Section 4. Power System Matrices and Matrix Operations

Nodal equations using Kirchhoff's current law. Admittance matrix and building algorithm. Gaussian elimination. Kron reduction. LU decomposition. Formation of impedance matrix by inversion, Gaussian elimination, and direct building algorithm.

4.1 Admittance Matrix

Most power system networks are analyzed by first forming the admittance matrix. The admittance matrix is based upon Kirchhoff's current law (KCL), and it is easily formed and very sparse.

Consider the three-bus network shown in Figure 4.1 that has five branch impedances and one current source.

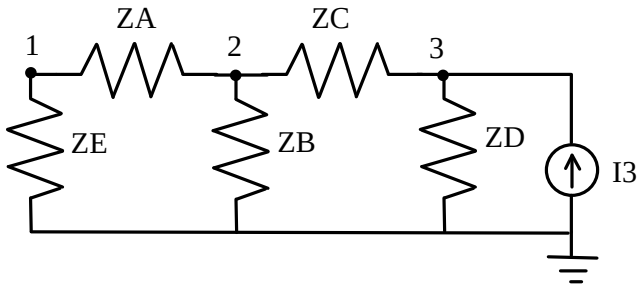

Figure 4.1: Three-Bus Network

Applying KCL at the three independent nodes yields the following equations for the bus voltages (with respect to ground):

$$\text{At bus 1, } \frac{V_1}{Z_E} + \frac{V_1 - V_2}{Z_A} = 0 \ ,$$

$$\text{At bus 2, } \frac{V_2}{Z_B} + \frac{V_2 - V_1}{Z_A} + \frac{V_2 - V_3}{Z_C} = 0 \ ,$$

$$\text{At bus 3, } \frac{V_3}{Z_D} + \frac{V_3 - V_2}{Z_C} = I_3 \ .$$

Collecting terms and writing the equations in matrix form yields

$$\begin{bmatrix} \frac{1}{Z_E}+\frac{1}{Z_A} & -\frac{1}{Z_A} & 0 \\ -\frac{1}{Z_A} & \frac{1}{Z_A}+\frac{1}{Z_B}+\frac{1}{Z_C} & -\frac{1}{Z_C} \\ 0 & -\frac{1}{Z_C} & \frac{1}{Z_C}+\frac{1}{Z_D} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ I_3 \end{bmatrix},$$

or in matrix form,

$$YV = I,$$

where *Y* is the admittance matrix, *V* is a vector of bus voltages (with respect to ground), and *I* is a vector of current injections.

Voltage sources, if present, can be converted to current sources using the usual network rules. If a bus has a zero-impedance voltage source attached to it, then the bus voltage is already known, and the dimension of the problem is reduced by one.

A simple observation of the structure of the above admittance matrix leads to the following rule for building *Y*:

1. The diagonal terms of Y contain the sum of all branch admittances connected directly to the corresponding bus.

2. The off-diagonal elements of *Y* contain the negative sum of all branch admittances connected directly between the corresponding busses.

These rules make *Y* very simple to build using a computer program. For example, assume that the impedance data for the above network has the following form, one data input line per branch:

From Bus	To Bus	Branch Impedance (Entered as Complex Numbers)
1	0	ZE
1	2	ZA
2	0	ZB
2	3	ZC
3	0	ZD

The following FORTRAN instructions would automatically build *Y*, without the need of manually writing the KCL equations beforehand:

```
      COMPLEX Y(3,3),ZB,YB
      DATA Y/9 * 0.0/
1     READ(1,*,END=2) NF,NT,ZB
```

```
      YB = 1.0 / ZB
C  MODIFY THE DIAGONAL TERMS
      IF(NF .NE. 0) Y(NF,NF) = Y(NF,NF) + YB
      IF(NT .NE. 0) Y(NT,NT) = Y(NT,NT) + YB
      IF(NF .NE. 0 .AND. NT .NE. 0) THEN
C  MODIFY THE OFF-DIAGONAL TERMS
        Y(NF,NT) = Y(NF,NT) - YB
        Y(NT,NF) = Y(NT,NF) - YB
      ENDIF
      GO TO 1
   2  STOP
      END
```

Of course, error checking is needed in an actual computer program to detect data errors and dimension overruns. Also, if bus numbers are not compressed (i.e. bus 1 through bus *N*), then additional logic is needed to internally compress the busses, maintaining separate internal and external (i.e. user) bus numbers.

Note that the Y matrix is symmetric unless there are branches whose admittance is direction-dependent. In AC power system applications, only phase-shifting transformers have this asymmetric property. The normal 30^{o} phase shift in wye-delta transformers creates asymmetry.

4.2 Gaussian Elimination and Backward Substitution

Gaussian elimination is the most common method for solving bus voltages in a circuit for which KCL equations have been written in the form

$$I = YV \quad .$$

Of course, direct inversion can be used, where

$$V = Y^{-1}I \ ,$$

but direct inversion for large matrices is computationally prohibitive or, at best, inefficient.

The objective of Gaussian elimination is to reduce the *Y* matrix to upper-right-triangular-plus-diagonal form (URT+D), then solve for *V* via backward substitution. A series of row operations (i.e. subtractions and additions) are used to change equation

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ \vdots \\ I_N \end{bmatrix} = \begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} & \cdots & y_{1,N} \\ y_{2,1} & y_{2,2} & y_{2,3} & \cdots & y_{2,N} \\ y_{3,1} & y_{3,2} & y_{3,3} & \cdots & y_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{N,1} & y_{N,2} & y_{N,3} & \cdots & y_{N,N} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ \vdots \\ V_N \end{bmatrix}$$

into

$$\begin{bmatrix} I_1 \\ I_2^{'} \\ I_3^{'} \\ \vdots \\ I_N^{'} \end{bmatrix} = \begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} & \cdots & y_{1,N} \\ 0 & y_{2,2}^{'} & y_{2,3}^{'} & \cdots & y_{2,N}^{'} \\ 0 & 0 & y_{3,3}^{'} & \cdots & y_{3,N}^{'} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & y_{N,N}^{'} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ \vdots \\ V_N \end{bmatrix},$$

in which the transformed *Y* matrix has zeros under the diagonal.

For illustrative purposes, consider the two equations represented by Rows 1 and 2, which are

$$\begin{aligned} I_1 &= y_{1,1}V_1 + y_{1,2}V_2 + y_{1,3}V_3 + \cdots + y_{1,N}V_N \\ I_2 &= y_{2,1}V_1 + y_{2,2}V_2 + y_{2,3}V_3 + \cdots + y_{2,N}V_N \end{aligned} .$$

Subtracting ($\frac{y_{2,1}}{y_{1,1}}$ • Row 1) from Row 2 yields

$$\begin{aligned} I_1 &= y_{1,1}V_1 + y_{1,2}V_2 + y_{1,3}V_3 + \cdots + y_{1,N}V_N \\ I_2 - \frac{y_{2,1}}{y_{1,1}}I_1 &= \left(y_{2,1} - \frac{y_{2,1}}{y_{1,1}}y_{1,1}\right)V_1 + \left(y_{2,2} - \frac{y_{2,1}}{y_{1,1}}y_{1,2}\right)V_2 + \left(y_{2,3} - \frac{y_{2,1}}{y_{1,1}}y_{1,3}\right)V_3 + \cdots + \left(y_{2,N} - \frac{y_{2,1}}{y_{1,1}}y_{1,N}\right)V_N \end{aligned} .$$

The coefficient of V_1 in Row 2 is forced to zero, leaving Row 2 with the desired "reduced" form of

$$I_2^{'} = 0 + y_{2,2}^{'}V_2 + y_{2,3}^{'}V_3 + \cdots + y_{2,N}^{'}V_N \ .$$

Continuing, Row 1 is then used to "zero" the V_1 coefficients in Rows 3 through N, one row at a time. Next, Row 2 is used to zero the V_2 coefficients in Rows 3 through N, and so forth.

After the Gaussian elimination is completed, and the *Y* matrix is reduced to (URT+D) form, the bus voltages are solved by backward substitution as follows:

For Row *N*,

$$I_N^{'} = y_{N,N}^{'}V_N \ , \text{ so } V_N = \frac{1}{y_{N,N}^{'}}\left(I_N^{'}\right) .$$

Next, for Row *N-1*,

$$I_{N-1}^{'} = y_{N-1,N-1}^{'}V_{N-1} + y_{N-1,N}^{'}V_N \ , \text{ so } V_{N-1} = \frac{1}{y_{N-1,N-1}^{'}}\left(I_{N-1}^{'} - y_{N-1,N}^{'}V_N\right)$$

.

Continuing for Row j, where $j = N-2, N-3, \cdots, 2$,

$$I_j' = y_{j,j}' V_j + y_{j,j+1}' V_{j+1} + \cdots + y_{j,N}' V_N \text{ , so}$$

$$V_j = \frac{1}{y_{j,j}'} \left(I_j' - y_{j,j+1}' V_{j+1} - \cdots - y_{j,N}' V_N \right),$$

which, in general form, is described by

$$V_j = \frac{1}{y_{j,j}'} \left(I_j' - \sum_{k=j+1}^{N} y_{j,k}' V_k \right).$$

A simple FORTRAN computer program for solving V in an N-dimension problem using Gaussian elimination and backward substitution is given below.

```
      COMPLEX Y(N,N),V(N),I(N),YMM
C  GAUSSIAN ELIMINATE Y AND I
      NM1    = N - 1
C  PIVOT ON ROW M, M = 1,2,3, ... ,N-1
      DO 1 M = 1,NM1
      MP1    = M + 1
      YMM    = 1.0 / Y(M,M)
C  OPERATE ON THE ROWS BELOW THE PIVOT ROW
      DO 1 J = MP1,N
C  THE JTH ROW OF I
      I(J)   = I(J) - Y(J,M) *YMM * I(M)
C  THE JTH ROW OF Y, BELOW AND TO THE RIGHT OF THE PIVOT
C  DIAGONAL
      DO 1 K = M,N
      Y(J,K) = Y(J,K) - Y(J,M) * YMM * Y(M,K)
   1  CONTINUE
C  BACKWARD SUBSTITUTE TO SOLVE FOR V
      V(N)   = I(N) / Y(N,N)
      DO 2 M = 1,NM1
      J      = N - M
C  BACKWARD SUBSTITUTE TO SOLVE FOR V, FOR
C  ROW J = N-1,N-2,N-3, ... ,1
      V(J)   = I(J)
      JP1    = J + 1
      DO 3 K = JP1,N
      V(J)   = V(J) - Y(J,K) * V(K)
```

```
3  CONTINUE
   V(J)   = V(J) / Y(J,J)
2  CONTINUE
   STOP
   END
```

One disadvantage of Gaussian elimination is that if *I* changes, even though *Y* is fixed, the entire problem must be re-solved since the elimination of *Y* determines the row operations that must be repeated on *I*. Inversion and LU decomposition to not have this disadvantage.

4.3 Kron Reduction

Gaussian elimination can be made more computationally efficient by simply *not* performing operations whose results are already known. For example, instead of arithmetically forcing elements below the diagonal to zero, simply set them to zero at the appropriate times. Similarly, instead of dividing all elements below and to the right of a diagonal element by the diagonal element, divide only the elements in the diagonal row by the diagonal element, make the diagonal element unity, and the same effect will be achieved. This technique, which is actually a form of Gaussian elimination, is known as Kron reduction.

Kron reduction "pivots" on each diagonal element $y'_{m,m}$, beginning with $y_{1,1}$, and continuing through $y_{N-1,N-1}$. Starting with Row *m* = 1, and continuing through Row *m* = *N* - *1*, the algorithm for Kron reducing $I = YV$ is

1. Divide the elements in Row *m*, that are to the right of the diagonal, by the diagonal element $y'_{m,m}$. (Note - the elements to the left of the diagonal are already zero).

2. Replace element I'_m with $\frac{I'_m}{y'_{m,m}}$.

3. Replace diagonal element $y'_{m,m}$ with unity.

4. Modify the Y' elements in rows greater than *m* and columns greater than *m* (i.e. below and to the right of the diagonal element) using

   $$y'_{j,k} = y'_{j,k} - y'_{j,m}y'_{m,k}$$, for $j > m$, $k > m$.

5. Modify the I' elements below the *mth* row according to

   $$I'_j = I'_j - y'_{j,m}I'_m$$, for $j > m$.

6. Zero the elements in Column *m* of Y' that are below the diagonal element.

A FORTRAN code for Kron reduction is given below.

```
      COMPLEX Y(N,N),V(N),I(N),YMM
C  KRON REDUCE Y, WHILE ALSO PERFORMING ROW OPERATIONS ON I
     NM1    = N - 1
C  PIVOT ON ROW M, M = 1,2,3, ... ,N-1
      DO 1 M = 1,NM1
      MP1    = M + 1
      YMM    = 1.0 / YMM
C  DIVIDE THE PIVOT ROW BY THE PIVOT
      DO 2 K = MP1,N
      Y(M,K) = Y(M,K) * YMM
   2  CONTINUE
C  OPERATE ON THE I VECTOR
      I(M)   = I(M) * YMM
C  SET THE PIVOT TO UNITY
      Y(M,M) = 1.0
C  REDUCE THOSE ELEMENTS BELOW AND TO THE RIGHT OF THE PIVOT
      DO 3 J = MP1,N
      DO 4 K = MP1,N
      Y(J,K) = Y(J,K) - Y(J,M) * Y(M,K)
   4  CONTINUE
C  OPERATE ON THE I VECTOR
      I(J)   = I(J) - Y(J,M) * I(M)
C  SET THE Y ELEMENTS DIRECTLY BELOW THE PIVOT TO ZERO
      Y(J,M) = 0.0
   3  CONTINUE
   1  CONTINUE
```

4.4 LU Decomposition

An efficient method for solving V in matrix equation $YV = I$ is to decompose Y into the product of a lower-left-triangle-plus-diagonal (LLT+D) matrix L, and an (URT+D) matrix U, so that $YV = I$ can be written as

$$LUV = I\ .$$

The benefits of decomposing Y will become obvious after observing the systematic procedure for finding V.

It is customary to set the diagonal terms of U equal to unity, so that there are a total of N^2 unknown terms in L and U. $LU = Y$ in expanded form is then

$$\begin{bmatrix} l_{1,1} & 0 & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & 0 & \cdots & 0 \\ l_{3,1} & l_{3,2} & l_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{N,1} & l_{N,2} & l_{N,3} & \cdots & l_{N,N} \end{bmatrix} \begin{bmatrix} 1 & u_{1,2} & u_{1,3} & \cdots & u_{1,N} \\ 0 & 1 & u_{2,3} & \cdots & u_{2,N} \\ 0 & 0 & 1 & \cdots & u_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} = \begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} & \cdots & y_{1,N} \\ y_{2,1} & y_{2,2} & y_{2,3} & \cdots & y_{2,N} \\ y_{3,1} & y_{3,2} & y_{3,3} & \cdots & y_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{N,1} & y_{N,2} & y_{N,3} & \cdots & y_{N,N} \end{bmatrix}.$$

Individual *l* and *u* terms are easily found by calculating them in the following order:

1. Starting from the top, work down Column 1 of *L*, finding $l_{1,1}$, then $l_{2,1}$, then $l_{3,1}$, … , $l_{N,1}$. For the special case of Column 1, these elements are $l_{j,1} = y_{j,1}$, j = 1,2,3, …,*N*.

2. Starting from the left, work across Row 1 of *U*, finding $u_{1,2}$, then $u_{1,3}$, then $u_{1,4}$, … , $u_{1,N}$. For the special case of Row 1, these elements are $u_{1,k} = \frac{y_{1,k}}{l_{1,1}}$, k = 2,3,4, …,*N*.

3. Work down Column (k = 2) of *L*, finding $l_{2,2}$, then $l_{3,2}$, then $l_{4,2}$, … , $l_{N,2}$, using

$$l_{j,k} = y_{j,k} - \sum_{m=1}^{k-1} l_{j,m} u_{m,k},\ j = k, k+1, k+2, \cdots, N \text{ , Column } k, 2 \le k \le N \text{ .}$$

4. Work across Row (k = 2) of *U*, finding $u_{2,3}$, then $u_{2,4}$, then $u_{2,5}$, … , $u_{2,N}$, using

$$u_{k,j} = \frac{y_{k,j} - \sum_{m=1}^{k-1} l_{k,m} u_{m,j}}{l_{k,k}},\ j = k+1, k+2, \cdots, N \text{ , Row } k, 2 \le k \le (N-1).$$

5. Repeat Steps 3 and 4, first for Column *k* of *L*, then for Row *k* of *U*. Continue for all *k* = 3,4,5, …,(*N*-1) for *L* and *U*, then for *k* = *N* for *L*.

The procedure given above in Steps 1 - 5 is often referred to as Crout's method. Note that elements of *L* and *U* never look "backward" for previously used elements of *Y*. Therefore, in order to conserve computer memory, *L* and *U* elements can be stored as they are calculated in the same locations at the corresponding *Y* elements. Thus, Crout's method is a memory-efficient "in situ" procedure.

An intermediate column vector is needed to find *V*. The intermediate vector *D* is defined as

D = *UV*,

so that

$$\begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_N \end{bmatrix} = \begin{bmatrix} 1 & u_{1,2} & u_{1,3} & \cdots & u_{1,N} \\ 0 & 1 & u_{2,3} & \cdots & u_{2,N} \\ 0 & 0 & 1 & \cdots & u_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ \vdots \\ V_N \end{bmatrix}.$$

Since $LUV = I$, then $LD = I$. Vector D is found using forward-substitution from

$$\begin{bmatrix} l_{1,1} & 0 & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & 0 & \cdots & 0 \\ l_{3,1} & l_{3,2} & l_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{N,1} & l_{N,2} & l_{N,3} & \cdots & l_{N,N} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_N \end{bmatrix} = \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ \vdots \\ I_N \end{bmatrix},$$

which proceeds as follows:

From Row 1, $l_{1,1}d_1 = I_1$, $d_1 = \dfrac{1}{l_{1,1}}\left(I_1\right)$,

From Row 2, $l_{2,1}d_1 + l_{2,2}d_2 = I_2$, $d_2 = \dfrac{1}{l_{2,2}}\left(I_2 - l_{2,1}d_1\right)$,

From Row k, $l_{k,1}d_1 + l_{k,2}d_2 + \cdots + l_{k,k}d_k = I_k$, $d_k = \dfrac{1}{l_{k,k}}\left(I_k - \sum_{j=1}^{k-1} l_{k,j}d_j\right)$.

Now, since $D = UV$, or

$$\begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_N \end{bmatrix} = \begin{bmatrix} 1 & u_{1,2} & u_{1,3} & \cdots & u_{1,N} \\ 0 & 1 & u_{2,3} & \cdots & u_{2,N} \\ 0 & 0 & 1 & \cdots & u_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ \vdots \\ V_N \end{bmatrix},$$

where D and U are known, then V is found using backward substitution.

An important advantage of LU decomposition over Gaussian elimination or Kron reduction is that the I vector is not modified during decomposition. Therefore, once Y has been decomposed into L and U, I can be modified, and V recalculated, with minimal work, using the forward and backward substitution steps shown above.

A special form of L is helpful when Y is symmetric. In that case, let both L and U have unity diagonal terms, and define a diagonal matrix D so that

$$Y = LDU ,$$

or

$$Y = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{2,1} & 1 & 0 & \cdots & 0 \\ l_{3,1} & l_{3,2} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ l_{N,1} & l_{N,2} & l_{N,3} & \cdots & 1 \end{bmatrix} \begin{bmatrix} d_{1,1} & 0 & 0 & \cdots & 0 \\ 0 & d_{2,2} & 0 & \cdots & 0 \\ 0 & 0 & d_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & d_{N,N} \end{bmatrix} \begin{bmatrix} 1 & u_{1,2} & u_{1,3} & \cdots & u_{1,N} \\ 0 & 1 & u_{2,3} & \cdots & u_{2,N} \\ 0 & 0 & 1 & \cdots & u_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} .$$

Since Y is symmetric, then $Y = Y^T$, and $LDU = [LDU]^T = U^T D^T L^T = U^T DL^T$. Therefore, an acceptable solution is to allow $L = U^T$. Incorporating this into the above equation yields

$$Y = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{2,1} & 1 & 0 & \cdots & 0 \\ l_{3,1} & l_{3,2} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ l_{N,1} & l_{N,2} & l_{N,3} & \cdots & 1 \end{bmatrix} \begin{bmatrix} d_{1,1} & d_{1,1}l_{2,1} & d_{1,1}l_{3,1} & \cdots & d_{1,1}l_{N,1} \\ 0 & d_{2,2} & d_{2,2}l_{3,2} & \cdots & d_{2,2}l_{N,2} \\ 0 & 0 & d_{3,3} & \cdots & d_{3,3}l_{N,3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & d_{N,N} \end{bmatrix} ,$$

which can be solved by working from top-to-bottom, beginning with Column 1 of Y, as follows:

Working down Column 1 of Y,

$$y_{1,1} = d_{1,1} ,$$

$$y_{2,1} = l_{2,1}d_{1,1}, \text{ so } l_{2,1} = y_{2,1} / d_{1,1} ,$$

$$y_{j,1} = l_{j,1}d_{1,1}, \text{ so } l_{j,1} = y_{j,1} / d_{1,1} .$$

Working down Column 2 of Y,

$$y_{2,2} = l_{2,1}d_{1,1}l_{2,1} + d_{2,2}, \text{ so } d_{2,2} = y_{2,2} - l_{2,1}d_{1,1}l_{2,1} ,$$

$$y_{j,2} = l_{j,1}d_{1,1}l_{2,1} + l_{j,2}d_{2,2}, \text{ so } l_{j,2} = \left(y_{j,2} - l_{j,1}d_{1,1}l_{2,1}\right) / d_{2,2}, \ 2 < j \le N .$$

Working down Column k of Y,

$$y_{k,k} = \sum_{m=1}^{k-1} l_{k,m} d_{m,m} l_{k,m} + d_{k,k}, \text{ so } d_{k,k} = y_{k,k} - \sum_{m=1}^{k-1} l_{k,m} d_{m,m} l_{k,m} ,$$

$$y_{j,k} = \sum_{m=1}^{k-1} l_{j,m} d_{m,m} l_{k,m}, \text{ so } l_{j,k} = \left(y_{j,k} - \sum_{m=1}^{k-1} l_{j,m} d_{m,m} l_{k,m} \right) / d_{k,k}, \; k < j \le N$$

.

This simplification reduces memory and computation requirements for LU decomposition by approximately one-half.

4.5 Bifactorization

Bifactorization recognizes the simple pattern that occurs when doing "in situ" LU decomposition. Consider the first four rows and columns of a matrix that has been LU decomposed according to Crout's method:

$$\begin{bmatrix} l_{1,1} = y_{1,1} & u_{1,2} = y_{1,2} / l_{1,1} & u_{1,3} = y_{1,3} / l_{1,1} & u_{1,4} = y_{1,4} / l_{1,1} \\ l_{2,1} = y_{2,1} & l_{2,2} = y_{2,2} - l_{2,1} u_{1,2} & u_{2,3} = \left(y_{2,3} - l_{2,1} u_{1,3} \right) / l_{2,2} & u_{2,4} = \left(y_{2,4} - l_{2,1} u_{1,4} \right) / l_{2,2} \\ l_{3,1} = y_{3,1} & l_{3,2} = y_{3,2} - l_{3,1} u_{1,2} & l_{3,3} = y_{3,3} - l_{3,1} u_{1,3} - l_{3,2} u_{2,3} & u_{3,4} = \left(y_{3,4} - l_{3,1} u_{1,4} - l_{3,2} u_{2,4} \right) / l_{3,3} \\ l_{4,1} = y_{4,1} & l_{4,2} = y_{4,2} - l_{4,1} u_{1,2} & l_{4,3} = y_{4,3} - l_{4,1} u_{1,3} - l_{4,2} u_{2,3} & l_{4,4} = y_{4,4} - l_{4,1} u_{1,4} - l_{4,2} u_{2,4} - l_{4,3} u_{3,4} \end{bmatrix} .$$

The pattern developed is very similar to Kron reduction, and it can be expressed in the following steps:

1. Beginning with Row 1, divide the elements to the right of the pivot element, and in the pivot row, by $l_{1,1}$, so that

   $$y'_{1,k} = \frac{y_{1,k}}{y_{1,1}} \text{ , for } k = 2,3,4, \ldots, N.$$

2. Operate on the elements below and to the right of the pivot element using

   $$y'_{j,k} = y_{j,k} - y_{j,1} y'_{1,k} \text{ , for } j = 2,3,4, \ldots, N, \; k = 2,3,4, \ldots, N.$$

3. Continue for pivots $m = 2,3,4, \ldots, (N - 1)$ using

   $$y'_{m,k} = \frac{y'_{m,k}}{y'_{m,m}}, k = m+1, m+2, \cdots, N \text{ ,}$$

   followed by

   $$y'_{j,k} = y'_{j,k} - y'_{j,m} y'_{m,k}, j = m+1, m+2, \cdots, N; k = m+1, m+2, \cdots, N \text{ ,}$$

for each pivot *m*.

When completed, matrix *Y'* has been replaced by matrices *L* and *U* as follows:

$$Y' = \begin{bmatrix} l_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,5} \\ l_{2,1} & l_{2,2} & u_{2,3} & \cdots & u_{2,5} \\ l_{3,1} & l_{3,2} & l_{3,3} & \cdots & u_{3,5} \\ \vdots & \vdots & \vdots & \ddots & \\ l_{N,1} & l_{N,2} & l_{N,3} & \cdots & l_{N,N} \end{bmatrix},$$

and where the diagonal *u* elements are unity (i.e. $u_{1,1} = u_{2,2} = u_{3,3} = \cdots = u_{N,N} = 1$).

The corresponding FORTRAN code for bifactorization is

```
      COMPLEX Y(N,N),YMM
C  DO FOR EACH PIVOT M = 1,2,3, ... ,N - 1
      NM1     = N - 1
      DO 1 M  = 1,NM1
C  FOR THE PIVOT ROW
      MP1     = M + 1
      YMM     = 1.0 / Y(M,M)
      DO 2 K  = MP1,N
      Y(M,K)  = Y(M,K) * YMM
   2  CONTINUE
C  BELOW AND TO THE RIGHT OF THE PIVOT ELEMENT
      DO 3 J   = MP1,N
      DO 3 K   = MP1,N
      Y(J,K)   = Y(J,K) - Y(J,M) * Y(M,K)
   3  CONTINUE
   1  CONTINUE
      STOP
      END
```

4.6 Shipley-Coleman Inversion

For relatively small matrices, it is possible to obtain the inverse directly. The Shipley-Coleman inversion method for inversion is popular because it is easily programmed. The algorithm is

1. For each pivot (i.e. diagonal term) *m*, *m* = 1,2,3, … ,*N*, perform the following Steps 2 - 4.

2. Kron reduce all elements in *Y*, above and below, except those in Column *m* and Row *m* using

$$y'_{j,k} = y'_{j,k} - \frac{y'_{j,m} y'_{m,k}}{y_{m,m}},\ j = 1,2,3,\cdots,N,\ j \neq m; k = 1,2,3,\cdots,N, k \neq m\ .$$

3. Replace pivot element $y'_{m,m}$ with its negative inverse, i.e. $\frac{-1}{y'_{m,m}}$.

4. Multiply all elements in Row *m* and Column *m*, except the pivot, by $y'_{m,m}$.

The result of this procedure is actually the negative inverse, so that when completed, all terms must be negated. A FORTRAN code for Shipley-Coleman is shown below.

```
      COMPLEX Y(N,N),YPIV
C  DO FOR EACH PIVOT M = 1,2,3, ... ,N
      DO 1 M  = 1,N
      YPIV    = 1.0 / Y(M,M)
C  KRON REDUCE ALL ROWS AND COLUMNS, EXCEPT THE PIVOT ROW
C  AND PIVOT COLUMN
      DO 2 J  = 1,N
      IF(J .EQ. M) GO TO 2
      DO 2 K  = 1,N
      IF(K. NE. M) Y(J,K) = Y(J,K) - Y(J,M) * Y(M,K) * YPIV
   2  CONTINUE
C  INVERT THE PIVOT ELEMENT AND NEGATE IT
      YPIV     = -YPIV
      Y(M,M)   = YPIV
C  WORK ACROSS THE PIVOT ROW AND DOWN THE PIVOT COLUMN,
C  MULTIPLYING BY THE NEW PIVOT VALUE
      DO 3 K   = 1,N
      IF(K .EQ. M) GO TO 3
      Y(M,K) = Y(M,K) * YPIV
      Y(K,M) = Y(K,M) * YPIV
   3  CONTINUE
   1  CONTINUE
C  NEGATE THE RESULTS
      DO 4 J   = 1,N
      DO 4 K   = 1,N
      Y(J,K)   = -Y(J,K)
   4  CONTINUE
      STOP
      END
```

The order of the number of calculations for Shipley-Coleman is N^3 .

4.7 Impedance Matrix

The impedance matrix is the inverse of the admittance matrix, or

$$Z = Y^{-1} ,$$

so that

$$V = ZI \ .$$

The reference bus both *Y* and *Z* is ground. Although the impedance matrix can be found via inversion, complete inversion is not common for matrices with more than a few hundred rows and columns because of the matrix storage requirements. In those instances, *Z* elements are usually found via Gaussian elimination, Kron reduction, or, less commonly, by a direct building algorithm. If only a few of the *Z* elements are needed, then Gaussian elimination or Kron reduction are best. Both methods are described in following sections.

4.8 Physical Significance of Admittance and Impedance Matrices

The physical significance of the admittance and impedance matrices can be seen by examining the basic matrix equations $I = YV$ and $V = Y^{-1}I = ZI$. Expanding the *jth* row of $I = YV$ yields $I_j = \sum_{k=1}^{N} y_{j,k} V_k$. Therefore,

$$y_{j,k} = \left. \frac{I_j}{V_k} \right|_{V_m = 0, m = 1,2,\cdots,N, m \neq k} ,$$

where, as shown in Figure 4.2, all busses except *k* are grounded, V_k is a voltage source attached to bus *k*, and I_j is the resulting current injection at (i.e. flowing into) bus *j*. Since all busses that neighbor bus *k* are grounded, the currents induced by V_k will not get past these neighbors, and the only non-zero injection currents I_j will occur at the neighboring busses. In a large power system, most busses do not neighbor any arbitrary bus *k*. Therefore, *Y* consists mainly of zeros (i.e. is sparse) in most power systems.

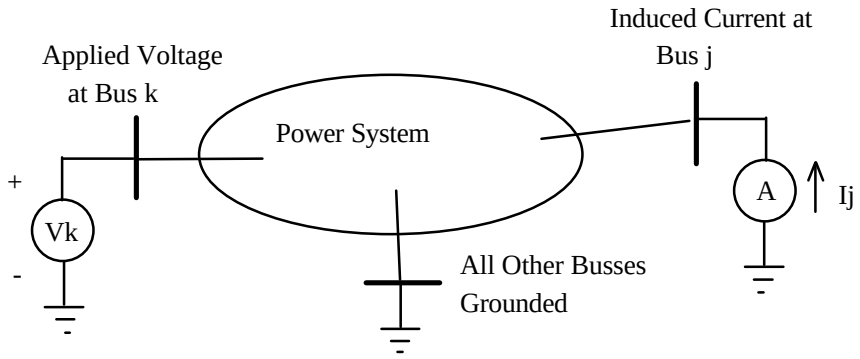

Figure 4.2: Measurement of Admittance Matrix Term $y_{j,k}$

Concerning *Z*, the *kth* row of $V = Y^{-1}I = ZI$ yields $V_k = \sum_{j=1}^{N} z_{k,j} I_j$. Hence,

$$z_{j,k} = \left.\frac{V_j}{I_k}\right|_{I_m=0, m=1,2,\cdots,N, m\neq k} ,$$

where, as shown in Figure 4.3, I_k is a current source attached to bus *k*, V_j is the resulting voltage at bus *j*, and all busses except *k* are open-circuited. Unless the network is disjoint, then current injection at one bus, when all other busses open-circuited, will raise the potential everywhere in the network. For that reason, *Z* tends to be full.

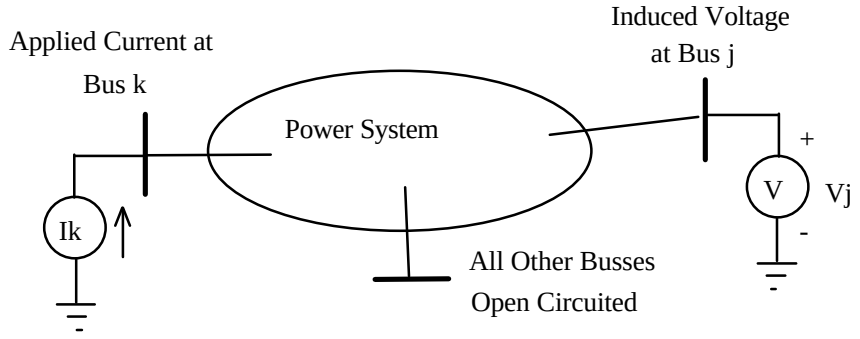

Figure 4.3: Measurement of Impedance Matrix Term $z_{j,k}$

4.9 Formation of the Impedance Matrix via Gaussian Elimination, Kron Reduction, and LU Decomposition

4.9.1 Gaussian Elimination

An efficient method of fully or partially inverting a matrix is to formulate the problem using Gaussian elimination. For example, given

$$YZ = I ,$$

where Y is a matrix of numbers, Z is a matrix of unknowns, and I is the identity matrix, the objective is to Gaussian eliminate Y, while performing the same row operations on I, to obtain the form

$$Y'Z = I' ,$$

where Y' is in (URT+D) form. Then, individual columns of Z can then be solved using backward substitution, one at a time. This procedure is also known as the augmentation method, and it is illustrated as follows:

Y is first Gaussian eliminated, as shown in a previous section, while at the same time performing identical row operations on I. Afterward, the form of $Y'Z = I'$ is

$$\begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} & \cdots & y_{1,N} \\ 0 & y'_{2,2} & y'_{2,3} & \cdots & y'_{2,N} \\ 0 & 0 & y'_{3,3} & \cdots & y'_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & y'_{N,N} \end{bmatrix} \begin{bmatrix} z_{1,1} & z_{1,2} & z_{1,3} & \cdots & z_{1,N} \\ z_{2,1} & z_{2,2} & z_{2,3} & \cdots & z_{2,N} \\ z_{3,1} & z_{3,2} & z_{3,3} & \cdots & z_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ z_{N,1} & z_{N,2} & z_{N,3} & \cdots & z_{N,N} \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ I'_{2,1} & I'_{2,2} & 0 & \cdots & 0 \\ I'_{3,1} & I'_{3,2} & I'_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ I'_{N,1} & I'_{N,2} & I'_{N,3} & \cdots & I'_{N,N} \end{bmatrix}$$

where Rows 1 of Y' and I' are the same as in Y and I. The above equation can be written in abbreviated column form as

$$\begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} & \cdots & y_{1,N} \\ 0 & y'_{2,2} & y'_{2,3} & \cdots & y'_{2,N} \\ 0 & 0 & y'_{3,3} & \cdots & y'_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & y'_{N,N} \end{bmatrix} \begin{bmatrix} Z_1 & Z_2 & Z_3 & \cdots & Z_N \end{bmatrix} = \begin{bmatrix} I'_1 & I'_2 & I'_3 & \cdots & I'_N \end{bmatrix} ,$$

where the individual column vectors Z_i and I_i have dimension N x 1. The above equation can be solved as N separate subproblems, where each subproblem computes a column of Z. For example, the kth column of Z can be computed by applying backward substitution to

$$\begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} & \cdots & y_{1,N} \\ 0 & y'_{2,2} & y'_{2,3} & \cdots & y'_{2,N} \\ 0 & 0 & y'_{3,3} & \cdots & y'_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & y'_{N,N} \end{bmatrix} \begin{bmatrix} Z_{1,k} \\ Z_{2,k} \\ Z_{3,k} \\ \vdots \\ Z_{N,k} \end{bmatrix} = \begin{bmatrix} I'_{1,k} \\ I'_{2,k} \\ I'_{3,k} \\ \vdots \\ I'_{N,k} \end{bmatrix}.$$

Each column of *Z* is solved independently from the others.

4.9.2 Kron Reduction

If Kron reduction, the problem is essentially the same, except that Row 1 of the above equation is divided by $y'_{1,1}$, yielding

$$\begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} & \cdots & y_{1,N} \\ 0 & y'_{2,2} & y'_{2,3} & \cdots & y'_{2,N} \\ 0 & 0 & y'_{3,3} & \cdots & y'_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & y'_{N,N} \end{bmatrix} \begin{bmatrix} Z_{1,k} \\ Z_{2,k} \\ Z_{3,k} \\ \vdots \\ Z_{N,k} \end{bmatrix} = \begin{bmatrix} I'_{1,k} \\ I'_{2,k} \\ I'_{3,k} \\ \vdots \\ I'_{N,k} \end{bmatrix}.$$

Because backward substitution can stop when the last desired *z* element is computed, the process is most efficient if the busses are ordered so that the highest bus numbers (i.e. *N*, *N-1*, *N-2*, etc.) correspond to the desired *z* elements. Busses can be ordered accordingly when forming *Y* to take advantage of this efficiency.

4.9.3 LU Decomposition

Concerning LU decomposition, once the *Y* matrix has been decomposed into *L* and *U*, then we have

$$YZ = LUZ = I,$$

where *I* is the identity matrix. Expanding the above equation as $L[UZ] = I$ yields

$$\begin{bmatrix} l_{1,1} & 0 & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & 0 & \cdots & 0 \\ l_{3,1} & l_{2,3} & l_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{N,1} & l_{N,2} & l_{N,3} & \cdots & l_{N,N} \end{bmatrix} \begin{bmatrix} uz_{1,1} & uz_{1,2} & uz_{1,3} & \cdots & uz_{1,N} \\ uz_{2,1} & uz_{2,2} & uz_{2,3} & \cdots & uz_{2,N} \\ uz_{3,1} & uz_{3,2} & uz_{3,3} & \cdots & uz_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ uz_{N,1} & uz_{N,2} & uz_{N,3} & \cdots & uz_{N,N} \end{bmatrix} = .$$

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

The special structure of the above equation shows that, in general, *UZ* must be (LLT+D) in form. *UZ* can be found by using forward substitution on the above equation, and then *Z* can be found, one column at a time, by using backward substitution on $UZ = [U][Z]$, which in expanded form is

$$\begin{bmatrix} 1 & u_{1,2} & u_{1,3} & \cdots & u_{1,N} \\ 0 & 1 & u_{2,3} & \cdots & u_{2,N} \\ 0 & 0 & 1 & \cdots & u_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} z_{1,1} & z_{1,2} & z_{1,3} & \cdots & z_{1,N} \\ z_{2,1} & z_{2,2} & z_{2,3} & \cdots & z_{2,N} \\ z_{3,1} & z_{3,2} & z_{3,3} & \cdots & z_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ z_{N,1} & z_{N,2} & z_{N,3} & \cdots & z_{N,N} \end{bmatrix} =$$

$$\begin{bmatrix} uz_{1,1} & 0 & 0 & \cdots & 0 \\ uz_{2,1} & uz_{2,2} & 0 & \cdots & 0 \\ uz_{3,1} & uz_{3,2} & uz_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ uz_{N,1} & uz_{N,2} & uz_{N,3} & \cdots & uz_{N,N} \end{bmatrix}.$$

4.10 Direct Impedance Matrix Building and Modification Algorithm

The impedance matrix can be built directly without need of the admittance matrix, if the physical properties of *Z* are exploited. Furthermore, once *Z* has been built, no matter which method was used, it can be easily modified when network changes occur. This after-the-fact modification capability is the most important feature of the direct impedance matrix building and modification algorithm.

The four cases to be considered in the algorithm are

Case 1. Add a branch impedance between a new bus and the reference bus.

Case 2. Add a branch impedance between a new bus and an existing non-reference bus.

Case 3. Add a branch impedance between an existing bus and the reference bus.

Case 4. Add a branch impedance between two existing non-reference busses.

Direct formation of an impedance matrix, plus all network modifications, can be achieved through these four cases. Each of the four is now considered separately.

4.10.1 Case 1, Add a Branch Impedance Between a New Bus and the Reference Bus

Case 1 is applicable in two situations. First, it is the starting point when building a new impedance matrix. Second, it provides (as does Case 2) a method for adding new busses to an existing impedance matrix.

Both situations applicable to Case 1 are shown in Figure 4.4. The starting-point situation is on the left, and the new-bus situation is on the right.

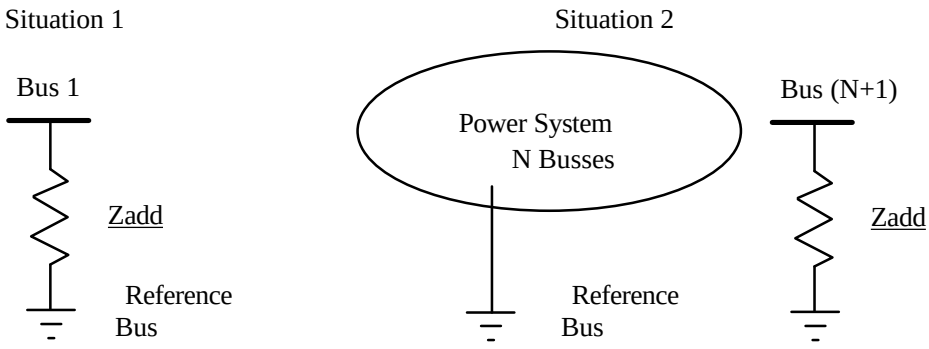

Figure 4.4: Case 1, Add a Branch Impedance Between a New Bus and the Reference Bus

The impedance matrices for the two situations are

Situation 1 $Z_{1,1} = \underline{Zadd}$,

Situation 2: $$Z_{N+1,N+1} = \begin{bmatrix} \left[Z_{N,N}\right]_{NxN} & \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{Nx1} \\ \left[0 \quad 0 \quad \cdots \quad 0\right]_{1xN} & \left[\underline{Zadd}\right]_{1x1} \end{bmatrix}.$$

The effect of situation 2 is simply to augment the existing $Z_{N,N}$ by a column of zeros, a row of zeros, and a new diagonal element ***Zadd***. New bus (*N+1*) is isolated from the rest of the system.

4.10.2 Case 2, Add a Branch Impedance Between a New Bus and an Existing Non-Reference Bus

Consider Case 2, shown in Figure 4.5, where a branch impedance ***Zadd*** is added from existing non-reference bus *j* to new bus (*N+1*). Before the addition, the power system has *N* busses, plus a reference bus (which is normally ground), and an impedance matrix $Z_{N,N}$.

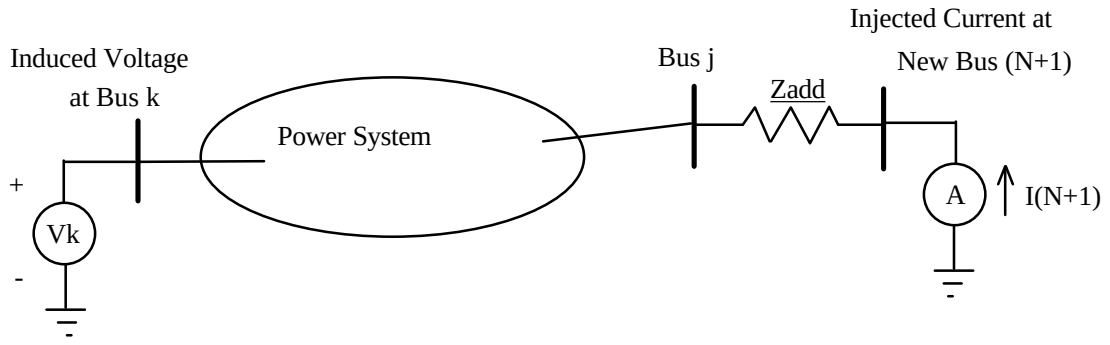

Figure 4.5: Case 2, Add a Branch Impedance Between New Bus (*N+1*) and Existing Non-Reference Bus j

Since all of the current I_{N+1} injected at new bus (*N+1*) flows through ***Zadd*** and into Bus *j*, the original *N* power system busses cannot distinguish between current injections I_j and I_{N+1}. Therefore,

$$\frac{\partial V_k}{\partial I_{N+1}} = \frac{\partial V_k}{\partial I_j}, k \neq (N+1) \ ,$$

meaning that impedance matrix elements $z_{k,j}$ and $z_{k,N+1}$ are identical for busses $k \neq (N+1)$, and that the effect on the impedance matrix is to augment it with an additional Row (*N+1*) that is identical to Row *j*.

Likewise, since bus (*N+1*) is an open-circuited radial bus stemming from bus *j*, a current injected at another bus *k* creates identical voltage changes on busses *j* and (*N+1*). This means that

$$\frac{\partial V_{N+1}}{\partial I_k} = \frac{\partial V_j}{\partial I_k}, k \neq (N+1) \ ,$$

so that the effect on the impedance matrix is to augment it with an additional Column (*N+1*) that is identical to Column *j*.

Next, due to the fact that all of injection current $I_{N+1,N+1}$ passes through new branch impedance ***Zadd***, the relationship between V_j and V_{N+1} is

$$\frac{\partial V_{N+1}}{\partial I_{N+1}} = \frac{\partial V_j}{\partial I_{N+1}} + \underline{Zadd} \ ,$$

so that the (*N+1*) diagonal term of the impedance matrix becomes

$$z_{N+1,N+1} = z_{j,j} + \underline{Zadd} \ ,$$

where $z_{j,j}$ is the *jth* diagonal term of Z_{NxN} .

Summarizing, the total effect of Case 2 is to increase the dimension of the impedance matrix by one row and column according to

$$Z_{N+1,N+1} = \begin{bmatrix} \lfloor Z_{N,N} \rfloor_{NxN} & \lfloor \text{Column } j \text{ of } Z_{N,N} \rfloor_{Nx1} \\ \lfloor \text{Row } j \text{ of } Z_{N,N} \rfloor_{1xN} & \lfloor z_{j,j} + \underline{Zadd} \rfloor_{1x1} \end{bmatrix}.$$

4.10.3 Case 3, Add a Branch Impedance Between an Existing Bus and the Reference Bus

Now, consider Case 3, where a new impedance-to-reference tie *Zadd* is added to existing Bus *j*. The case is handled as a two-step extension of Case 2, as shown in Figure 4.6. First, extend *Zadd* from Bus *j* to fictitious Bus (*N+1*), as was done in Case 2. Then, tie fictitious Bus (*N+1*) to the reference bus, and reduce the size of the augmented impedance matrix back to (*N* x *N*).

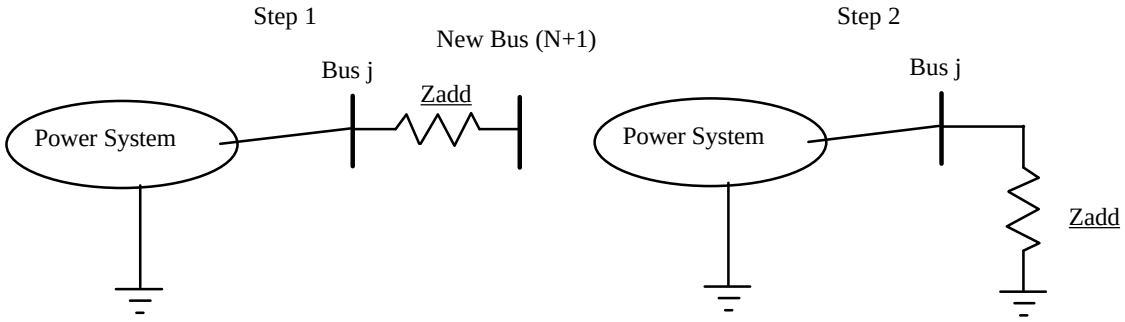

Figure 4.6: Case 3. Two-Step Procedure for Adding Branch Impedance *Zadd* from Existing Bus *j* to the Reference Bus

Step 1 creates the augmented $Z_{N+1,N+1}$ matrix shown in Case 2. The form of equation $V_{N+1} = Z_{N+1,N+1} I_N$ is

$$\begin{bmatrix} V_{Nx1} \\ V_{REF} = 0 \end{bmatrix} = \begin{bmatrix} \lfloor Z_{N,N} \rfloor_{NxN} & \lfloor \text{Column } j \text{ of } Z_{N,N} \rfloor_{Nx1} \\ \lfloor \text{Row } j \text{ of } Z_{N,N} \rfloor_{1xN} & \lfloor z_{j,j} + \underline{Zadd} \rfloor_{1x1} \end{bmatrix} \begin{bmatrix} I_{Nx1} \\ I_{REF} \end{bmatrix},$$

where V_{REF}, $z_{j,j} + \underline{Zadd}$, and I_{REF} are scalars. Defining the row and column vectors as R_j and C_j, respectively, yields

$$\begin{bmatrix} V_{Nx1} \\ V_{REF} = 0 \end{bmatrix} = \begin{bmatrix} \lfloor Z_{N,N} \rfloor_{NxN} & \lfloor C_j \rfloor_{Nx1} \\ \lfloor R_j \rfloor_{1xN} & \lfloor z_{j,j} + \underline{Zadd} \rfloor_{1x1} \end{bmatrix} \begin{bmatrix} I_{Nx1} \\ I_{REF} \end{bmatrix}.$$

At this point, scalar I_{REF} can be eliminated by expanding the bottom row to obtain

$$I_{REF} = -\frac{\lfloor R_j I_{Nx1} \rfloor_{1x1}}{z_{j,j} + \underline{Zadd}}.$$

Substituting into the top *N* rows yields

$$V_{Nx1} = Z_{NxN} I_{Nx1} - \frac{1}{z_{j,j} + \underline{Zadd}} \left(C_j R_j\right) I_{Nx1} = \left(Z_{NxN} - \frac{1}{z_{j,j} + \underline{Zadd}} \left(C_j R_j\right) \right) I_{Nx1} \ ,$$

or

$$V_{Nx1} = Z'_{NxN} I_{Nx1} \ .$$

Expanding $C_j R_j$ gives

$$\begin{bmatrix} z_{1,j} \\ z_{2,j} \\ z_{3,j} \\ \vdots \\ z_{N,j} \end{bmatrix} \begin{bmatrix} z_{j,1} & z_{j,2} & z_{j,3} & \cdots & z_{j,N} \end{bmatrix} = \begin{bmatrix} z_{1,j}z_{j,1} & z_{1,j}z_{j,2} & z_{1,j}z_{j,3} & \cdots & z_{1,j}z_{j,N} \\ z_{2,j}z_{j,1} & z_{2,j}z_{j,2} & z_{2,j}z_{j,3} & \cdots & z_{2,j}z_{j,N} \\ z_{3,j}z_{j,1} & z_{3,j}z_{j,2} & z_{3,j}z_{j,3} & \cdots & z_{3,j}z_{j,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ z_{N,j}z_{j,1} & z_{N,j}z_{j,2} & z_{N,j}z_{j,3} & \cdots & z_{N,j}z_{j,N} \end{bmatrix} ,$$

so that the individual elements of the modified impedance matrix Z'_{Nx1} can be written as

$$z'_{i,k} = z_{i,k} - \frac{z_{i,j}z_{j,k}}{z_{j,j} + \underline{Zadd}}, i = 1,2,3,\cdots,N; k = 1,2,3,\cdots,N \ .$$

Note that the above equation corresponds to Kron reducing the augmented impedance matrix, using element $z_{N+1,N+1}$ as the pivot.

4.10.4 Case 4, Add a Branch Impedance Between Two Existing Non-Reference Busses

The final case to be considered is that of adding a branch between existing busses *j* and *k* in an *N*-bus power system that has impedance matrix $Z_{N,N}$. The system and branch are shown in Figure 4.7.

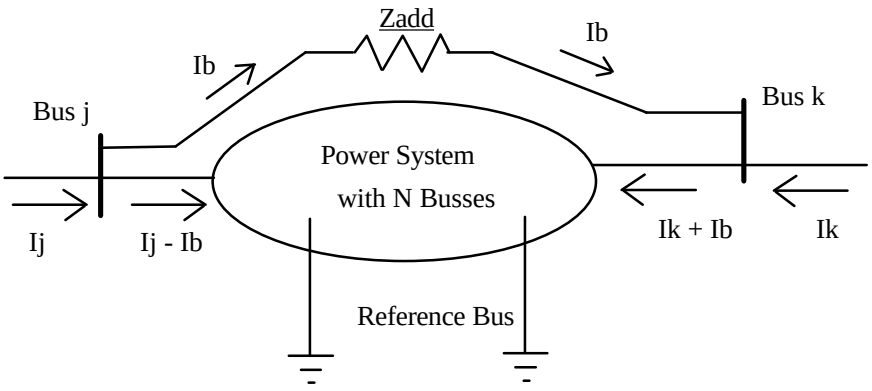

Figure 4.7: Case 4, Add Branch Impedance $\underline{Zadd}$ between Existing Busses *j* and *k*

As seen in the figure, the actual current injected into the system via Bus *j* is $I_j - I_b$, and the actual injection via Bus *k* is $I_k + I_b$, where

$$I_b = \frac{V_j - V_k}{\underline{Zadd}} \text{ , or } V_j - V_k = (\underline{Zadd})I_b \text{ .}$$

The effect of the branch addition on the voltage at arbitrary bus *m* can be found by substituting the true injection currents at busses *j* and *k* into $V = Z_{N,N} I$, yielding

$$V_m = z_{m,1}I_1 + z_{m,2}I_2 + z_{m,3}I_3 + \cdots z_{m,j}(I_j - I_b) + \cdots + z_{m,k}(I_k + I_b) + \cdots + z_{m,N}I_N \text{ ,}$$

or

$$V_m = z_{m,1}I_1 + z_{m,2}I_2 + z_{m,3}I_3 + \cdots z_{m,j}I_j + \cdots + z_{m,k}I_k + \cdots + z_{m,N}I_N - (z_{m,j} - z_{m,k})I_b \text{ .}$$

For busses *j* and *k* specifically,

$$\begin{aligned} V_j &= z_{j,1}I_1 + z_{j,2}I_2 + z_{j,3}I_3 + \cdots + z_{j,j}(I_j - I_b) + \cdots + z_{j,k}(I_k + I_b) + \cdots + z_{j,N}I_N \\ V_k &= z_{k,1}I_1 + z_{k,2}I_2 + z_{k,3}I_3 + \cdots + z_{k,j}(I_j - I_b) + \cdots + z_{k,k}(I_k + I_b) + \cdots + z_{k,N}I_N \end{aligned} \text{ .}$$

Then, $V_j - V_k$ gives

$$V_j - V_k = (z_{j,1} - z_{k,1})I_1 + (z_{j,2} - z_{k,2})I_2 + (z_{j,3} - z_{k,3})I_3 + \cdots + (z_{j,j} - z_{k,j})(I_j - I_b) + \cdots$$
$$(z_{j,k} - z_{k,k})(I_k + I_b) + \cdots + (z_{j,N} - z_{k,N})I_N \ .$$

Substituting in $V_j - V_k = (\underline{Zadd})I_b$ and combining terms then yields

$$0 = (z_{j,1} - z_{k,1})I_1 + (z_{j,2} - z_{k,2})I_2 + (z_{j,3} - z_{k,3})I_3 + \cdots + (z_{j,j} - z_{k,j})I_j + \cdots$$
$$(z_{j,k} - z_{k,k})I_k + \cdots + (z_{j,N} - z_{k,N})I_N - (z_{j,j} + z_{k,k} - z_{j,k} - z_{k,j} + \underline{Zadd})I_b \ .$$

All of the above effects can be included as an additional row and column in equation $V = Z_{N,N} I$ as follows:

$$\begin{bmatrix} [V_N]_{Nx1} \\ [0]_{Nx1} \end{bmatrix} = \begin{bmatrix} [Z_{N,N}]_{NxN} & [\text{Col } j \text{ of } Z_{N,N} - \text{Col } k \text{ of } Z_{N,N}]_{Nx1} \\ [\text{Row } j \text{ of } Z_{N,N} - \text{Row } k \text{ of } Z_{N,N}]_{1xN} & [z_{j,j} + z_{k,k} - z_{j,k} - z_{k,j} + \underline{Zadd}]_{Nx1} \end{bmatrix} \cdot$$
$$\begin{bmatrix} [I_N]_{Nx1} \\ [-I_b]_{1x1} \end{bmatrix}.$$

As was done in Case 3, the effect of the augmented row and column of the above equation can be incorporated into a modified impedance matrix by using Kron reduction, where element $z_{N+1,N+1}$ is the pivot, yielding

$$z'_{i,m} = z_{i,m} - \frac{(z_{i,j} - z_{i,k})(z_{j,m} - z_{k,m})}{z_{j,j} + z_{k,k} - z_{j,k} - z_{k,j} + \underline{Zadd}} \ .$$

4.10.5 Application Notes

Once an impedance matrix is built, no matter which method is used, the modification algorithm can very easily adjust *Z* for network changes. For example, a branch outage can be effectively achieved by placing an impedance, of the same but negative value, in parallel with the actual impedance, so that the impedance of the parallel combination is infinite.

4.11 Example Code YZBUILD for Building Admittance Matrix and Impedance Matrix

```
c
c  Program YZBUILD for EE394J.  Reads bdat and ldat (positive
c  sequence) files in PCFLO fixed column format.  Builds the
c  admittance and impedance matrices.  Impedance matrix is
c  produced in two ways - lu decomposition (zbus.lu), and gaussian
c  elimination (zbus.gau).
c
c  YZBUILD is intended for use with positive sequence short
c  circuit BDAT and LDAT files.  Therefore, positive sequence
c  subtransient impedances should be included in BDAT.
```

```
c
c  Mack Grady.  See date below in variable versio
c
      character*6 versio/'981025'/
      complex ybus,yline,ycap,ctap,yii,ykk,yik,yki,zsub
      complex lmat,umat,unity,czero,ytest,ysave,diff,uz,
     1 uzelem,zbus,elem,alpha
c
c  Dimensioned here for up to 100 busses
c
      dimension ybus(100,100),nbext(100),nbint(9999)
      dimension lmat(100,100),umat(100,100),ysave(100,100),
     1 uz(100,100),zbus(100,100),unity(100,100)
      data mxint/100/,mxext/9999/,nbus/0/, nbint/9999 * 0/, iout/17/,
     1 nbext/100 * 0/,czero /(0.0,0.0)/,eps4/1.0e-04/,
     2 eps6/1.0e-06/,eps9/1.0e-09/
      data ybus  /10000 * (0.0,0.0)/
      data zbus  /10000 * (0.0,0.0)/
      data uz    /10000 * (0.0,0.0)/
      data unity /10000 * (0.0,0.0)/
c
      pi = 4.0 * atan(1.0)
      dr = 180.0 / pi
      open(unit=2,file='ybus.yzb')
      open(unit=1,file='bdat.yzb')
      open(unit=17,file='exlog.yzb')
      write(6,*) 'Program YZBUILD, Version ',versio
      write(2,*) 'Program YZBUILD, Version ',versio
      write(iout,*) 'Program YZBUILD, Version ',versio
      write(6,*) 'Read bus data from PCFLO fixed column format ',
     1 'file BDAT.YZB'
      write(6,*) 'number        q shunt      rsubtrans',
     1 '      xsubtrans'
      write(2,*) 'Read bus data from PCFLO fixed column format ',
     1 'file BDAT.YZB'
      write(2,*) 'number        q shunt      rsubtrans',
     1 '      xsubtrans'
      write(iout,*) 'Read bus data from PCFLO fixed column format ',
     1 'file BDAT.YZB'
      write(iout,*) 'number        q shunt      rsubtrans',
     1 '      xsubtrans'
c
c  read the bdat file
c
    1 read(1,1001,end=5) nb,qshunt,rsub,xsub
 1001 format(i4,t58,f8.0,t90,2f8.0)
      if(nb .eq. 0) go to 5
      write(6,1002) nb,qshunt,rsub,xsub
      write(2,1002) nb,qshunt
      write(iout,1002) nb,qshunt,rsub,xsub
 1002 format(i7,5x,f10.2,2(f15.4))
      if(nb .le. 0 .or. nb .gt. mxext) then
        write(6,*) 'Illegal bus number - stop'
        write(2,*) 'Illegal bus number - stop'
        write(iout,*) 'Illegal bus number - stop'
        stop
      endif
      nbus = nbus + 1
      if(nbus .gt. mxint) then
        write(6,*) 'Too many busses - stop'
```

```
        write(2,*) 'Too many busses - stop'
        write(iout,*) 'Too many busses - stop'
        stop
      endif
      zsub        = cmplx(rsub,xsub)
      if(cabs(zsub) .gt. eps6) then
        if(rsub .lt. 0.0 .or. xsub .lt. 0.0) then
          write(6,*) 'Illegal rsubtrans or xsubtrans - stop'
          write(2,*) 'Illegal rsubtrans or xsubtrans - stop'
          write(iout,*) 'Illegal rsubtrans or xsubtrans - stop'
        endif
      endif
      nbext(nbus) = nb
      nbint(nb)   = nbus
      ycap        = cmplx(0.0,-qshunt / 100.0)
      ybus(nbus,nbus) = ybus(nbus,nbus) + ycap
      if(cabs(zsub) .gt. eps6) then
      ybus(nbus,nbus) = ybus(nbus,nbus) + 1.0 / zsub
      endif
      go to 1
    5 close(unit=1,status='keep')
c
      open(unit=1,file='ldat.yzb')
      write(6,*) 'Read line/transformer data from PCFLO fixed column ',
     1 'format file LDAT.YZB'
      write(6,*) ' from     to           r           x',
     1 '      charge         tap  phase shift'
      write(2,*) 'Read line/transformer data from PCFLO fixed column ',
     1 'format file LDAT.YZB'
      write(2,*) ' from     to           r           x',
     1 '      charge         tap  phase shift'
      write(iout,*) 'Read line/transformer data from PCFLO fixed ',
      1 'column format file LDAT.YZB'
      write(iout,*) ' from     to           r           x',
     1 '      charge         tap  phase shift'
c
c  read the ldat file
c
   10 read(1,1003,end=15) nfrom,nto,r,x,charge,tap,phase
 1003 format(2i4,2x,3f12.0,t76,f7.0,f8.0)
      if(nfrom .eq. 0 .and. nto .eq. 0) go to 15
      write(6,1004) nfrom,nto,r,x,charge,tap,phase
      write(2,1004) nfrom,nto,r,x,charge,tap,phase
      write(iout,1004) nfrom,nto,r,x,charge,tap,phase
 1004 format(1x,i5,2x,i5,4(2x,f10.4),3x,f10.2)
      if(nfrom .lt. 0 .or. nfrom .gt. mxext .or. nto .lt. 0
     1 .or. nto .gt. mxext) then
        write(6,*) 'Illegal bus number - stop'
        write(2,*) 'Illegal bus number - stop'
        write(iout,*) 'Illegal bus number - stop'
        stop
      endif
      if(nfrom .eq. nto) then
        write(6,*) 'Same bus number given on both ends - stop'
        write(2,*) 'Same bus number given on both ends - stop'
        write(iout,*) 'Same bus number given on both ends - stop'
        stop
      endif
      if(r .lt. -eps6 .or. tap .lt. -eps6) then
        write(6,*) 'Illegal resistance or tap - stop'
```

```
        write(2,*) 'Illegal resistance or tap - stop'
        write(iout,*) 'Illegal resistance or tap - stop'
        stop
      endif
      if(charge .lt. -eps6) then
        write(6,*) 'Line charging should be positive'
        write(2,*) 'Line charging should be positive'
        write(iout,*) 'Line charging should be positive'
        stop
      endif
      if(nfrom .ne. 0) then
        nfrom = nbint(nfrom)
        if(nfrom .eq. 0) then
          write(6,*) 'Bus not included in file bdat - stop'
          write(2,*) 'Bus not included in file bdat - stop'
          write(iout,*) 'Bus not included in file bdat - stop'
          stop
        endif
      endif
      if(nto   .ne. 0) then
        nto   = nbint(nto)
        if(nto .eq. 0) then
          write(6,*) 'Bus not included in file bdat - stop'
          write(2,*) 'Bus not included in file bdat - stop'
          write(iout,*) 'Bus not included in file bdat - stop'
          stop
        endif
      endif
      phase = phase / dr
      if(abs(phase) .gt. eps6 .and. abs(tap) .lt. eps6) tap = 1.0
      if((nfrom .eq. 0 .or. nto .eq. 0) .and. tap .gt. eps6) then
        write(6,*) 'Shunt elements should not have taps - stop'
        write(2,*) 'Shunt elements should not have taps - stop'
        write(iout,*) 'Shunt elements should not have taps - stop'
        stop
      endif
      if(tap .gt. eps6 .and. charge .gt. eps6) then
        write(6,*) 'Transformers should not have line charging'
        write(2,*) 'Transformers should not have line charging'
        write(iout,*) 'Transformers should not have line charging'
        stop
      endif
      yline = cmplx(r,x)
      if(cabs(yline) .lt. eps6) go to 10
      yline = 100.0 / yline
      ycap  = cmplx(0.0,charge / 200.0)
c
c  the line charging terms
c
      if(nfrom .ne. 0) ybus(nfrom,nfrom) = ybus(nfrom,nfrom) + ycap
      if(nto   .ne. 0) ybus(nto  ,nto  ) = ybus(nto  ,nto  ) + ycap
c
c  shunt elements
c
      if(nfrom .ne. 0 .and. nto   .eq. 0) ybus(nfrom,nfrom) =
     1 ybus(nfrom,nfrom) + yline
      if(nfrom .eq. 0 .and. nto   .ne. 0) ybus(nto  ,nto  ) =
     1 ybus(nto  ,nto  ) + yline
c
c  transmission lines
```

```
c
      if(nfrom .ne. 0 .and. nto .ne. 0 .and. tap .lt. eps6) then
        ybus(nfrom,nto  ) = ybus(nfrom,nto  ) - yline
        ybus(nto  ,nfrom) = ybus(nto  ,nfrom) - yline
        ybus(nfrom,nfrom) = ybus(nfrom,nfrom) + yline
        ybus(nto  ,nto  ) = ybus(nto  ,nto  ) + yline
      endif
c
c  transformers
c
      if(nfrom .ne. 0 .and. nto .ne. 0 .and. tap .ge. eps6) then
        ctap = tap * cmplx(cos(phase),sin(phase))
        yii  = yline / conjg(ctap) * (1.0 / ctap - 1.0)
        ykk  = yline * (1.0 - 1.0 / ctap)
        yik  = yline / conjg(ctap)
        yki  = yline / ctap
        ybus(nfrom,nfrom) = ybus(nfrom,nfrom) + yii + yik
        ybus(nto  ,nto  ) = ybus(nto  ,nto  ) + ykk + yki
        ybus(nfrom,nto  ) = ybus(nfrom,nto  ) - yik
        ybus(nto  ,nfrom) = ybus(nto  ,nfrom) - yki
      endif
      go to 10
c
c  write the nonzero ybus elements to file ybus
c
   15 close(unit=1,status='keep')
      write(6,*) 'Nonzero elements of ybus (in rectangular form)'
      write(2,*) 'Nonzero elements of ybus (in rectangular form)'
      write(iout,*) 'Nonzero elements of ybus (in rectangular form)'
      write(6,*) '-internal-   -external-'
      write(2,*) '-internal-   -external-'
      write(iout,*) '-internal-   -external-'
      ny = 0
      do 20 j = 1,nbus
      do 20 k = 1,nbus
      ysave(j,k) = ybus(j,k)
      if(cabs(ybus(j,k)) .lt. eps9) go to 20
      ny = ny + 1
      write(6,1005) j,k,nbext(j),nbext(k),ybus(j,k)
      write(2,1005) j,k,nbext(j),nbext(k),ybus(j,k)
      write(iout,1005) j,k,nbext(j),nbext(k),ybus(j,k)
 1005 format(2i5,3x,2i5,2e20.8)
   20 continue
      close(unit=2,status='keep')
      fill = ny / float(nbus * nbus)
      write(iout,525) ny,fill
      write(6,525) ny,fill
  525 format(/1x,'Number of nonzero elements in YBUS.YZB = ',i5/1x,
     1'percent fill = ',2pf8.2/)
c
c  bifactorization - replace original ybus with lu
c
      nm1 = nbus - 1
      do 30 ipiv = 1,nm1
      write(iout,530) ipiv
      write(6,530) ipiv
  530 format(1x,'LU.YZB pivot element = ',i5)
      ipiv1 = ipiv + 1
      alpha = 1.0 / ybus(ipiv,ipiv)
      do 32 k = ipiv1,nbus
```

```
      ybus(ipiv,k) = alpha * ybus(ipiv,k)
   32 continue
      do 34 j = ipiv1,nbus
      alpha   = ybus(j,ipiv)
      do 34 k = ipiv1,nbus
      ybus(j,k) = ybus(j,k) - alpha * ybus(ipiv,k)
   34 continue
   30 continue
      write(iout,530) nbus
      write(iout,532)
      write(6,530) nbus
      write(6,532)
  532 format(/1x,'Nonzero LU.YZB follows'/)
      open(unit=4,file='lu.yzb')
      nlu = 0
      do 35 j = 1,nbus
      do 35 k = 1,nbus
      ymag = cabs(ybus(j,k))
      if(ymag .le. eps9) go to 35
      nlu = nlu + 1
      write(iout,1005) j,k,nbext(j),nbext(k),ybus(j,k)
      write(6,1005) j,k,nbext(j),nbext(k),ybus(j,k)
      write(4,1005) j,k,nbext(j),nbext(k),ybus(j,k)
   35 continue
      fill = nlu / float(nbus * nbus)
      write(iout,535) nlu,fill
      write(6,535) nlu,fill
  535 format(/1x,'Number of nonzero elements in LU.YZB = ',i5/1x,
     1'percent fill = ',2pf8.2/)
      close(unit=4,status='keep')
c
c  check l times u
c
      write(iout,560)
      write(6,560)
  560 format(1x,'LU.YZB .ne. YBUS.YZB follows'/)
      do 36 j = 1,nbus
      do 37 k = 1,nbus
      lmat(j,k) = czero
      umat(j,k) = czero
      if(j .ge. k) lmat(j,k) = ybus(j,k)
      if(j .lt. k) umat(j,k) = ybus(j,k)
   37 continue
      umat(j,j) = 1.0
   36 continue
      do 38 j = 1,nbus
      do 38 k = 1,nbus
      ytest   = czero
      do 39 l = 1,k
      ytest = ytest + lmat(j,l) * umat(l,k)
   39 continue
      diff = ysave(j,k) - ytest
      if(cabs(diff) .gt. eps4) then
        write(iout,1005) j,k,nbext(j),nbext(k)
     1 ,diff
        write(6,1005) j,k,nbext(j),nbext(k)
     1 ,diff
      endif
   38 continue
c
```

```
c  form uz (urt + diag)
c
      write(iout,536) nbus
      write(6,536) nbus
  536 format(1x,'UZ.YZB pivot = ',i5)
      uz(nbus,nbus) = 1.0 / ybus(nbus,nbus)
      nm1 = nbus - 1
      do 40 kdum = 1,nm1
      k   = nbus - kdum
      write(iout,536) k
      write(6,536) k
      uz(k,k) = 1.0 / ybus(k,k)
      kp1     = k + 1
      do 41 j = kp1,nbus
      uzelem  = czero
      jm1     = j - 1
      alpha   = 1.0 / ybus(j,j)
      do 42 l = k,jm1
      uzelem  = uzelem - ybus(j,l) * uz(l,k)
   42 continue
      uz(j,k) = uzelem * alpha
   41 continue
   40 continue
      write(iout,537)
      write(6,537)
  537 format(/1x,'Nonzero UZ.YZB follows'/)
      open(unit=8,file='uz.yzb')
      nlu     = 0
      do 45 j = 1,nbus
      do 45 k = 1,nbus
      ymag    = cabs(uz(j,k))
      if(ymag .le. eps9) go to 45
      nlu     = nlu + 1
      write(iout,1005) j,k,nbext(j),nbext(k),uz(j,k)
      write(6,1005) j,k,nbext(j),nbext(k),uz(j,k)
      write(8,1005) j,k,nbext(j),nbext(k),uz(j,k)
   45 continue
      fill = nlu / float(nbus * nbus)
      write(iout,545) nlu,fill
      write(6,545) nlu,fill
  545 format(/1x,'Number of nonzero elements in UZ.YZB = ',i5/1x,
     1'percent fill = ',2pf8.2/)
      close(unit=8,status='keep')
c
c  form z
c
      open(unit=10,file='zbuslu.yzb')
      do 50 kdum = 1,nbus
      k   = nbus - kdum + 1
      write(iout,550) k
      write(6,550) k
  550 format(1x,'ZUBSLU.YZB column = ',i5)
      do 51 jdum = 1,nbus
      j   = nbus - jdum + 1
      zbus(j,k) = czero
      if(j .ge. k) zbus(j,k) = uz(j,k)
      if(j .eq. nbus) go to 51
      jp1 = j + 1
      do 52 l    = jp1,nbus
      zbus(j,k) = zbus(j,k) - ybus(j,l) * zbus(l,k)
```

```
   52 continue
   51 continue
   50 continue
      write(iout,565)
      write(6,565)
  565 format(/1x,'Writing ZBUSLU.YZB to disk')
      do 80 j = 1,nbus
      do 80 k = 1,nbus
      write(10,1005) j,k,nbext(j),nbext(k),zbus(j,k)
   80 continue
      close(unit=10,status='keep')
c
c  check ybus * zbus
c
      write(iout,5010)
      write(6,5010)
 5010 format(/1x,'Nonzero YBUS.YZB * ZBUSLU.YZB follows')
      do 60 j = 1,nbus
      do 60 k = 1,nbus
      elem    = czero
      do 61 l = 1,nbus
      elem    = elem + ysave(j,l) * zbus(l,k)
   61 continue
      ymag    = cabs(elem)
      if(ymag .gt. eps4) then
        write(iout,1005) j,k,nbext(j),nbext(k),elem
        write(6,1005) j,k,nbext(j),nbext(k),elem
      endif
   60 continue
c
c  copy ysave back into ybus, and zero zbus
c
      do 70 j = 1,nbus
      unity(j,j) = 1.0
      do 70 k = 1,nbus
      ybus(j,k) = ysave(j,k)
      zbus(j,k) = czero
   70 continue
c
c  gaussian eliminate ybus, while performing the same operations
c  on the unity matrix
c
      nm1 = nbus - 1
      do 71 ipiv = 1,nm1
      write(iout,561) ipiv
      write(6,561) ipiv
  561 format(1x,'Gaussian elimination YBUS.YZB pivot = ',i5)
      alpha      = 1.0 / ybus(ipiv,ipiv)
      ipp1 = ipiv + 1
c
c  pivot row operations for ybus and unity
c
      do 72 k = 1,nbus
      if(k .gt. ipiv) then
        ybus(ipiv,k)  = ybus(ipiv,k)  * alpha
        else
        unity(ipiv,k) = unity(ipiv,k) * alpha
      endif
   72 continue
c
```

```
c  pivot element
c
      ybus(ipiv,ipiv) = 1.0
c
c  kron reduction of ybus and unity below the pivot row and to
c  the right of the pivot column
c
      do 74 j = ipp1,nbus
      alpha   = ybus(j,ipiv)
      do 74 k = 1,nbus
      if(k .gt. ipiv)
     1 ybus(j,k)  = ybus(j,k)  - alpha * ybus(ipiv,k)
      if(k .lt. ipiv)
     1 unity(j,k) = unity(j,k) - alpha * unity(ipiv,k)
   74 continue
c
c  elements directly below the pivot element
c
      do 76 j = ipp1,nbus
      alpha   = ybus(j,ipiv)
      ybus(j,ipiv)  = ybus(j,ipiv)  - alpha * ybus(ipiv,ipiv)
      unity(j,ipiv) = unity(j,ipiv) - alpha * unity(ipiv,ipiv)
   76 continue
   71 continue
c
c  last row
c
      write(iout,561) nbus
      write(6,561) nbus
      alpha   = 1.0 / ybus(nbus,nbus)
      do 77 k = 1,nbus
      unity(nbus,k) = unity(nbus,k) * alpha
   77 continue
      ybus(nbus,nbus) = 1.0
      ngy = 0
      ngu = 0
      write(iout,562)
      write(6,562)
  562 format(/1x,'Writing gaussian eliminated YBUS.YZB and ',
     1 'UNITYMAT.YZB to disk')
      open(unit=12,file='ybusred.yzb')
      open(unit=13,file='unitymat.yzb')
      do 78 j = 1,nbus
      do 78 k = 1,nbus
      ymag = cabs(ybus(j,k))
      if(ymag .ge. eps9) then
        write(12,1005) j,k,nbext(j),nbext(k),ybus(j,k)
        ngy   = ngy + 1
      endif
      umag = cabs(unity(j,k))
      if(umag .ge. eps9) then
        write(13,1005) j,k,nbext(j),nbext(k),unity(j,k)
        ngu   = ngu + 1
      endif
   78 continue
      close(unit=12,status='keep')
      close(unit=13,status='keep')
      fill = ngy / float(nbus * nbus)
      write(iout,555) ngy,fill
      write(6,555) ngy,fill
```

```
  555 format(/1x,'Number of nonzero elements in gaussian eliminated',
     1' YBUZ.YZB = ',i5/1x,
     1'percent fill = ',2pf8.2/)
      fill = ngu / float(nbus * nbus)
      write(iout,556) ngu,fill
      write(6,556) ngu,fill
  556 format(/1x,'Number of nonzero elements in gaussian eliminated',
     1' UNITYMAT.YZB = ',i5/1x,
     1'percent fill = ',2pf8.2/)
c
c  back substitute to find z
c
      do 81 k = 1,nbus
      zbus(nbus,k) = unity(nbus,k)
   81 continue
      do 82 jdum = 2,nbus
      j = nbus - jdum + 1
      write(iout,550) j
      write(6,550) j
      do 82 kdum = 1,nbus
      k = nbus - kdum + 1
      zbus(j,k) = unity(j,k)
      jp1 = j + 1
      do 83 l = jp1,nbus
      zbus(j,k) = zbus(j,k) - ybus(j,l) * zbus(l,k)
   83 continue
   82 continue
      write(iout,565)
      write(6,565)
      open(unit=11,file='zbusgau.yzb')
      do 84 j = 1,nbus
      do 84 k = 1,nbus
      write(11,1005) j,k,nbext(j),nbext(k),zbus(j,k)
   84 continue
      close(unit=11,status='keep')
c
c  check ybus * zbus
c
      write(iout,5030)
      write(6,5030)
 5030 format(/1x,'Nonzero YBUS.YZB * ZBUSGAU.YZB follows')
      do 85 j = 1,nbus
      do 85 k = 1,nbus
      elem    = czero
      do 86 l = 1,nbus
      elem    = elem + ysave(j,l) * zbus(l,k)
   86 continue
      ymag    = cabs(elem)
      if(ymag .gt. eps4) then
          write(iout,1005) j,k,nbext(j),nbext(k),elem
          write(6,1005) j,k,nbext(j),nbext(k),elem
        endif
   85 continue
      stop
      end
```