum(q)-1)+zthet(n))/linez(n);
    end
    deltaz(startrow+x)=deltaz(startrow+x)-
    V(from)*cos(temp1*variable(startcol+1:startcol+busnum(p)-
    1)+temp2*variable(startq+1:startq+busnum(q)-1)+zthet(n))/linez(n);
end
if busnum(p)>2
    deltaz(startrow+x)=deltaz(startrow+x)-
    Plcom1(x,1:linknum(p)-
    nonzerobranch(p),p)*variable(startcol+busnum(p):startcol+varnum(p));
end
end
if busnum(p)>2
    H(startrow+1:startrow+flownum(p)+injnum(p),startcol+busnum(p):startcol+varnum(p))=Plcom1(1:flownum(p)+injnum(p),1:linknum(p)-nonzerobranch(p),p);
end
startrow=startrow+flownum(p)+injnum(p);
startcol=startcol+varnum(p);
end

startcol=0;
y=1;
z=1;
cbnum=0;
pxinum=0;
cbnumber=zeros(total_bus,1);
for p=1:total_bus % divide the Jacobian matrix into
    two parts
if busnum(p)>2                  % Hi and Hcb according to the tree
determined
for x=1:busnum(p)-1
    if tree(p,x)<=total_line & stat(tree(p,x))==1
        Hcb(:,y)=H(:,startcol+x);
        cb(p,y-cbnum)=tree(p,x);
        y=y+1;
    else
        if tree(p,x)>total_line | stat(tree(p,x))~=0
            Hi(:,z)=H(:,startcol+x);
            pxi(z)=startcol+x;
            z=z+1;
        end
    end
end
for x=busnum(p):varnum(p)
    if tree(p,x)<=total_line & stat(tree(p,x))==0
        Hcb(:,y)=H(:,startcol+x);
        cb(p,y-cbnum)=tree(p,x);
        y=y+1;
    elseif tree(p,x)<=total_line & stat(tree(p,x))==1
        Hi(:,z)=H(:,startcol+x);
        pxi(z)=startcol+x;
        z=z+1;
        cb(p,y-cbnum)=tree(p,x);
        y=y+1;
    else
        if tree(p,x)>total_line
            Hi(:,z)=H(:,startcol+x);
            pxi(z)=startcol+x;
            z=z+1;
        end
    end
end
else
    for x=1:total_pflow
        if startbuspflow(x)==p | endbuspflow(x)==p
            flag=1;
        end
    end
    if flag==1
        Hi(:,z)=H(:,startcol+1);
        pxi(z)=startcol+1;
        z=z+1;
        flag=0;
    end
startcol=startcol+varnum(p);
cbnum=y-1;
cbnumber(p)=cbnum-sum(cbnumber);
end
[m,n]=size(Hi);
Hi=Hi(:,2:n);
Ro=[measureweightpflow';measureweightpinj'];
Hp=Hi;
Hg=Hcb;
Ci=Hp;
ci=deltaz;
Ccb=Hg;
cnumber=zeros(sum(injnum),1);
y=0;
z=0;
q=0;
cn=1;
for p=1:total_bus
    x=flownum(p);
    Ci(q+1:x+q,:)=[];
    ci(q+1:x+q,:)=[];
    Ccb(q+1:x+q,:)=[];
    y=y+flownum(p);
    x=injnum(p);
    qq=0;
    while x>0
        if measurepinj(z+x)~=0
            Ci(x+q,:)=[];
            ci(x+q,:)=[];
            Ccb(x+q,:)=[];
        else
            Hp(x+y,:)=[];
            Hg(x+y,:)=[];
            Ro(x+y)=[];
            deltapinj(x+y)=[];
            cnumber(cn)=z+x;
            cn=cn+1;
            qq=qq+1;
        end
        x=x-1;
    end
    y=y+injnum(p)-qq;
    z=z+injnum(p);
end
R=diag(Ro);
W=inv(R);
[m,n]=size(Hp);
[p,q]=size(Ci);
G=[zeros(n,n) Hp'*W Ci';Hp eye(m) zeros(m,p);Ci zeros(p,m)
zeros(p,p)];
vec=[zeros(n,1);deltaz;ci];
Lag=inv(G);
quest=Lag*vec;
deltaxi=quest(1:n);
delta=max(abs(deltaxi));
for x=1:n
    variable(pxi(x+1))=variable(pxi(x+1))+deltaxi(x);
end
end
mu=quest(n+1:n+m+p);
T=-[Hg'*W' Ccb'];               % compute lamda
lumda=T*mu;
E1=Lag(n+1:n+m,n+1:n+m);
E2=Lag(1+n+m:n+m+p,n+1:n+m);
cov=[E1;E2];
covr=cov*R*cov';
covlumda=T*covr*T';
[a,b]=size(Hcb);
lumdan=zeros(b,1);              % normalize lamda
for x=1:b
    if covlumda(x,x)~=0
        lumdan(x)=abs(lamda(x)/sqrt(covlumda(x,x)));
    end
end
r=quest(n+1:n+m+p);             % normalize r
rn=zeros(m+p,1);
for x=1:m+p
    if covr(x,x)~=0
        rn(x)=abs(r(x)/sqrt(covr(x,x)));
    end
end
x=cn-1;
zpinj=zeros(cn-1,1);
while x>0
    zpinj(x)=buspinj(cnumber(x));
    buspinj(cnumber(x))=[];
    x=x-1;
end
for x=1:cn-1
    p=stationpinj(cnumber(x));
    injnum(p)=injnum(p)-1;
end
% Output the results
foutput = fopen('BADDATAOTP.dat','w');
fprintf(foutput,'
The maximum normalized lagrange multiplier 
');
[lumdamax,I]=max(lumdan);
y=0;
for p=1:total_bus
    for x=1:cbnumber(p)
        if I==y+x
fprintf(foutput,\n%5d%5d%15.4f\n',startnode(cb(p,x)),endnode(cb(p,x)),lambda max);
    end
    end
    y=y+cbnumber(p);
end

fprintf(foutput,\n\nNormalized lagrange multiplier \n');
y=0;
for p=1:total_bus
    for x=1:cbnumber(p)
        fprintf(foutput,\n%5d%5d%15.4f\n',startnode(cb(p,x)),endnode(cb(p,x)),lambda n(y+x));
        end
        y=y+cbnumber(p);
    end
end

fprintf(foutput,\n\nNormalized residual \n');
y=0;
z=0;
for p=1:total bus
    for x=1:flownum(p)
        fprintf(foutput,\n%3d%5d%5d%15.4f\n',1,startbuspflow(y+x),endbuspflow(y+x),rn(y+z+x));
        end
        y=y+flownum(p);
        for x=1:injnum(p)
            fprintf(foutput,\n%3d%5d%20.4f\n',3,buspinj(z+x),rn(y+z+x));
            end
            z=z+injnum(p);
        end
for x=1:cn-1
    fprintf(foutput,\n%3d%5d%20.4f\n',3,zpinj(x),rn(m+x));
end
fprintf(foutput,\n\n');
close(foutput);
APPENDIX 2. Input Data Files Generating Program

The input data files for the topology analysis program include the detailed circuit breaker connection information and the power flow measurements through the circuit breakers. For large systems, it is a hard work to get these data by hand according to the complexity of the system. In such cases, this program will be applied to generate the topology analysis input data files from the normal state estimation data to make a test of the large systems.

A2.1 Standard Substation Model

Since the objective of this program is to generate the data for testing, the specific substation will be given the standard model on the topological structure.

A2.1.1 One-line Substation

![One-line Substation Model](image)

Figure 8.1 One-line Substation Model

There is no circuit breaker in this substation.
A2.1.2 Two-line Substation
The figure is shown below. There are 4 circuit breakers in this substation.

A2.1.3 Three-line Substation
There are 5 circuit breakers.
A2.1.4 Four-line Substation

There are 6 circuit breakers in this substation.

Figure 8.4 Four-line Substation Model
A2.1.5 Five-line Substation

There are 8 circuit breakers in this substation.

Figure 8.5 Five-line Substation Model
A2.1.6 Six-line Substation

There are 9 circuit breakers in this substation.

Figure 8.6 Six-line Substation Model
A2.1.7  Seven-line Substation
There are 11 circuit breakers in this substation.

The power flow measurements of circuit breakers in the specific substation will be calculated according to its own topology.

A2.2  Input Data Files
The input data is the normal state estimation input data. It is composed of two files, the line parameter data and the measurements data. However, For the purpose of topology analysis, only the real power flow data is needed. And the power flows of both directions of the lines and all of the bus real injections are necessary. The data files of the 14-bus system will be listed below.

A2.2.1  seinput.dat

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1.00000</td>
<td>0.01938</td>
<td>0.05917</td>
<td>0.00000</td>
<td>0.05280</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>1.00000</td>
<td>0.05403</td>
<td>0.22304</td>
<td>0.00000</td>
<td>0.04920</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>1.00000</td>
<td>0.05695</td>
<td>0.17388</td>
<td>0.00000</td>
<td>0.03400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>1.00000</td>
<td>0.05811</td>
<td>0.17632</td>
<td>0.00000</td>
<td>0.03740</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1.00000</td>
<td>0.04699</td>
<td>0.19797</td>
<td>0.00000</td>
<td>0.04380</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1.00000</td>
<td>0.06701</td>
<td>0.17103</td>
<td>0.00000</td>
<td>0.03460</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>1.00000</td>
<td>0.01335</td>
<td>0.04211</td>
<td>0.00000</td>
<td>0.01280</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>8</td>
<td>0</td>
<td>1.00000</td>
<td>0.00000</td>
<td>0.17615</td>
<td>0.00000</td>
<td>0.00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>9</td>
<td>0</td>
<td>1.00000</td>
<td>0.00000</td>
<td>0.11001</td>
<td>0.00000</td>
<td>0.00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>10</td>
<td>0</td>
<td>1.00000</td>
<td>0.03181</td>
<td>0.08450</td>
<td>0.00000</td>
<td>0.00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>14</td>
<td>0</td>
<td>1.00000</td>
<td>0.12711</td>
<td>0.27038</td>
<td>0.00000</td>
<td>0.00000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This input file includes the line parameter data. –99 is the flag of the end of the data.

1st column: the line number.
2nd to 3rd columns: the terminal buses of the line.
4th column: the line type. For a transformer, the value is 1 and for a normal transmission line, the value is 0.
5th column: tap value. For a transformer, it gives the tap value of that transformer. For normal transmission lines it is 1.
6th to 9th columns: the resistance, reactance, conductance and susceptance of the branch.

A2.2.2 measureinput.dat

1 15 12 13  0.01195  0.000001
1 12 11  6  -0.05548  0.000001
1  3  5  2  -0.40324  0.000001
1  5  2  3   0.73607  0.000001
1  9  7  9   0.30244  0.000001
1 14  6 13   0.16723  0.000001
1  4  4  2  -0.54544  0.000001
1  1  1  2   1.57062  0.000001
1  6  3  4  -0.22939  0.000001
1  8  7  8   0.00000  0.000001
1 13  6 12   0.07367  0.000001
1 10  9 10   0.07033  0.000001
1 11  9 14   0.10972  0.000001
1  2  1  5   0.75538  0.000001
1  2  5  1  -0.72781  0.000001
1  1  2  1  -1.52755  0.000001
1  8  8  7   0.00000  0.000001
1  9  9  7  -0.30244  0.000001
1 17 10 11  -0.02022  0.000001
1 17 11 10   0.02048  0.000001
1 16 14 13  -0.04165  0.000001
1 16 13 14   0.04209  0.000001
1 15 13 12  -0.01191  0.000001
1 14 13  6  -0.16520  0.000001
1 13 12  6  -0.07294  0.000001
1 12  6 11   0.05590  0.000001
1 10 10  9  -0.06978  0.000001
This input file includes the real power measurements of the line flows and bus injections. It is composed of these two parts.

**Power Flow Data**
1st column: the flag of the measurement. 1 represents it is a real power flow measurement.
2nd column: the line number of that power flow.
3rd to 4th columns: The start and end bus number of that power flow.
5th column: the power flow measurement.
6th column: the weight of that measurement.

**Power Injection Data**
1st column: the flag of the measurement. 3 represents it is a real power injection measurement.
2nd column: the bus number of that power injection.
3rd column: the power injection measurement.
4th column: the weight of that measurement.

A2.3 Output Data files

The output data files of this program are the input data of the topology analysis program. It is composed of two files, the detailed CB topology and the CB measurements.

A2.3.1 topoanainput.dat

```
1 4 6 1 15 16 17
2 6 10 2 18 19 20 21 22
3 4 6 3 23 24 25
4 7 13 4 26 27 28 29 30 31
5 6 10 5 32 33 34 35 36
6 6 10 6 37 38 39 40 41
7 5 8 7 42 43 44 45
8 1 1 8
9 6 10 9 46 47 48 49 50
10 4 6 10 51 52 53
11 4 6 11 54 55 56
12 4 6 12 57 58 59
13 5 8 13 60 61 62 63
14 4 6 14 64 65 66
-99
```

---

```
1 0 -1 0 1.000 1 2 16 19 0.01938 0.05917 0.00000 0.05280
2 0 -1 0 1.000 1 5 17 33 0.05403 0.22304 0.00000 0.04920
3 0 -1 0 1.000 2 5 20 34 0.06959 0.17388 0.00000 0.04380
4 0 -1 0 1.000 2 4 21 27 0.06701 0.17103 0.00000 0.03460
5 0 -1 0 1.000 3 4 25 28 0.01335 0.04211 0.00000 0.01280
6 0 -1 0 1.000 4 5 29 35 0.00000 0.17615 0.00000 0.00000
7 0 -1 0 1.000 7 8 43 52 0.00000 0.17615 0.00000 0.00000
8 0 -1 0 1.000 7 9 44 47 0.00000 0.11001 0.00000 0.00000
9 0 -1 0 1.000 7 9 44 47 0.00000 0.11001 0.00000 0.00000
10 0 -1 0 1.000 9 10 48 52 0.03181 0.08450 0.00000 0.00000
11 0 -1 0 1.000 9 14 49 65 0.12711 0.27038 0.00000 0.00000
12 0 -1 0 1.000 6 11 38 55 0.09498 0.19890 0.00000 0.00000
13 0 -1 0 1.000 6 12 39 58 0.12291 0.25581 0.00000 0.00000
14 0 -1 0 1.000 6 13 40 61 0.06615 0.13027 0.00000 0.00000
15 0 -1 0 1.000 12 13 59 62 0.22092 0.19988 0.00000 0.00000
16 0 -1 0 1.000 13 14 63 66 0.17093 0.34802 0.00000 0.00000
17 0 -1 0 1.000 10 11 53 56 0.08205 0.19207 0.00000 0.00000
18 0 -1 1 1.090 6 5 41 36 0.00000 0.25202 0.00000 0.00000
19 0 -1 1 1.090 6 5 41 36 0.00000 0.55618 0.00000 0.00000
20 0 -1 1 0.972 4 7 31 45 0.00000 0.20912 0.00000 0.00000
21 1 1 -1 -1 -1 -1 1 16 0.00000 0.00000 0.00000 0.00000
22 1 1 -1 -1 -1 -1 1 16 0.00000 0.00000 0.00000 0.00000
23 1 0 -1 -1 -1 -1 1 17 0.00000 0.00000 0.00000 0.00000
24 1 1 -1 -1 -1 -1 17 15 0.00000 0.00000 0.00000 0.00000
25 2 1 -1 -1 -1 -1 2 19 0.00000 0.00000 0.00000 0.00000
26 2 0 -1 -1 -1 -1 19 20 0.00000 0.00000 0.00000 0.00000
27 2 1 -1 -1 -1 -1 20 18 0.00000 0.00000 0.00000 0.00000
28 2 1 -1 -1 -1 -1 2 21 0.00000 0.00000 0.00000 0.00000
29 2 1 -1 -1 -1 -1 21 22 0.00000 0.00000 0.00000 0.00000
30 2 1 -1 -1 -1 -1 22 18 0.00000 0.00000 0.00000 0.00000
31 3 1 -1 -1 -1 -1 3 24 0.00000 0.00000 0.00000 0.00000
32 3 1 -1 -1 -1 -1 3 24 0.00000 0.00000 0.00000 0.00000
33 3 0 -1 -1 -1 -1 3 25 0.00000 0.00000 0.00000 0.00000
34 3 1 -1 -1 -1 -1 3 25 23 0.00000 0.00000 0.00000 0.00000
......
This file is the parameter of the circuit breakers in the system. It includes two parts.

**Substation Information Data**
This data section is about the overall information of the substation.
1st column: substation number.
2nd column: the total number of electrical nodes in the substation.
3rd column: total number of branches belong to that substation. It includes all the circuit breakers and the non-zero impedance branches connected to this station.
4th column to the end: the number of columns after the 3rd column differs for each substation, it shows the detailed node number of that substation.

**Circuit Breaker Topology Data**
1st column: branch number.
2nd column: station number. For the non-zero impedance branches, it is 0. And for circuit breakers, it is the station number that CB belongs to.
3rd column: CB status. If the CB is open, the value is 0. if it is closed, the value is 1. For the non-zero impedance branches, it is -1.
4th column: the line type. For a transformer, the value is 1 and for a normal transmission line, the value is 0. For circuit breakers, the value is –1.
5th column: tap value. For a transformer, it gives the tap value of that transformer. For normal transmission line it is 1. For circuit breaker it is –1.
6th to 7th columns: the two station numbers connected by a non-zero impedance branch. For the circuit breaker, these values will be set as –1.
8th to 9th columns: the two terminal nodes numbers of a branch, for both the non-zero impedance branch and circuit breaker.
10th to 13th columns: the resistance, reactance, conductance and susceptance of the branch. For the circuit breaker, the values will be all zeros.

A2.3.2 topomeasure.dat

```
1 1 1 16 19 1.570620 0.000001
1 1 2 17 33 0.755380 0.000001
1 1 21 1 16 2.326000 0.000001
1 1 22 16 15 0.755380 0.000001
1 1 24 17 15 -0.755380 0.000001
1 2 5 22 24 0.736070 0.000001
1 2 1 19 16 -1.527550 0.000001
1 2 3 20 34 0.412110 0.000001
1 2 4 21 27 0.562330 0.000001
1 2 25 2 19 -1.527550 0.000001
```
This file is the detailed power flow measurements through the circuit breaker. It is composed of two parts.

**Power Flow Measurements**

1st column: the flag of the measurement. 1 represents it is a real power flow measurement.

2nd column: the station number that power flow belongs to, whether it is a non-zero impedance branch connected to that station or it is a circuit breaker in that station.

3rd column: the branch number of that power flow.

4th to 5th columns: the start and end node number of that power flow.

6th column: the power flow measurement.

7th column: the weight of that measurement.

**Power Injection Measurements**

1st column: the flag of the measurement. 3 represents it is a real power injection measurement.

2nd column: the substation number of that power injection belongs to.
3rd column: the node number of that power injection. For the standard model used in this program, this node number will be the same with the substation number.

4th column: the power injection measurement.

5th column: the weight of that measurement.

### A2.4 Matlab Code

```matlab
% start to input data
clear all

finput=fopen('seinput.dat'); % input system network data
ReadLine=fgets(finput);
BranchData=str2num(ReadLine);
I=1;
while BranchData(1)~=-99
    startbus(I) = BranchData(2);
    endbus(I) = BranchData(3);
    linetype(I)=BranchData(4);
    linetap(I)=BranchData(5);
    liner(I)=BranchData(6);
    linex(I)=BranchData(7);
    con(I)=BranchData(8);
    sup(I)=BranchData(9);
    ReadLine=fgets(finput);
    BranchData=str2num(ReadLine);
    I=I+1;
end
total_line=I-1;
total_bus=max(max(startbus),max(endbus));

fclose(finput);

finput=fopen('measureinput.dat'); % input measurement data
ReadLine=fgets(finput);
BranchData=str2num(ReadLine);
I=1;
while BranchData(1)~=-99
    linenumpflow(I)=BranchData(2);
    startbuspflow(I) = BranchData(3);
    endbuspflow(I) = BranchData(4);
    measurepflow(I) = BranchData(5);
    measureweightpflow(I) = BranchData(6);
    ReadLine=fgets(finput);
    BranchData=str2num(ReadLine);
    I=I+1;
end
total_pflow=I-1;

ReadLine=fgets(finput);
BranchData=str2num(ReadLine);
I=1;
```

46
while BranchData(1)~=-99
    buspinj(I) = BranchData(2);
    measurepinj(I) = BranchData(3);
    measureweightpinj(I) = BranchData(4);
    ReadLine=fgets(finput);
    BranchData=str2num(ReadLine);
    I=I+1;
end

total_pinj=I-1;
fclose(finput);

station(1:total_line,1)=zeros(total_line,1);
stat(1:total_line,1)=-ones(total_line,1);
startbuscb(1:total_line,1)=startbus';
endbuscb(1:total_line,1)=endbus';
startnode(1:total_line,1)=zeros(total_line,1);
endnode(1:total_line,1)=zeros(total_line,1);
linenum=total_line;
nodenum=total_bus;
a=0;
b=0;
c=0;
d=0;
for x=1:total_bus
    % build the substation structure
    [a,b]=size(find(startbus==x));
    [c,d]=size(find(endbus==x));
    outnum=b+d;
    if outnum==1
        totalnode(x)=1;
        totalout(x)=1;
        nodecont(x,1)=x;
    elseif outnum==2
        station(linenum+1:linenum+4)=x*ones(4,1);
        stat(linenum+1:linenum+4)=ones(4,1);
        stat(linenum+3)=0;
        startbuscb(linenum+1:linenum+4)=-ones(4,1);
        endbuscb(linenum+1:linenum+4)=-ones(4,1);
        startnode(linenum+1)=x;
        endnode(linenum+1)=nodenum+2;
        startnode(linenum+2)=nodenum+2;
        endnode(linenum+2)=nodenum+1;
        startnode(linenum+3)=x;
        endnode(linenum+3)=nodenum+3;
        startnode(linenum+4)=nodenum+3;
        endnode(linenum+4)=nodenum+1;
        totalnode(x)=4;
        totalout(x)=6;
        nodecont(x,1)=x;
        for y=2:4
            nodecont(x,y)=nodenum+y-1;
        end
        linenum=linenum+4;
        nodenum=nodenum+3;
    end
end
elseif outnum==3
    station(linenum+1:linenum+5)=x*ones(5,1);
    stat(linenum+1:linenum+5)=ones(5,1);
    stat(linenum+2)=0;
    startbuscb(linenum+1:linenum+5)=-ones(5,1);
    endbuscb(linenum+1:linenum+5)=-ones(5,1);
    startnode(linenum+1)=x;
    endnode(linenum+1)=nodenum+2;
    startnode(linenum+2)=nodenum+2;
    endnode(linenum+2)=nodenum+3;
    startnode(linenum+3)=nodenum+3;
    endnode(linenum+3)=nodenum+1;
    startnode(linenum+4)=x;
    endnode(linenum+4)=nodenum+4;
    startnode(linenum+5)=nodenum+4;
    endnode(linenum+5)=nodenum+1;
    totalnode(x)=5;
    totalout(x)=8;
    nodecont(x,1)=x;
    for y=2:5
        nodecont(x,y)=nodenum+y-1;
    end
    linenum=linenum+5;
    nodenum=nodenum+4;
elseif outnum==4
    station(linenum+1:linenum+6)=x*ones(6,1);
    stat(linenum+1:linenum+6)=ones(6,1);
    stat(linenum+2)=0;
    startbuscb(linenum+1:linenum+6)=-ones(6,1);
    endbuscb(linenum+1:linenum+6)=-ones(6,1);
    startnode(linenum+1)=x;
    endnode(linenum+1)=nodenum+2;
    startnode(linenum+2)=nodenum+2;
    endnode(linenum+2)=nodenum+3;
    startnode(linenum+3)=nodenum+3;
    endnode(linenum+3)=nodenum+1;
    startnode(linenum+4)=x;
    endnode(linenum+4)=nodenum+4;
    startnode(linenum+5)=nodenum+4;
    endnode(linenum+5)=nodenum+5;
    startnode(linenum+6)=nodenum+5;
    endnode(linenum+6)=nodenum+1;
    totalnode(x)=6;
    totalout(x)=10;
    nodecont(x,1)=x;
    for y=2:6
        nodecont(x,y)=nodenum+y-1;
    end
    linenum=linenum+6;
    nodenum=nodenum+5;
elseif outnum==5
    station(linenum+1:linenum+8)=x*ones(8,1);
    stat(linenum+1:linenum+8)=ones(8,1);
    stat(linenum+5)=0;
    stat(linenum+7)=0;
startbuscb(linenum+1:linenum+8)=-ones(8,1); 
endbuscb(linenum+1:linenum+8)=-ones(8,1); 
startnode(linenum+1)=x; 
endnode(linenum+1)=nodenum+2; 
startnode(linenum+2)=nodenum+2; 
endnode(linenum+2)=nodenum+3; 
startnode(linenum+3)=nodenum+3; 
endnode(linenum+3)=nodenum+1; 
startnode(linenum+4)=x; 
endnode(linenum+4)=nodenum+4; 
startnode(linenum+5)=nodenum+4; 
endnode(linenum+5)=nodenum+5; 
startnode(linenum+6)=nodenum+5; 
endnode(linenum+6)=nodenum+1; 
startnode(linenum+7)=x; 
endnode(linenum+7)=nodenum+6; 
startnode(linenum+8)=nodenum+6; 
endnode(linenum+8)=nodenum+1; 
totalnode(x)=7; 
totalout(x)=13; 
nodecont(x,1)=x; 
for y=2:7 
    nodecont(x,y)=nodenum+y-1; 
end 
linenum=linenum+8; 
nodenum=nodenum+6; 
elseif outnum==6 
    station(linenum+1:linenum+9)=x*ones(9,1); 
    stat(linenum+1:linenum+9)=ones(9,1); 
    stat(linenum+5)=0; 
    stat(linenum+8)=0; 
    startbuscb(linenum+1:linenum+9)=-ones(9,1); 
    endbuscb(linenum+1:linenum+9)=-ones(9,1); 
    startnode(linenum+1)=x; 
    endnode(linenum+1)=nodenum+2; 
    startnode(linenum+2)=nodenum+2; 
    endnode(linenum+2)=nodenum+3; 
    startnode(linenum+3)=nodenum+3; 
    endnode(linenum+3)=nodenum+1; 
    startnode(linenum+4)=x; 
    endnode(linenum+4)=nodenum+4; 
    startnode(linenum+5)=nodenum+4; 
    endnode(linenum+5)=nodenum+5; 
    startnode(linenum+6)=nodenum+5; 
    endnode(linenum+6)=nodenum+1; 
    startnode(linenum+7)=x; 
    endnode(linenum+7)=nodenum+6; 
    startnode(linenum+8)=nodenum+6; 
    endnode(linenum+8)=nodenum+1; 
    totalnode(x)=8; 
    totalout(x)=15; 
nodecont(x,1)=x; 
for y=2:8 

nodecont(x,y)=nodenum+y-1;
end
linenum=linenum+9;
nodenum=nodenum+7;
elseif outnum==7
station(linenum+1:linenum+11)=x*ones(11,1);
stat(linenum+1:linenum+11)=ones(11,1);
stat(linenum+5)=0;
stat(linenum+8)=0;
stat(linenum+10)=0;
startbuscb(linenum+1:linenum+11)=-ones(11,1);
endbuscb(linenum+1:linenum+11)=-ones(11,1);
startnode(linenum+1)=x;
endnode(linenum+1)=nodenum+2;
startnode(linenum+2)=nodenum+2;
endnode(linenum+2)=nodenum+3;
startnode(linenum+3)=nodenum+3;
endnode(linenum+3)=nodenum+1;
startnode(linenum+4)=x;
endnode(linenum+4)=nodenum+4;
startnode(linenum+5)=nodenum+4;
endnode(linenum+5)=nodenum+5;
startnode(linenum+6)=nodenum+5;
endnode(linenum+6)=nodenum+1;
startnode(linenum+7)=x;
endnode(linenum+7)=nodenum+6;
startnode(linenum+8)=nodenum+6;
endnode(linenum+8)=nodenum+7;
startnode(linenum+9)=nodenum+7;
endnode(linenum+9)=nodenum+1;
startnode(linenum+10)=x;
endnode(linenum+10)=nodenum+8;
startnode(linenum+11)=nodenum+8;
endnode(linenum+11)=nodenum+1;
totalnode(x)=9;
totalout(x)=18;
odecont(x,1)=x;
for y=2:9
    nodecont(x,y)=nodenum+y-1;
end
linenum=linenum+11;
nodenum=nodenum+8;
end

sequence=nodecont; % build the detailed node connection for each branch
for x=1:total_line
    from=startbuscb(x);
to=endbuscb(x);
    frombus=find(sequence(from,:));
tobus=find(sequence(to,:));
    if frombus==1
        startnode(x)=sequence(from,1);
    else
        % additional code
    end
end
startnode(x)=sequence(from, frombus(3));
sequence(from, frombus(3))=0;
end
if tobus==1
   endnode(x)=sequence(to, 1);
else
   endnode(x)=sequence(to, tobus(3));
   sequence(to, tobus(3))=0;
end
end
linercb=zeros(linenum, 1);
linexcb=zeros(linenum, 1);
supcb=zeros(linenum, 1);
concb=zeros(linenum, 1);
linercb(1:total_line)=liner';
linexcb(1:total_line)=linex';
supcb(1:total_line)=sup';
concb(1:total_line)=con';
pflownum=0;
for x=1:total_bus
   z=1;
   for y=1:total_pflow % create the measurements for the non-zero impedance branches
      if startbuspflow(y)==x
         lnum=linenumpflow(y);
         linenumpflowcb(pflownum+z)=lnum;
         if startbuscb(lnum)==x
            startbuspflowcb(pflownum+z)=startnode(lnum);
            endbuspflowcb(pflownum+z)=endnode(lnum);
         else
            startbuspflowcb(pflownum+z)=endnode(lnum);
            endbuspflowcb(pflownum+z)=startnode(lnum);
         end
         measurepflowcb(pflownum+z)=measurepflow(y);
         measureweightpflowcb(pflownum+z)=measureweightpflow(y);
         z=z+1;
      end
      if totalnode(x)==1 % create the measurements for CB in each substation
         measnum=1;
         stationpflow(pflownum+1)=x;
         pflownum=pflownum+1;
      elseif totalnode(x)==4
         measnum=5;
         stationpflow(pflownum+1:pflownum+5)=x*ones(5, 1);
         for y=1:total_pinj
            if buspinj(y)==x
               pinj=measurepinj(y);
            end
         end
         david=linenumpflowcb(pflownum+1:pflownum+2);
for z=1:2
    [C,I]=min(david);
    pflow(z)=measurepflowcb(pflownum+I);
    david(I)=9999;
end
for y=1:linenum
    if station(y)==x && stat(y)==1
        linenumpflowcb(pflownum+3)=y;
        startbuspflowcb(pflownum+3)=startnode(y);
        endbuspflowcb(pflownum+3)=endnode(y);
        measurepflowcb(pflownum+3)=pinj;
        measureweightpflowcb(pflownum+3)=1e-6;
        break;
    end
end
linenumpflowcb(pflownum+4)=y+1;
startbuspflowcb(pflownum+4)=startnode(y+1);
endbuspflowcb(pflownum+4)=endnode(y+1);
measurepflowcb(pflownum+4)=pflow(1);
measureweightpflowcb(pflownum+4)=1e-6;
linenumpflowcb(pflownum+5)=y+3;
startbuspflowcb(pflownum+5)=startnode(y+3);
endbuspflowcb(pflownum+5)=endnode(y+3);
measurepflowcb(pflownum+5)=-pflow(2);
measureweightpflowcb(pflownum+5)=1e-6;
pflownum=pflownum+5;
elseif totalnode(x)==5
    measnum=7;
    stationpflow(pflownum+1:pflownum+7)=x*ones(7,1);
    for y=1:total_pinj
        if buspinj(y)==x
            pinj=measurepinj(y);
        end
    end
    david=linenumpflowcb(pflownum+1:pflownum+3);
    for z=1:3
        [C,I]=min(david);
        pflow(z)=measurepflowcb(pflownum+I);
        david(I)=9999;
    end
    for y=1:linenum
        if station(y)==x && stat(y)==1
            linenumpflowcb(pflownum+4)=y;
            startbuspflowcb(pflownum+4)=startnode(y);
            endbuspflowcb(pflownum+4)=endnode(y);
            measurepflowcb(pflownum+4)=pflow(1);
            measureweightpflowcb(pflownum+4)=1e-6;
            break;
        end
    end
    linenumpflowcb(pflownum+5)=y+2;
    startbuspflowcb(pflownum+5)=startnode(y+2);
    endbuspflowcb(pflownum+5)=endnode(y+2);
    measurepflowcb(pflownum+5)=-pflow(2);
    measureweightpflowcb(pflownum+5)=1e-6;
linenumpflowcb(pflownum+6) = y+3;
startbuspflowcb(pflownum+6) = startnode(y+3);
endbuspflowcb(pflownum+6) = endnode(y+3);
measurepflowcb(pflownum+6) = pinj - pflow(1);
measureweightpflowcb(pflownum+6) = 1e-6;
linenumpflowcb(pflownum+7) = y+4;
startbuspflowcb(pflownum+7) = startnode(y+4);
endbuspflowcb(pflownum+7) = endnode(y+4);
measurepflowcb(pflownum+7) = pflow(2);
measureweightpflowcb(pflownum+7) = 1e-6;
pflownum = pflownum + 7;

elseif totalnode(x) == 6
  measnum = 9;
  stationpflow(pflownum+1:pflownum+9) = x*ones(9,1);
  for y=1:total_pinj
    if buspinj(y) == x
      pinj = measurepinj(y);
    end
  end
  david = linenumdefflowcb(pflownum+1:pflownum+4);
  for z=1:4
    [C,I] = min(david);
    pflow(z) = measurepflowcb(pflownum+I);
    david(I) = 9999;
  end
  for y=1:linenum
    if station(y) == x && stat(y) == 1
      linenumpflowcb(pflownum+5) = y;
      startbuspflowcb(pflownum+5) = startnode(y);
      endbuspflowcb(pflownum+5) = endnode(y);
      measurepflowcb(pflownum+5) = pflow(1);
      measureweightpflowcb(pflownum+5) = 1e-6;
      break;
    end
  end

linenumpflowcb(pflownum+6) = y+2;
startbuspflowcb(pflownum+6) = startnode(y+2);
endbuspflowcb(pflownum+6) = endnode(y+2);
measurepflowcb(pflownum+6) = -pflow(2);
measureweightpflowcb(pflownum+6) = 1e-6;
linenumpflowcb(pflownum+7) = y+3;
startbuspflowcb(pflownum+7) = startnode(y+3);
endbuspflowcb(pflownum+7) = endnode(y+3);
measurepflowcb(pflownum+7) = pinj - pflow(1);
measureweightpflowcb(pflownum+7) = 1e-6;
linenumpflowcb(pflownum+8) = y+4;
startbuspflowcb(pflownum+8) = startnode(y+4);
endbuspflowcb(pflownum+8) = endnode(y+4);
measurepflowcb(pflownum+8) = pflow(2) + pflow(4);
measureweightpflowcb(pflownum+8) = 1e-6;
linenumpflowcb(pflownum+9) = y+5;
startbuspflowcb(pflownum+9) = startnode(y+5);
endbuspflowcb(pflownum+9) = endnode(y+5);
measurepflowcb(pflownum+9) = pflow(2);
measureweightpflowcb(pflownum+9) = 1e-6;
pflownum=pflownum+9;
elseif totalnode(x)==7
    measnum=11;
    stationpflow(pflownum+1:pflownum+11)=x*ones(11,1);
    for y=1:total_pinj
        if buspinj(y)==x
            pinj=measurepinj(y);
        end
    end
    david=linenumpflowcb(pflownum+1:pflownum+5);
    for z=1:5
        [C,I]=min(david);
        pflow(z)=measurepflowcb(pflownum+I);
        david(I)=9999;
    end
    for y=1:linenum
        if station(y)==x && stat(y)==1
            linenumpflowcb(pflownum+6)=y;
            startbuspflowcb(pflownum+6)=startnode(y);
            endbuspflowcb(pflownum+6)=endnode(y);
            measurepflowcb(pflownum+6)=pinj-pflow(3);
            measureweightpflowcb(pflownum+6)=1e-6;
            break;
        end
    end
    linenumpflowcb(pflownum+7)=y+1;
    startbuspflowcb(pflownum+7)=startnode(y+1);
    endbuspflowcb(pflownum+7)=endnode(y+1);
    measurepflowcb(pflownum+7)=pinj-pflow(3)-pflow(1);
    measureweightpflowcb(pflownum+7)=1e-6;
    linenumpflowcb(pflownum+8)=y+2;
    startbuspflowcb(pflownum+8)=startnode(y+2);
    endbuspflowcb(pflownum+8)=endnode(y+2);
    measurepflowcb(pflownum+8)=pflow(4)+pflow(5);
    measureweightpflowcb(pflownum+8)=1e-6;
    linenumpflowcb(pflownum+9)=y+3;
    startbuspflowcb(pflownum+9)=startnode(y+3);
    endbuspflowcb(pflownum+9)=endnode(y+3);
    measurepflowcb(pflownum+9)=pflow(3);
    measureweightpflowcb(pflownum+9)=1e-6;
    linenumpflowcb(pflownum+10)=y+5;
    startbuspflowcb(pflownum+10)=startnode(y+5);
    endbuspflowcb(pflownum+10)=endnode(y+5);
    measurepflowcb(pflownum+10)=pflow(4);
    measureweightpflowcb(pflownum+10)=1e-6;
    linenumpflowcb(pflownum+11)=y+7;
    startbuspflowcb(pflownum+11)=startnode(y+7);
    endbuspflowcb(pflownum+11)=endnode(y+7);
    measurepflowcb(pflownum+11)=pflow(5);
    measureweightpflowcb(pflownum+11)=1e-6;
    pflownum=pflownum+11;
elseif totalnode(x)==8
    measnum=12;
    stationpflow(pflownum+1:pflownum+13)=x*ones(13,1);
    for y=1:total_pinj
        if buspinj(y)==x
            pinj=measurepinj(y);
        end
    end
    david=linenumpflowcb(pflownum+1:pflownum+5);
    for z=1:5
        [C,I]=min(david);
        pflow(z)=measurepflowcb(pflownum+I);
        david(I)=9999;
    end
    for y=1:linenum
        if station(y)==x && stat(y)==1
            linenumpflowcb(pflownum+6)=y;
            startbuspflowcb(pflownum+6)=startnode(y);
            endbuspflowcb(pflownum+6)=endnode(y);
            measurepflowcb(pflownum+6)=pinj-pflow(3);
            measureweightpflowcb(pflownum+6)=1e-6;
            break;
        end
    end
    linenumpflowcb(pflownum+7)=y+1;
    startbuspflowcb(pflownum+7)=startnode(y+1);
    endbuspflowcb(pflownum+7)=endnode(y+1);
    measurepflowcb(pflownum+7)=pinj-pflow(3)-pflow(1);
    measureweightpflowcb(pflownum+7)=1e-6;
    linenumpflowcb(pflownum+8)=y+2;
    startbuspflowcb(pflownum+8)=startnode(y+2);
    endbuspflowcb(pflownum+8)=endnode(y+2);
    measurepflowcb(pflownum+8)=pflow(4)+pflow(5);
    measureweightpflowcb(pflownum+8)=1e-6;
    linenumpflowcb(pflownum+9)=y+3;
    startbuspflowcb(pflownum+9)=startnode(y+3);
    endbuspflowcb(pflownum+9)=endnode(y+3);
    measurepflowcb(pflownum+9)=pflow(3);
    measureweightpflowcb(pflownum+9)=1e-6;
    linenumpflowcb(pflownum+10)=y+5;
    startbuspflowcb(pflownum+10)=startnode(y+5);
    endbuspflowcb(pflownum+10)=endnode(y+5);
    measurepflowcb(pflownum+10)=pflow(4);
    measureweightpflowcb(pflownum+10)=1e-6;
    linenumpflowcb(pflownum+11)=y+7;
    startbuspflowcb(pflownum+11)=startnode(y+7);
    endbuspflowcb(pflownum+11)=endnode(y+7);
    measurepflowcb(pflownum+11)=pflow(5);
    measureweightpflowcb(pflownum+11)=1e-6;
    pflownum=pflownum+11;
    linenumpflowcb(pflownum+12)=y+9;
    startbuspflowcb(pflownum+12)=startnode(y+9);
    endbuspflowcb(pflownum+12)=endnode(y+9);
    measurepflowcb(pflownum+12)=pinj-pflow(3)-pflow(1);
    measureweightpflowcb(pflownum+12)=1e-6;
    break;
    end
end
if buspinj(y)==x
    pinj=measurepinj(y);
end
end
david=linenumpflowcb(pflownum+1:pflownum+6);
for z=1:6
    [C,I]=min(david);
    pflow(z)=measurepflowcb(pflownum+I);
    david(I)=9999;
end
for y=1:linenum
    if station(y)==x && stat(y)==1
        linenumpflowcb(pflownum+7)=y;
        startbuspflowcb(pflownum+7)=startnode(y);
        endbuspflowcb(pflownum+7)=endnode(y);
        measurepflowcb(pflownum+7)=pinj-pflow(3)-pflow(5);
        measureweightpflowcb(pflownum+7)=1e-6;
        break;
    end
    linenumpflowcb(pflownum+8)=y+1;
    startbuspflowcb(pflownum+8)=startnode(y+1);
    endbuspflowcb(pflownum+8)=endnode(y+1);
    measurepflowcb(pflownum+8)=pflow(4)+pflow(6);
    measureweightpflowcb(pflownum+8)=1e-6;
    linenumpflowcb(pflownum+10)=y+3;
    startbuspflowcb(pflownum+10)=startnode(y+3);
    endbuspflowcb(pflownum+10)=endnode(y+3);
    measurepflowcb(pflownum+10)=pflow(3);
    measureweightpflowcb(pflownum+10)=1e-6;
    linenumpflowcb(pflownum+12)=y+6;
    startbuspflowcb(pflownum+12)=startnode(y+6);
    endbuspflowcb(pflownum+12)=endnode(y+6);
    measurepflowcb(pflownum+12)=pflow(5);
    measureweightpflowcb(pflownum+12)=1e-6;
    linenumpflowcb(pflownum+13)=y+8;
    startbuspflowcb(pflownum+13)=startnode(y+8);
    endbuspflowcb(pflownum+13)=endnode(y+8);
    measurepflowcb(pflownum+13)=pflow(6);
    measureweightpflowcb(pflownum+13)=1e-6;
end
elseif totalnode(x)==9
    measnum=15;
    stationpflow(pflownum+1:pflownum+15)=x*ones(15,1);
    for y=1:total_pinj
if buspinj(y)==x
    pinj=measurepinj(y);
end
end
david=linenumpflowcb(pflownum+1:pflownum+7);
for z=1:7
    [C,I]=min(david);
    pflow(z)=measurepflowcb(pflownum+I);
    david(I)=9999;
end
for y=1:linenum
    if station(y)==x && stat(y)==1
        linenumpflowcb(pflownum+8)=y;
        startbuspflowcb(pflownum+8)=startnode(y);
        endbuspflowcb(pflownum+8)=endnode(y);
        measurepflowcb(pflownum+8)=pinj-pflow(3)-pflow(5);
        measureweightpflowcb(pflownum+8)=1e-6;
        break;
    end
end
linenumpflowcb(pflownum+9)=y+1;
startbuspflowcb(pflownum+9)=startnode(y+1);
endbuspflowcb(pflownum+9)=endnode(y+1);
measurepflowcb(pflownum+9)=pinj-pflow(3)-pflow(5)-pflow(1);
measureweightpflowcb(pflownum+9)=1e-6;
linenumpflowcb(pflownum+10)=y+2;
startbuspflowcb(pflownum+10)=startnode(y+2);
endbuspflowcb(pflownum+10)=endnode(y+2);
measurepflowcb(pflownum+10)=pflow(4)+pflow(6)+pflow(7);
measureweightpflowcb(pflownum+10)=1e-6;
linenumpflowcb(pflownum+11)=y+3;
startbuspflowcb(pflownum+11)=startnode(y+3);
endbuspflowcb(pflownum+11)=endnode(y+3);
measurepflowcb(pflownum+11)=pflow(3);
measureweightpflowcb(pflownum+11)=1e-6;
linenumpflowcb(pflownum+12)=y+5;
startbuspflowcb(pflownum+12)=startnode(y+5);
endbuspflowcb(pflownum+12)=endnode(y+5);
measurepflowcb(pflownum+12)=-pflow(4);
measureweightpflowcb(pflownum+12)=1e-6;
linenumpflowcb(pflownum+13)=y+6;
startbuspflowcb(pflownum+13)=startnode(y+6);
endbuspflowcb(pflownum+13)=endnode(y+6);
measurepflowcb(pflownum+13)=pflow(5);
measureweightpflowcb(pflownum+13)=1e-6;
linenumpflowcb(pflownum+14)=y+8;
startbuspflowcb(pflownum+14)=startnode(y+8);
endbuspflowcb(pflownum+14)=endnode(y+8);
measurepflowcb(pflownum+14)=-pflow(6);
measureweightpflowcb(pflownum+14)=1e-6;
linenumpflowcb(pflownum+15)=y+10;
startbuspflowcb(pflownum+15)=startnode(y+10);
endbuspflowcb(pflownum+15)=endnode(y+10);
measurepflowcb(pflownum+15)=-pflow(7);
measureweightpflowcb(pflownum+15)=1e-6;
pflownum=pflownum+15;
end
end

% Output the results
foutput = fopen('topoanainput.dat','w');

for x=1:total_bus
    fprintf(foutput,'%d%5d%5d',x,totalnode(x),totalout(x));
    for y=1:totalnode(x)
        fprintf(foutput,'%5d',nodecont(x,y));
    end
    fprintf(foutput,'
');
end
fprintf(foutput,'%d',-99);
for x=1:linenum
    if x<=total_line
        fprintf(foutput,'
%d%5d%5d%5d%6.3f%12.5f%12.5f%12.5f',x,
            station(x),stat(x),linetype(x),linetap(x),startbuscb(x),endbuscb(x),start
            node(x),endnode(x),linercb(x),linexc(x),concb(x),supcb(x));
    else
        fprintf(foutput,'
%d%5d%5d%5d%6d%5d%5d%5d%5d%10.5f%12.5f%12.5f%12.5f',x,
            station(x),stat(x),-1,-1,startbuscb(x),endbuscb(x),startnode(x),endnode(x),linercb(x),linexc(x),concb(x),supcb(x));
    end
    fprintf(foutput,'
\n%d',-99);
end
fprintf(foutput,'
\n\n');
fclose(foutput);

foutput = fopen('topomeasure.dat','w');

for x=1:pflownum
    fprintf(foutput,'%d%5d%5d%5d%5d%12.6f%12.6f
',1,stationpflow(x),linenumpf
    lowcb(x),startbuspflowcb(x),endbuspflowcb(x),measurepflowcb(x),measureweig
    htpflowcb(x));
end
fprintf(foutput,'%d\n',-99);

[b,busseq]=sort(buspinj);
for x=1:total_pinj
    y=busseq(x);
    fprintf(foutput,'%d%5d%5d%6.3f%12.6f%12.6f\n',3,x,x,measurepinj(y),measureweigh
    tpinj(y));
end
fprintf(foutput,'%d\n',-99);
fprintf(foutput, '\n\n');
fclose(foutput);