4

Sparse Matrix Solution Techniques

A sparse matrix is one that has "very few" nonzero elements. A sparse system is one in which its mathematical description gives rise to sparse matrices. Any large system that can be described by coupled nodes may give rise to a sparse matrix if the majority of nodes in the system have very few connections. Many systems in engineering and science result in sparse matrix descriptions. Large systems in which each node is connected to only a handful of other nodes include the mesh points in a finite-element analysis, nodes in an electronic circuit, and the busbars in an electric power network. For example, power networks may contain thousands of nodes (busbars), but the average connectivity of electric power network nodes is three; each node is connected, on average, to three other nodes. This means that, in a system composed of a thousand nodes, the nonzero percentage of the descriptive system matrix is

$$\frac{\frac{4 \text{ nonzero elements}}{\text{row}} \times 1000 \text{ rows}}{1000 \times 1000 \text{ elements}} \times 100\% = 0.4\% \text{ nonzero elements}$$

Thus, if only the nonzero elements were stored in memory, they would require only 0.4% of the memory requirements of the full 1000×1000 matrix. Full storage of an $n \times n$ system matrix grows as n^2, whereas the sparse storage of the same system matrix increases only linearly as $\sim n$. Thus significant storage and computational savings can be realized by exploiting sparse storage and solution techniques. Another motivating factor in exploiting sparse matrix solution techniques is the computational effort involved in solving matrices with large percentages of zero elements. Consider the solution of the linear problem

$$Ax = b$$

where A is sparse. The factorization of L and U from A requires a significant number of multiplications where one or both of the factors may be zero. If it is known ahead of time where the zero elements reside in the matrix, these multiplications can be avoided (since their product will be zero) and significant computational effort can be saved. The salient point here is that these computations are skipped altogether. A person performing an LU factorization by hand can note which values are zero and skip those particular multiplications. A computer, however, does not have the ability to "see" the zero elements. Therefore, the sparse solution technique must be formulated in such a way as to avoid zero computations altogether and operate only upon nonzero elements.

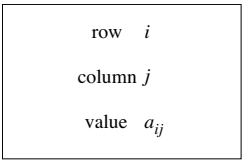


FIGURE 4.1
Basic storage element for a_{ij}

In this chapter, both the storage and computational aspects of sparse matrix solutions will be explored. Several storage techniques will be discussed and ordering techniques to minimize computational effort will be developed.

4.1 Storage Methods

In sparse storage methods, only the nonzero elements of the $n \times n$ matrix A are stored in memory, along with the indexing information needed to traverse the matrix from element to element. Thus each element must be stored with its real value (a_{ij}) and its position indices (row and column) in the matrix. The basic storage unit may be visualized as the object shown in Figure 4.1.

In addition to the basic information, indexing information must also be included in the object, such as a link to the next element in the row, or the next element in the column, as shown in Figure 4.2.

The only additional information necessary to fully traverse the matrix either by row or column is an indication of the first element in each row or column. This is a stand-alone set of links that point to the first element in each row or column.

Example 4.1
Determine a linked list representation for the sparse matrix

$$A = \begin{bmatrix} -1 & 0 & -2 & 0 & 0 \\ 2 & 8 & 0 & 1 & 0 \\ 0 & 0 & 3 & 0 & -2 \\ 0 & -3 & 2 & 0 & 0 \\ 1 & 2 & 0 & 0 & -4 \end{bmatrix}$$

Solution 4.1 A linked list representation of this matrix is shown in Figure 4.3. The last element of each column and row are linked to a null point. Note that each object is linked to its adjacent neighbors in the matrix, both

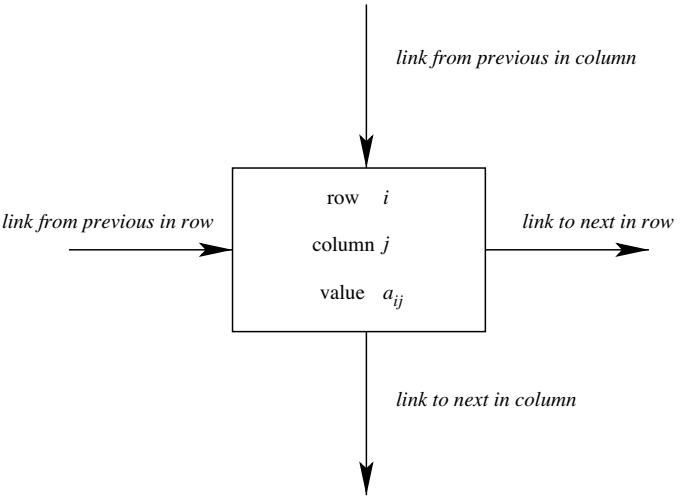


FIGURE 4.2
Storage element for element a_{ij} with links

by column and by row. In this way, the entire matrix can be traversed in any direction by starting at the first element and following the links until the desired element is reached. ■

If a command requires a particular matrix element, then by choosing either the column or row, the element can be located by progressing along the links. If, during the search, the null point is reached, then the desired element does not exist and a value of zero is returned. Additionally, if the matrix elements are linked by increasing index and an element is reached that has a greater index than the desired element, then the progression terminates and a value of zero is returned.

A linked list representation for a sparse matrix is not unique and the elements do not necessarily have to be linked in increasing order by index. However, ordering the elements by increasing index leads to a simplified search since the progression can be terminated before reaching the null point if the index of the linked object exceeds the desired element. If the elements are not linked in order, the entire row or column must be traversed to determine whether or not the desired element is nonzero. The drawback to an ordered list is that the addition of new nonzero elements to the matrix requires the update of both row and column links.

Example 4.2

Insert the matrix element $A(4, 5) = 10$ to the linked list of Example 4.1.

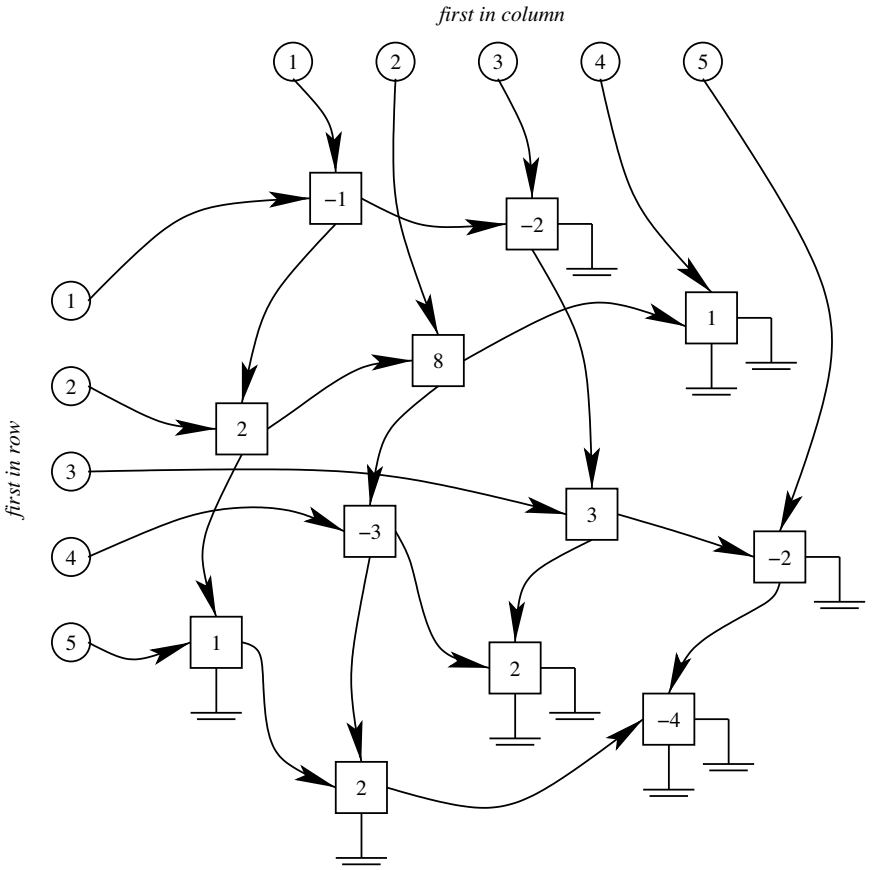


FIGURE 4.3
Linked list for Example 4.1

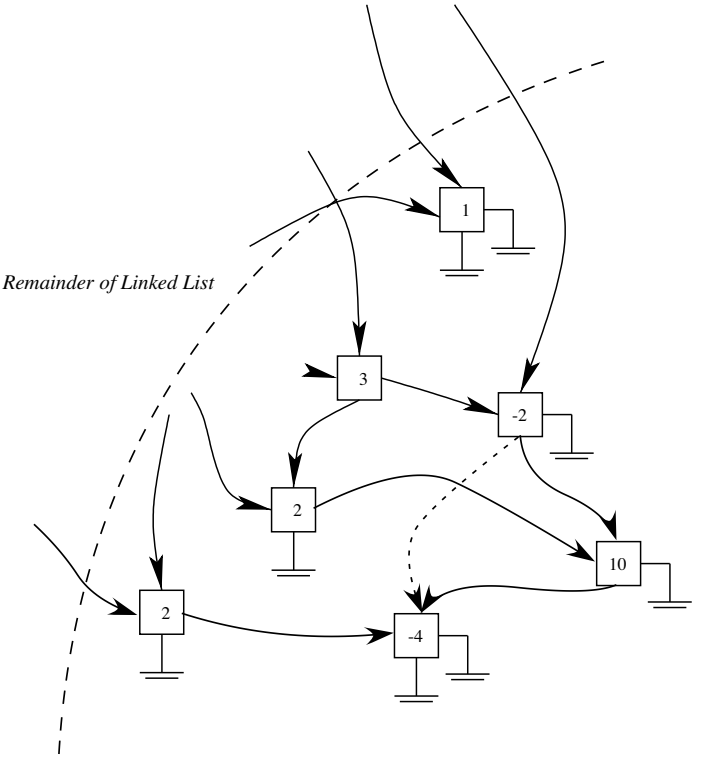


FIGURE 4.4
Insertion of matrix element $A(4,5) = 10$

Solution 4.2 The insertion of the new element is shown in Figure 4.4. The addition of this element requires two traversals of the matrix to insert the new element and to update the links: one traversal by row and one by column. Starting at the *first in row* link for row 4 (value=−3), the elements are progressed by link. Following the links and monitoring the column indices, it is discovered that the element with column index 3 (value=2) is the last element in the row since it points to null. Since the new element has column index 5, it is inserted between the (value=2) element and the null point in the row linked list. Similarly, starting at the *first in column* link for column 5 (value=−2), the column is traversed and inserted between the elements with row indices 3 (value=−2) and 5 (value=−4). The column links are updated to reflect the insertion. ■

If the linked lists of the matrix are not ordered by index, then new elements can be added without transversing the rows or columns. A new element can be inserted in each row or column by inserting them before the first element and updating the *first in row* and *first in column* pointers.

Many software languages, however, do not support the use of objects, pointers, and linked lists. In this case it is necessary to develop a procedure to mimic a linked list format by the use of vectors. Three vectors are required to represent each nonzero element object: one vector containing the row number (NROW), one vector containing the column number (NCOL), and one vector containing the value of the element (VALUE). These vectors are of length nnz where nnz is the number of nonzero elements. Two vectors, also of length nnz, are required to represent the next-in-row links (NIR) and the next-in-column (NIC) links. If an element is the last in the row or column, then the NIR or NIC value for that element is 0. Finally, two vectors of length n contain the *first in row* (FIR) and *first in column* (FIC) links.

The elements of the matrix are assigned a (possibly arbitrary) numbering scheme that corresponds to their order in the NROW, NCOL, VALUE, NIR, and NIC vectors. This order is the same for each of these five vectors. The FIR and FIC vectors will also refer to this number scheme.

Example 4.3

Find the vectors NROW, NCOL, VALUE, NIR, NIC, FIR, and FIC for the sparse matrix of Example 4.1.

Solution 4.3 The matrix of Example 4.1 is reproduced below with the numbering scheme given in parentheses to the left of each nonzero element. The numbering scheme is sequential by row and goes from 1 to $nnz = 12$.

$$A = \begin{bmatrix} (1)\ -1 & 0 & (2)\ -2 & 0 & 0 \\ (3)\ 2 & (4)\ 8 & 0 & (5)\ 1 & 0 \\ 0 & 0 & (6)\ 3 & 0 & (7)\ -2 \\ 0 & (8)\ -3 & (9)\ 2 & 0 & 0 \\ (10)\ 1 & (11)\ 2 & 0 & 0 & (12)\ -4 \end{bmatrix}$$

The ordering scheme indicated yields the following nnz vectors:

k	VALUE	NROW	NCOL	NIR	NIC
1	–1	1	1	2	3
2	–2	1	3	0	6
3	2	2	1	4	10
4	8	2	2	5	8
5	1	2	4	0	0
6	3	3	3	7	9
7	–2	3	5	0	12
8	–3	4	2	9	11
9	2	4	3	0	0
10	1	5	1	11	0
11	2	5	2	12	0
12	–4	5	5	0	0

and the following n vectors:

	FIR	FIC
1	1	1
2	3	4
3	6	2
4	8	5
5	10	7

Consider the matrix element $A(2,2) = 8$. It is the fourth element in the numbering scheme, so its information is stored in the fourth place in vectors VALUE, NROW, NCOL, NIR, and NIC. Thus VALUE(4)=8, NROW(4)=2, and NCOL(4)=2. The next element in row 2 is $A(2,4) = 1$ and it is element 5 in the numbering scheme. Therefore NROW(4)=5, signifying that element 5 follows element 4 in its row (note, however, that it does not indicate which row they are in). Similarly, the next element in column 2 is $A(4,2) = -3$ and it is element 8 in the numbering scheme. Therefore NCOL(4)=8. ■

Example 4.4

Find the sparse storage vectors for the matrix data given below.

i	j	$A(i,j)$
8	8	–28
7	2	5
10	5	7
5	10	7
5	7	3
6	6	–33
6	5	10
1	8	8
5	5	–44
1	4	19
4	3	6
8	3	1
3	4	6
10	3	9
2	1	2
9	7	13
10	2	10
3	8	1
3	10	9
4	1	19
7	7	–68
8	1	8
2	10	10

i	j	$A(i,j)$
3	3	–40
6	7	19
7	8	15
4	4	–38
5	6	10
3	5	11
7	5	3
5	4	9
1	2	2
3	7	9
2	7	5
7	3	9
2	2	–21
1	1	–33
7	9	13
4	5	9
5	3	11
10	10	–30
7	6	19
9	9	–17
8	7	15

Solution 4.4 Since data is read in from a file, there is no guarantee that the data will be given in any particular order. The elements are numbered in the order in which they are read, and not in the order of the matrix A. The full matrix (including the zero elements) should never be explicitly created.

i	NROW	NCOL	NIR	NIC	Value
1	8	8	0	0	–28
2	7	2	35	17	5
3	10	5	41	0	7
4	5	10	0	41	7
5	5	7	4	25	3
6	6	6	25	42	–33
7	6	5	6	30	10
8	1	8	0	18	8
9	5	5	28	7	–44
10	1	4	8	13	19
11	4	3	27	40	6
12	8	3	44	14	1
13	3	4	29	27	6
14	10	3	3	0	9
15	2	1	36	20	2
16	9	7	43	0	13
17	10	2	14	0	10
18	3	8	19	26	1

i	NROW	NCOL	NIR	NIC	Value
19	3	10	0	4	9
20	4	1	11	22	19
21	7	7	26	44	–68
22	8	1	12	0	8
23	2	10	0	19	10
24	3	3	13	11	–40
25	6	7	0	21	19
26	7	8	38	1	15
27	4	4	39	31	–38
28	5	6	5	6	10
29	3	5	33	39	11
30	7	5	42	3	3
31	5	4	9	0	9
32	1	2	10	36	2
33	3	7	18	5	9
34	2	7	23	33	5
35	7	3	30	12	9
36	2	2	34	2	–21
37	1	1	32	15	–33
38	7	9	0	43	13
39	4	5	0	9	9
40	5	3	31	35	11
41	10	10	0	0	–30
42	7	6	21	0	19
43	9	9	0	0	–17
44	8	7	1	16	15

and

i	FIR	FIC
1	37	37
2	15	32
3	24	24
4	20	10
5	40	29
6	7	28
7	2	34
8	22	8
9	16	38
10	17	23

Figure 4.5 shows a visualization of the linked list produced by the sparse vectors. The first-in-row and first-in-column pointers point to the first element in each row/column. Note that not all of the first elements are shown. The next-in-row and next-in-column links are shown. Each row or column can be traversed by starting at the first element and subsequently moving through the links. ■

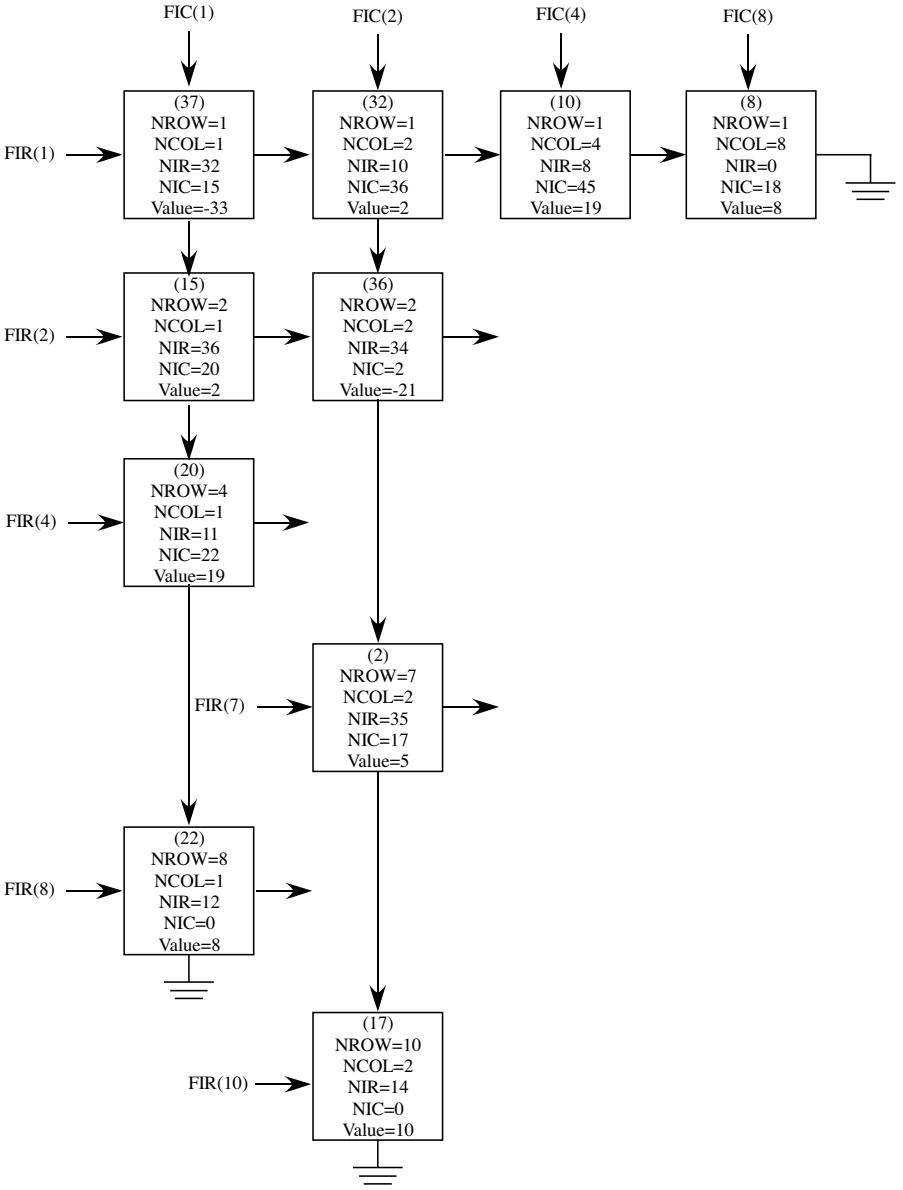


FIGURE 4.5
Elements in columns 1 and 2 of A in Example 4.4

4.2 Sparse Matrix Representation

Sparse matrices arise as the result of the mathematical modeling of a sparse system. In many cases, the system has a naturally occurring physical network representation or lends itself to a physically intuitive representation. In these cases, it is often informative to visualize the connectivity of the system by graphical means. In the graphical representation, each node of the graph corresponds to a node in the system. Each edge of the graph corresponds to a branch of the network. As with a network, the graph, consisting of vertices and edges, is often represented by a set of points in the plane joined by a line representing each edge. Matrices that arise from the mathematical model of a graphically represented network are structurally symmetric. In other words, if the matrix element a_{ij} is nonzero, then the matrix element a_{ji} is also nonzero. This implies that, if node i is connected to node j, then node j is also connected to node i. Matrices that are not structurally symmetric can be made symmetric by adding an element of value zero in the appropriate position within the matrix.

In addition to a graphical representation, it is also common to visualize sparse matrices by a matrix that is clear except for an identifying symbol (such as a $\times, \bullet, *$, or other mark) to represent the position of the nonzero elements in the matrix. The finite element grid of the trapezoid shown in Figure 4.6(a) gives rise to the sparse matrix structure shown in Figure 4.6(b). Note that the ordering of the matrix is not unique; another numbering scheme for the nodes will result in an alternate matrix structure.

Example 4.5
Find the sparse storage matrix representation for the data given in Example 4.4.

Solution 4.5 Figure 4.7 shows the sparse matrix visual representation of the matrix data given in Example 4.5. While it is not necessary to visualize the matrix explicitly to perform an LU factorization, it is often informative. ■

4.3 Ordering Schemes

Node ordering schemes are important in minimizing the number of multiplications and divisions required for both L and U triangularization and forward/backward substitution. A good ordering will result in the addition of few *fills* to the triangular factors during the LU factorization process. A *fill*

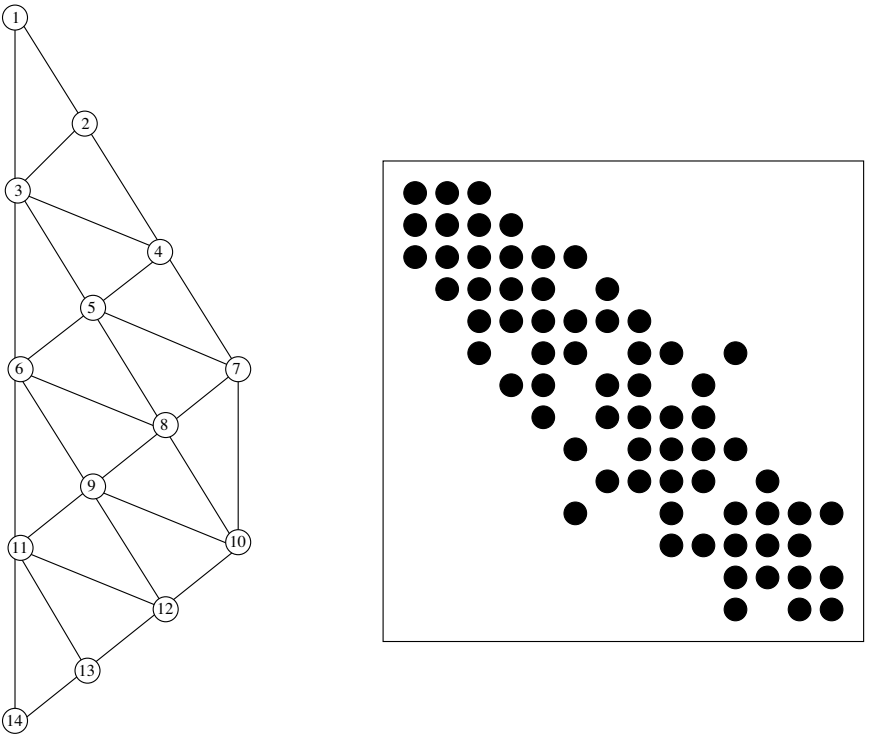


FIGURE 4.6
(a) A finite element grid model (b) The corresponding matrix

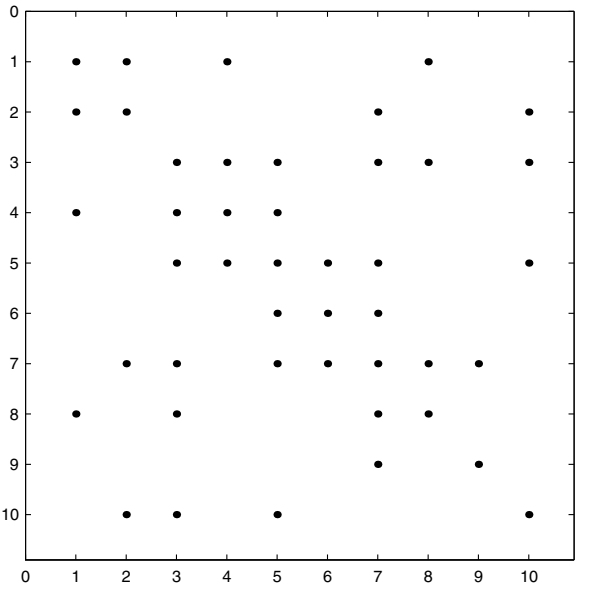


FIGURE 4.7
Matrix visualization for Example 4.5

is a nonzero element in the L or U matrix that was zero in the original A matrix. If A is a full matrix, $\alpha = \frac{n^3-n}{3}$ multiplications and divisions are required for the LU factorization process and $\beta = n^2$ multiplications and divisions are required for the forward/backward substitution process. The number of multiplications and divisions required can be substantially reduced in sparse matrix solutions if a proper node ordering is used.

Example 4.6
Determine the number of multiplications, divisions, and fills required for the solution of the system shown in Figure 4.8.

Solution 4.6 The LU factorization steps yield

$$q_{11} = a_{11}$$
$$q_{21} = a_{21}$$
$$q_{31} = a_{31}$$
$$q_{41} = a_{41}$$
$$q_{51} = a_{51}$$

$$q_{12} = a_{12}/q_{11}$$

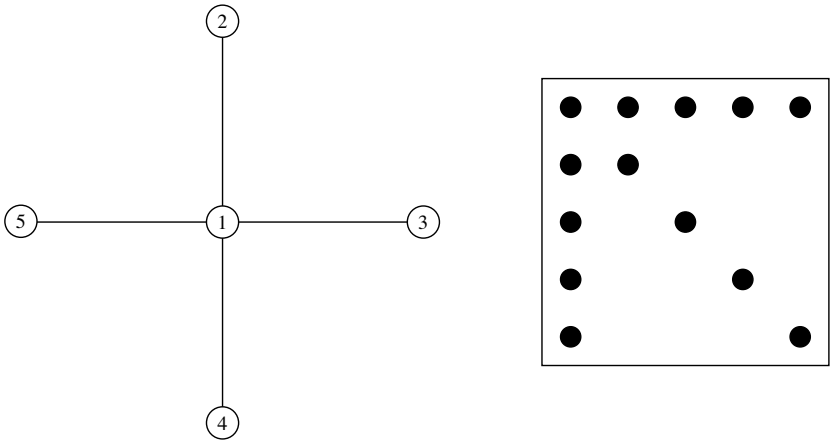


FIGURE 4.8
Graph and matrix for Example 4.6

$$q_{13} = a_{13}/q_{11}$$
$$q_{14} = a_{14}/q_{11}$$
$$q_{15} = a_{15}/q_{11}$$

$$q_{22} = a_{22} - q_{21}q_{12}$$
$$q_{32} = a_{32} - q_{31}q_{12}$$
$$q_{42} = a_{42} - q_{41}q_{12}$$
$$q_{52} = a_{52} - q_{51}q_{12}$$

$$q_{23} = (a_{23} - q_{21}q_{13})/q_{22}$$
$$q_{24} = (a_{24} - q_{21}q_{14})/q_{22}$$
$$q_{25} = (a_{25} - q_{21}q_{15})/q_{22}$$

$$q_{33} = a_{33} - q_{31}q_{13} - q_{32}q_{23}$$
$$q_{43} = a_{43} - q_{41}q_{13} - q_{42}q_{23}$$
$$q_{53} = a_{53} - q_{51}q_{13} - q_{52}q_{23}$$

$$q_{34} = (a_{34} - q_{31}q_{14} - q_{32}q_{24})/q_{33}$$
$$q_{35} = (a_{35} - q_{31}q_{15} - q_{32}q_{25})/q_{33}$$

$$q_{44} = a_{44} - q_{41}q_{14} - q_{42}q_{24} - q_{43}q_{34}$$

$$q_{54} = a_{54} - q_{51}q_{14} - q_{52}q_{24} - q_{53}q_{34}$$

$$q_{45} = \left(a_{45} - q_{41}q_{15} - q_{22}q_{25} - q_{43}q_{35}\right)/q_{44}$$

$$q_{55} = a_{55} - q_{51}q_{15} - q_{52}q_{25} - q_{53}q_{35} - q_{54}q_{45}$$

The multiplications and divisions required for the LU factorization are summarized by row and column.

row	column	multiplications	divisions	fills
1		0	0	
	1	0	4	
2		4	0	$a_{32},\ a_{42},\ a_{52}$
	2	3	3	$a_{23},\ a_{24},\ a_{25}$
3		6	0	$a_{43},\ a_{53}$
	3	4	2	$a_{34},\ a_{35}$
4		6	0	a_{54}
	4	3	1	a_{45}
5		4	0	

Therefore $\alpha = 40$ is the total number of multiplications and divisions in the LU factorization. The forward ($Ly = b$) and backward ($Ux = y$) substitution steps yield

$$\begin{aligned}
y_1 &= b_1/q_{11}\\
y_2 &= \left(b_2 - q_{21}y_1\right)/q_{22}\\
y_3 &= \left(b_3 - q_{31}y_1 - q_{32}y_2\right)/q_{33}\\
y_4 &= \left(b_4 - q_{41}y_1 - q_{42}y_2 - q_{43}y_3\right)/q_{44}\\
y_5 &= \left(b_5 - q_{51}y_1 - q_{52}y_2 - q_{53}y_3 - q_{54}y_4\right)/q_{55}\\
x_5 &= y_5\\
x_4 &= y_4 - q_{45}x_5\\
x_3 &= y_3 - q_{35}x_5 - q_{34}x_4\\
x_2 &= y_2 - q_{25}x_5 - q_{24}x_4 - q_{23}x_3\\
x_1 &= y_1 - q_{15}x_5 - q_{14}x_4 - q_{13}x_3 - q_{12}x_2
\end{aligned}$$

	forward		backward	
row	multiplications	divisions	multiplications	divisions
1	0	1	4	0
2	1	1	3	0
3	2	1	2	0
4	3	1	1	0
5	4	1	0	0

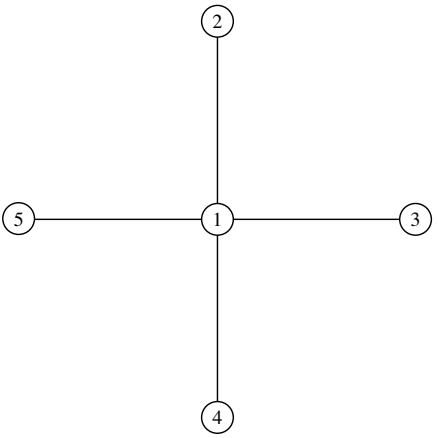


FIGURE 4.9
Graph for Example 4.6

Thus $\beta = 25$ is the total number of multiplications and divisions in the forward and backward substitution steps. The total number of multiplications and divisions for the solution of $Ax = b$ is $\alpha + \beta = 65$. ■

A fill occurs when a matrix element that was originally zero becomes nonzero during the factorization process. This can be visually simulated using a graphical approach. Consider the graph of Example 4.6 shown again in Figure 4.9.

In this numbering scheme, the row and column corresponding to node 1 is factorized first. This corresponds to the removal of node 1 from the graph. When node 1 is removed, all of the vertices to which it was connected must then be joined. Each edge added represents two fills in the Q matrix (q_{ij} and q_{ji}) since Q is symmetric. The graph after the removal of node 1 is shown in Figure 4.10. The dashed lines indicate that six fills will occur as a result: q_{23}, q_{24}, q_{25}, q_{34}, q_{35}, and q_{45}. These are the six fills that are also listed in the solution of the example.

Example 4.7
Determine the number of multiplications, divisions, and fills required for the solution of the system shown in Figure 4.11.

Solution 4.6 The LU factorization steps yield

$$q_{11} = a_{11}$$
$$q_{51} = a_{51}$$
$$q_{15} = a_{15}/q_{11}$$
$$q_{22} = a_{22}$$

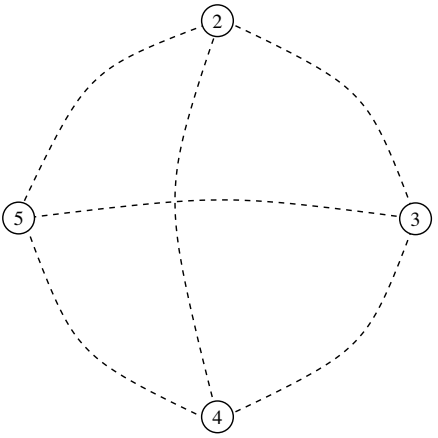


FIGURE 4.10
Resulting fills after removing node 1

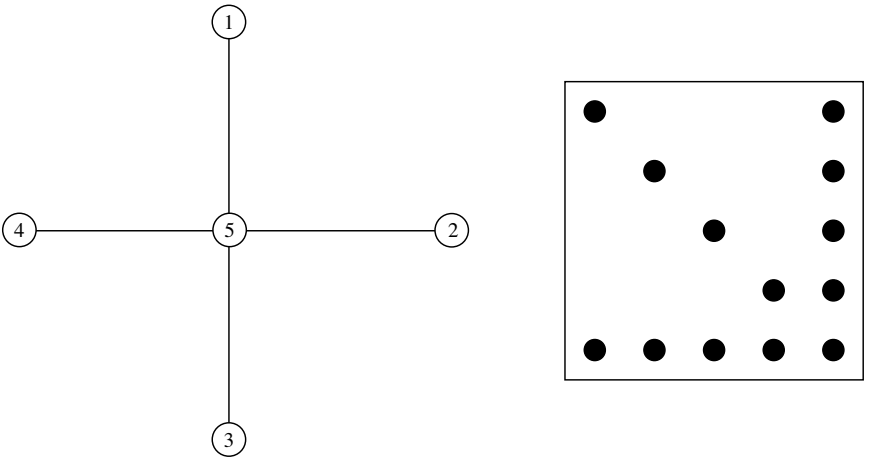


FIGURE 4.11
Graph and matrix for Example 4.7

$$
\begin{aligned}
q_{25} &= a_{25}/q_{22} \\
q_{52} &= a_{52} \\
q_{33} &= a_{33} \\
q_{53} &= a_{53} \\
q_{35} &= a_{35}/q_{33}
\end{aligned}
$$

$$
\begin{aligned}
q_{44} &= a_{44} \\
q_{54} &= a_{54} \\
q_{45} &= a_{45}/q_{44} \\
q_{55} &= a_{55} - q_{51}q_{15} - q_{52}q_{25} - q_{53}q_{35} - q_{54}q_{45}
\end{aligned}
$$

The multiplications and divisions required for the LU factorization are summarized by row and column.

row	column	multiplications	divisions	fills
1		0	0	
	1	0	1	
2		0	0	
	2	0	1	
3		0	0	
	3	0	1	
4		0	0	
	4	0	1	
5		4	0	

Therefore $\alpha = 8$ is the total number of multiplications and divisions in the LU factorization. The forward ($Ly = b$) and backward ($Ux = y$) substitution steps yield

$$
\begin{aligned}
y_1 &= b_1/q_{11} \\
y_2 &= b_2/q_{22} \\
y_3 &= b_3/q_{33} \\
y_4 &= b_4/q_{44} \\
y_5 &= \left(b_5 - q_{51}y_1 - q_{52}y_2 - q_{53}y_3 - q_{54}y_4\right)/q_{55} \\
x_5 &= y_5 \\
x_4 &= y_4 - q_{45}x_5 \\
x_3 &= y_3 - q_{35}x_5 \\
x_2 &= y_2 - q_{25}x_5 \\
x_1 &= y_1 - q_{15}x_5
\end{aligned}
$$

	forward		backward	
row	multiplications	divisions	multiplications	divisions
1	0	1	1	0
2	0	1	1	0
3	0	1	1	0
4	0	1	1	0
5	4	1	0	0

Thus $\beta = 13$ is the total number of multiplications and divisions in the forward and backward substitution steps. The total number of multiplications and divisions for the solution of $Ax = b$ is $\alpha + \beta = 21$. ■

Even though both original matrices had the same number of nonzero elements, there is a significant reduction in the number of multiplications and divisions by simply renumbering the vertices of the matrix graph. This is due, in part, to the number of fills that occurred during the LU factorization of the matrix. The Q matrix of Example 4.6 became full, whereas the Q matrix of Example 4.7 retained the same sparse structure as the original A matrix. From these two examples, it can be concluded that, although various node orders do not affect the accuracy of the linear solution, the ordering scheme greatly affects the time in which the solution is achieved. A good ordering scheme is one in which the resulting Q matrix has a structure similar to the original A matrix. This means that the number of fills is minimized. This objective forms the basis for a variety of ordering schemes. The problem of optimal ordering is an NP-complete problem [59], but several schemes have been developed that provide near-optimal results.

Example 4.8

Determine number of fills, α, and β for the matrix shown in Figure 4.12 as currently ordered. This is the same matrix as in Example 4.5.

Solution 4.8 The first step is to determine where the fills from LU factorization will occur. By observation, the fills will occur in the places designated by the $\triangle$ in the matrix shown in Figure 4.13. From the figure, the number of fills is 24.

Rather than calculating the number of multiplications and divisions required for LU factorization and forward/backward substitution, there is a handy way of calculating α and β directly from the filled matrix.

$$\alpha = \sum_{i=1}^{n} (nnz \text{ in column } i \text{ below } q_{ii} + 1) \times (nnz \text{ in row } i \text{ to right of } q_{ii}) \quad (4.1)$$

$$\beta = nnz \text{ of matrix } Q \quad (4.2)$$

Using Equations (4.1) and (4.2),

$$\alpha = (3 \times 4) + (4 \times 5) + (5 \times 6) + (4 \times 5) + (4 \times 5) + (3 \times 4) + (3 \times 4) + (2 \times 3) + (1 \times 2) + (0 \times 1) = 134$$

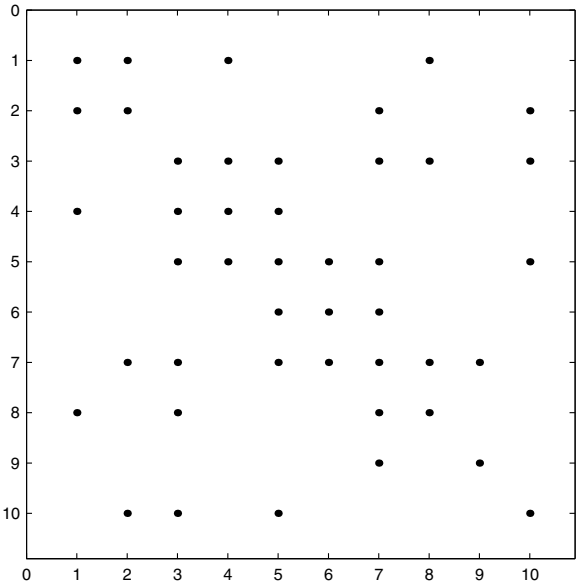


FIGURE 4.12
Matrix for Example 4.8

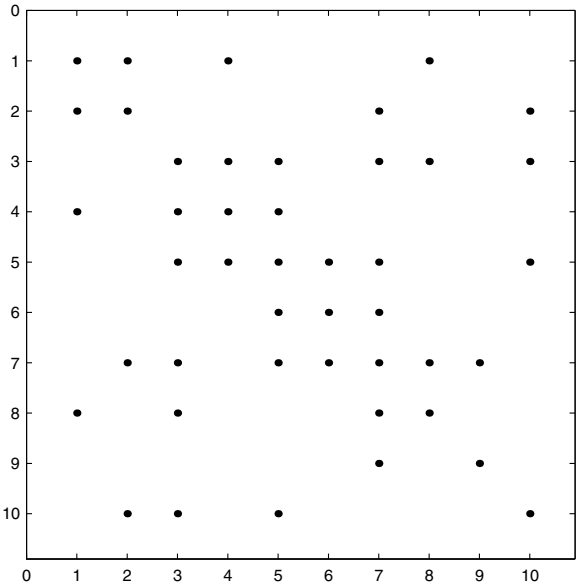


FIGURE 4.13
Matrix with fills for Example 4.8

and $\beta = nnz = 68$ for the Q matrix shown in Figure 4.13; thus $\alpha + \beta = 202$. ■

Even without an ordering scheme, the sparse matrix solution process yields over a 50% reduction in computation. One goal of an ordering scheme is to introduce the least number of fills in the factored matrix Q to minimize the number of multiplications and divisions α. A second goal is also to minimize β, which is the number of multiplications and divisions in the forward/backward substitution step. These dual objectives lead to several approaches to ordering.

Example 4.9
Update the sparse vectors of Example 4.4 to include the new fills.

Solution 4.9 The locations of the fills were calculated in Example 4.8. These fills are added as zero elements in the sparse storage vectors.

Recall that Example 4.8 indicated that β would be 68. Note that elements 45 to 68 correspond to the new fills, but since the LU factorization has not yet been conducted, the Value element for each of the fills is still zero. Note that the NIR and NIC columns have been updated throughout the vectors to accommodate the insertion of the new elements.

The FIR and FIC vectors have not changed. ■

i	NROW	NCOL	NIR	NIC	Value	i	NROW	NCOL	NIR	NIC	Value
1	8	8	65	66	–28	51	4	7	49	5	0
2	7	2	35	48	5	52	7	4	30	50	0
3	10	5	64	0	7	53	4	10	0	4	0
4	5	10	0	63	7	54	10	4	3	0	0
5	5	7	59	25	3	55	7	10	0	57	0
6	6	6	25	42	–33	56	10	7	58	0	0
7	6	5	6	30	10	57	8	10	0	67	0
8	1	8	0	47	8	58	10	8	68	0	0
9	5	5	28	7	–44	59	5	8	4	61	0
10	1	4	8	45	19	60	8	5	62	3	0
11	4	3	27	40	6	61	6	8	63	26	0
12	8	3	50	14	1	62	8	6	44	64	0
13	3	4	29	27	6	63	6	10	0	55	0
14	10	3	54	0	9	64	10	6	56	0	0
15	2	1	36	20	2	65	8	9	57	43	0
16	9	7	66	56	13	66	9	8	43	58	0
17	10	2	14	0	10	67	9	10	0	41	0
18	3	8	19	49	1	68	10	9	41	0	0
19	3	10	0	53	9						
20	4	1	46	22	19						
21	7	7	26	44	–68						
22	8	1	48	0	8						
23	2	10	0	19	10						
24	3	3	13	11	–40						
25	6	7	61	21	19						
26	7	8	38	1	15						
27	4	4	39	31	–38						
28	5	6	5	6	10						
29	3	5	33	39	11						
30	7	5	42	60	3						
31	5	4	9	52	9						
32	1	2	10	36	2						
33	3	7	18	51	9						
34	2	7	47	33	5						
35	7	3	52	12	9						
36	2	2	45	46	–21						
37	1	1	32	15	–33						
38	7	9	55	65	13						
39	4	5	51	9	9						
40	5	3	31	35	11						
41	10	10	0	0	–30						
42	7	6	21	62	19						
43	9	9	67	68	–17						
44	8	7	1	16	15						
45	2	4	34	13	0						
46	4	2	11	2	0						
47	2	8	23	18	0						
48	8	2	12	17	0						
49	4	8	53	59	0						
50	8	4	60	54	0						

Example 4.10

In the following LU factors, find the missing element (26) (which corresponds to $q(7,8)$).

i	NROW	NCOL	NIR	NIC	Value	i	NROW	NCOL	NIR	NIC	Value
1	8	8	65	66	–19.5512	35	7	3	52	12	9.0000
2	7	2	35	48	5.0000	36	2	2	45	46	–20.8788
3	10	5	64	0	10.2510	37	1	1	32	15	–33.0000
4	5	10	0	63	–0.2799	38	7	9	55	65	–0.2625
5	5	7	59	25	–0.1676	39	4	5	51	9	–0.4081
6	6	6	25	42	–30.2699	40	5	3	31	35	11.0000
7	6	5	6	30	10.0000	41	10	10	0	0	–16.9270
8	1	8	0	47	–0.2424	42	7	6	21	62	20.6759
9	5	5	28	7	–36.6288	43	9	9	67	68	–12.6365
10	1	4	8	45	–0.5758	44	8	7	1	16	16.4275
11	4	3	27	40	6.0000	45	2	4	34	13	–0.0552
12	8	3	50	14	1.0000	46	4	2	11	2	1.1515
13	3	4	29	27	–0.1500	47	2	8	23	18	–0.0232
14	10	3	54	0	9.0000	48	8	2	12	17	0.4848
15	2	1	36	20	2.0000	49	4	8	53	59	–0.1833
16	9	7	66	56	13.0000	50	8	4	60	54	4.7828
17	10	2	14	0	10.0000	51	4	7	49	5	–0.0623
18	3	8	19	49	–0.0250	52	7	4	30	50	1.6258
19	3	10	0	53	–0.2250	53	4	10	0	4	–0.0729
20	4	1	46	22	19.0000	54	10	4	3	0	1.9015
21	7	7	26	44	–49.5250	55	7	10	0	57	–0.1649
22	8	1	48	0	8.0000	56	10	7	58	0	8.1677
23	2	10	0	19	–0.4790	57	8	10	0	67	–0.2145
24	3	3	13	11	–40.0000	58	10	8	68	0	4.1944
25	6	7	61	21	–0.6830	59	5	8	4	61	–0.0608
26	7	8	38	1	******	60	8	5	62	3	2.2268
27	4	4	39	31	–26.0971	61	6	8	63	26	–0.0201
28	5	6	5	6	–0.2730	62	8	6	44	64	0.6079
29	3	5	33	39	–0.2750	63	6	10	0	55	–0.0925
30	7	5	42	60	6.1385	64	10	6	56	0	2.7986
31	5	4	9	52	10.6500	65	8	9	57	43	–0.2206
32	1	2	10	36	–0.0606	66	9	8	43	58	4.3121
33	3	7	18	51	–0.2250	67	9	10	0	41	–0.2429
34	2	7	47	33	–0.2395	68	10	9	41	0	3.0691

Solution 4.10 The locations of the fills were calculated in Examples 4.8 and 4.9.

To begin the calculation of element (26), recall that row calculations are given by

$$q_{jk} = \frac{1}{q_{jj}} \left(a_{jk} - \sum_{i=1}^{j-1} q_{ji} q_{ik} \right) \quad \text{for } k = j+1, \ldots, n \tag{4.3}$$

where in this case $j = 7$ and $k = 8$. Note that the product $q_{ji}q_{ik}$ will only be nonzero if both q_{ji} and q_{ik} are nonzero.

To determine which q_{ji}s are nonzero, the first-in-row pointer will begin at FIR(7)=2. Traversing the row, the nonzero elements in the row to the left of the diagonal are

$$\begin{array}{ccccccccc} (2) & \rightarrow & (35) & \rightarrow & (52) & \rightarrow & (30) & \rightarrow & (42) \\ [2] & & [3] & & [4] & & [5] & & [6] \end{array}$$

The corresponding NCOL values are shown in the square brackets below each element. The diagonal element q_{77} (NROW=7, NCOL=7) is the next-in-row NIR(42)=21.

Similarly, to determine which q_{ik}s are nonzero, the first-in-column pointer will begin at FIC(8)=8. Traversing the column, the nonzero elements in the column above the diagonal are

$$\begin{array}{ll} (8) & [1] \\ \downarrow & \\ (47) & [2] \\ \downarrow & \\ (18) & [3] \\ \downarrow & \\ (49) & [4] \\ \downarrow & \\ (59) & [5] \\ \downarrow & \\ (61) & [6] \end{array}$$

The corresponding NROW values are shown in the square brackets to the right of each element.

Matching up the corresponding elements:

$$\begin{aligned} q_{78} &= q_{78} - \left(q_{72}q_{28} + q_{73}q_{38} + q_{74}q_{48} + q_{75}q_{58} + q_{76}q_{68}\right)/q_{77} \\ &\quad \text{(elements)} \\ (26) &= \left((26) - \left((2)(47) + (35)(18) + (52)(49) + (30)(59) + (42)(61)\right)\right)/(21) \end{aligned}$$

$$
\begin{aligned}
&\text{(Values (elements))}\\
&= (15 - ((5.0000)(-0.0232) + (9.0000)(-0.0250) + (1.6258)(-0.1833) +\\
&\quad (6.1385)(-0.0608) + (20.6759)(-0.0201))) / (-49.5250)\\
&= -0.3317
\end{aligned}
$$

Note that the initial value of element (26) is a_{78}. ■

4.3.1 Scheme 0

From Examples 4.6 and 4.7, it can be generalized that a better ordering is achieved if the nodes are ordered into a lower-right pointing "arrow" matrix. One rapid method of achieving this effect is to number the nodes according to their degree, where the degree of a node is defined as the number of edges connected to it. In this scheme, the nodes are ordered from lowest degree to highest degree.

Scheme 0

1. Calculate the degree of all vertices.
2. Choose the node with the lowest degree. Place in the ordering scheme.
3. In case of a tie, choose the node with the lowest natural ordering.
4. Return to step 2.

Example 4.11
Using Scheme 0, reorder the matrix of Example 4.8. Calculate α, β, and the number of fills for this ordering.

Solution 4.11 The degrees of each of the nodes are given below:

node	degree
1	3
2	3
3	5
4	3
5	5
6	2
7	6
8	3
9	1
10	3

Applying Scheme 0, the new ordering is

$$\text{Ordering 0} = [9\ \ 6\ \ 1\ \ 2\ \ 4\ \ 8\ \ 10\ \ 3\ \ 5\ \ 7]$$

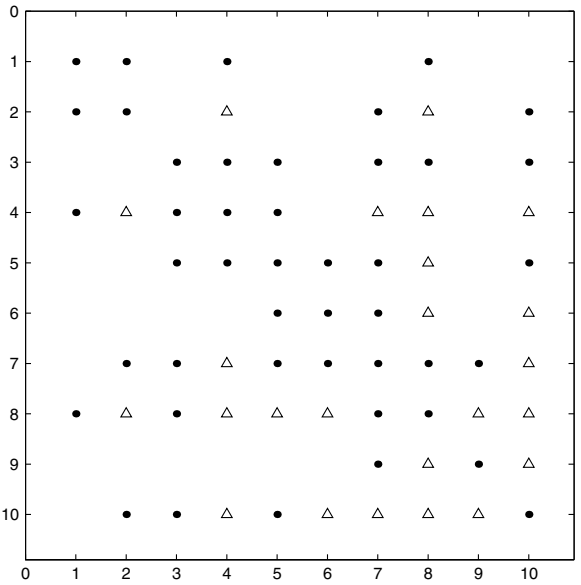


FIGURE 4.14
Matrix with fills for Example 4.11

Reordering the matrix of Example 4.8 to reflect this ordering yields the matrix (with fills) shown in Figure 4.14. Note how the nonzero elements begin to resemble the desired lower-right pointing arrow. The Scheme 0 ordering results in 16 fills as compared to 24 with the original ordering. From the matrix and Equations (4.1) and (4.2), $\alpha = 110$ and $\beta = 60$; thus $\alpha + \beta = 170$, which is a considerable reduction over the original $\alpha + \beta = 202$. ■

4.3.2 Scheme I

Scheme 0 offers simplicity and speed of generation, but does not directly take into account the effect of fills on the ordering procedure. To do this, the effect of eliminating the nodes as they are ordered must be taken into account. This modification is given in Scheme I.

Scheme I

1. Calculate the degree of all vertices.
2. Choose the node with the lowest degree. Place in the ordering scheme. Eliminate it and update degrees accordingly.
3. In case of a tie, choose the node with the lowest natural ordering.
4. Return to step 1.

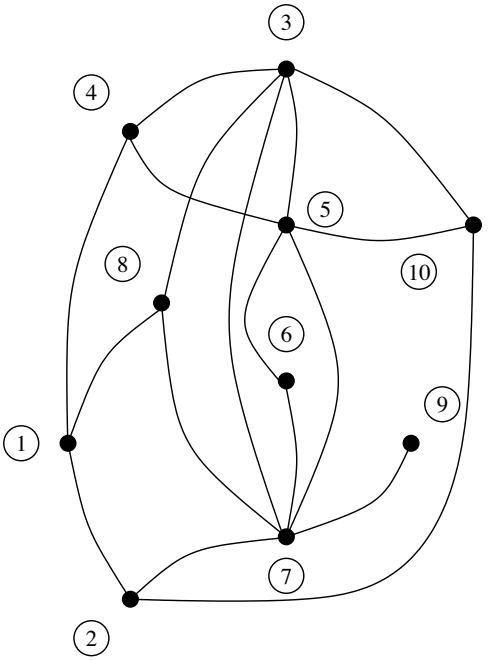


FIGURE 4.15
Graph of the matrix in Figure 4.12

Scheme I is also known by many names, including the Markowitz algorithm [35], the Tinney I algorithm [55], or most generally as the minimum degree algorithm.

Example 4.12

Using Scheme I, reorder the matrix of Example 4.8. Calculate α, β, and the number of fills for this ordering.

Solution 4.12 The ordering for Scheme I takes into account the effect of fills on the ordering as nodes are placed in the ordering scheme and eliminated. This algorithm is best visualized using the graphical representation of the matrix. The graph of the original unordered matrix of Figure 4.12 is shown in Figure 4.15.

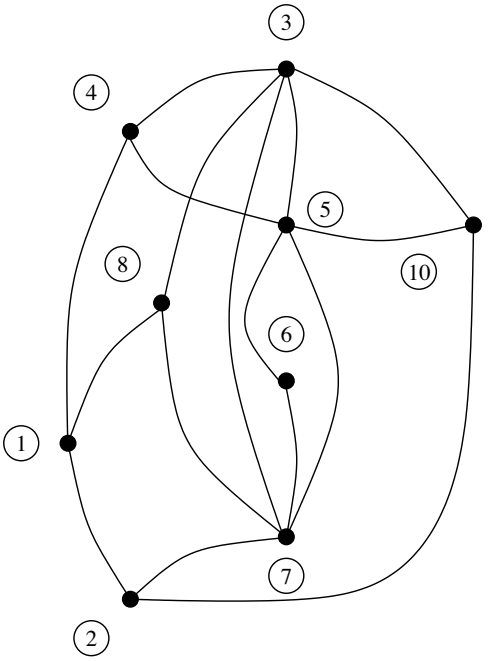


FIGURE 4.16
Updated graph with the removal of node 9

The degrees of each of the nodes are given below:

node	degree
1	3
2	3
3	5
4	3
5	5
6	2
7	6
8	3
9	1
10	3

From the degrees, the node with the lowest degree is ordered first. Node 9 has the lowest degree, with only one connection. Its elimination does not cause any fills. The updated graph is shown in Figure 4.16.

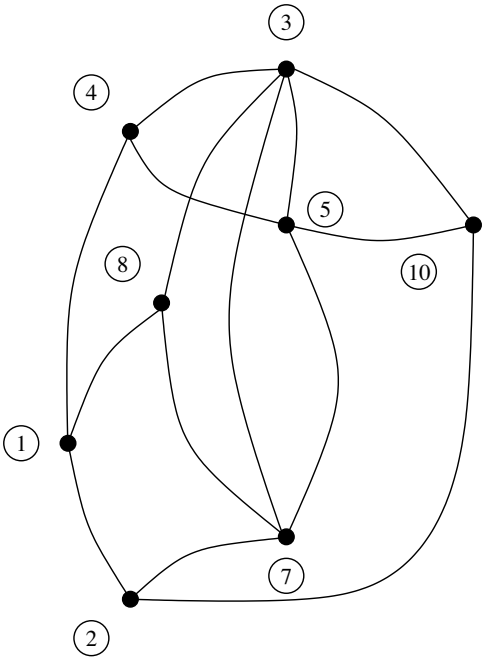


FIGURE 4.17
Updated graph with the removal of node 6

The updated degree of each of the nodes is given below:

node	degree
1	3
2	3
3	5
4	3
5	5
6	2
7	5
8	3
10	3

Node 7 now has one less degree. Applying the Scheme I algorithm again indicates that the next node to be chosen is node 6, with a degree of 2. Node 6 is connected to both node 5 and node 7. Since there is a preexisting connection between these nodes, the elimination of node 6 does not create a fill between nodes 5 and 6. The elimination of node 6 is shown in Figure 4.17.

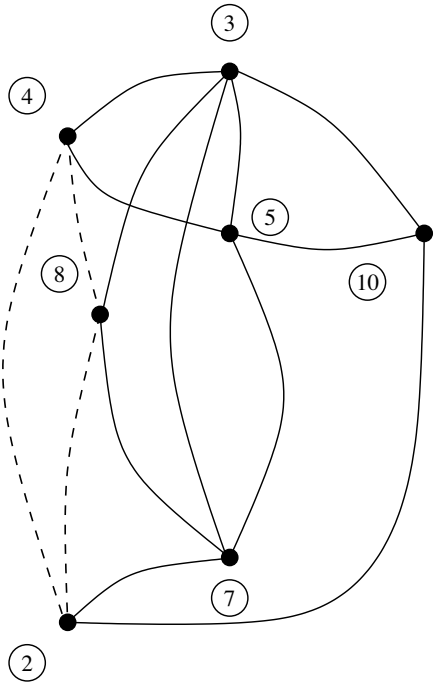


FIGURE 4.18
Updated graph with the removal of node 1

The new node degrees are

node	degree
1	3
2	3
3	5
4	3
5	4
7	4
8	3
10	3

As a result of the elimination of node 6, the degrees of nodes 5 and 7 decrease by one. Applying the Scheme I algorithm again indicates that the nodes with the lowest degrees are nodes [1 2 4 8 10]. Since there is a tie between these nodes, the node with the lowest natural ordering, node 1, is chosen and eliminated. Node 1 is connected to nodes 2, 4, and 8. None of these nodes is connected; therefore, the elimination of node 1 creates three fills: 4–8, 4–2, and 2–8. These fills are shown with the dashed edges in Figure 4.18.

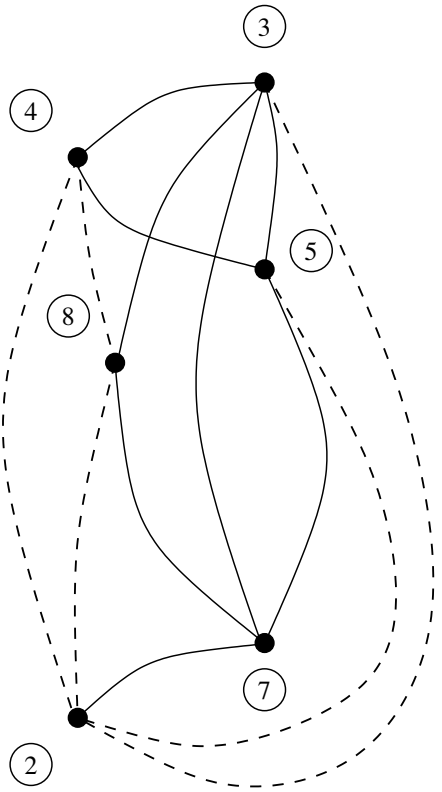


FIGURE 4.19
Updated graph with the removal of node 10

The new node degrees after the removal of node 1 are

node	degree
2	4
3	5
4	4
5	5
7	5
8	4
10	3

The addition of the three fills increased the degrees of nodes 2, 4, and 8. Applying the Scheme I algorithm again indicates that the node with the lowest degree is node 10. There is no tie in degree this time. Node 10 is chosen and eliminated. The elimination of node 10 creates two fills between nodes 2–5 and 2–3. These fills are shown with the dashed edges in Figure 4.19.

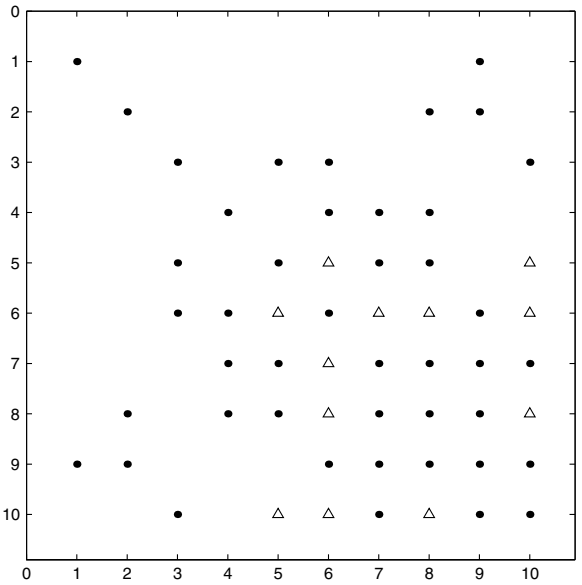


FIGURE 4.20
Matrix with fills for Example 4.12

Continuing to apply the Scheme I algorithm successively until all nodes have been chosen and eliminated yields the following final ordering:

$$\text{Ordering I} = [9 \;\; 6 \;\; 1 \;\; 10 \;\; 4 \;\; 2 \;\; 3 \;\; 5 \;\; 7 \;\; 8]$$

Reordering the matrix of Example 4.12 to reflect this ordering yields the matrix (with fills) shown in Figure 4.20. Note how the nonzero elements continue to resemble the desired lower-right pointing arrow. The Scheme I ordering results in 12 fills as compared to 24 with the original ordering, and 16 with Scheme 0. From the matrix and Equations (4.1) and (4.2), $\alpha = 92$ and $\beta = 56$; thus $\alpha + \beta = 148$, which is a considerable reduction over the original $\alpha + \beta = 202$ and the $\alpha + \beta = 170$ of Scheme 0. ■

4.3.3 Scheme II

Scheme 0 offers a rapid way to order the nodes to give a quick "once-over" and obtain a reasonable ordering. It requires little computation beyond calculating the degrees of each node of the matrix. Scheme I takes this approach one step further. It still relies on the minimum-degree approach, but it includes a simulation of the LU factorization process to update the node degrees at each step of the factorization. One further improvement to this approach is

to develop a scheme that endeavors to minimize the number of fills at each step of the factorization. Thus, at each step, each elimination alternative is considered and the number of resulting fills is calculated. This scheme is also known as the Berry algorithm and the Tinney II algorithm and is summarized below:

Scheme II

1. For each node, calculate the number of fills that would result from its elimination.

2. Choose the node with the lowest number of fills.

3. In case of a tie, choose the node with the lowest degree.

4. In case of a tie, choose the node with the lowest natural ordering.

5. Place the node in the ordering scheme. Eliminate it and update fills and degrees accordingly.

6. Return to step 1.

Example 4.13
Using Scheme II, reorder the matrix of Example 4.8. Calculate α, β, and the number of fills for this ordering.

Solution 4.13 The ordering for Scheme II takes into account the effect of fills on the ordering as nodes are placed in the ordering scheme and eliminated. The degrees and resulting fills are given below:

node	degree	fills if eliminated	edges created
1	3	3	2–4, 2–8, 4–8
2	3	3	1–7, 1–10, 7–10
3	5	6	4–7, 4–8, 4–10, 5–8, 7–10, 8–10
4	3	2	1–3, 1–5
5	5	6	3–6, 4–6, 4–7, 4–10, 6–10, 7–10
6	2	0	none
7	6	12	2–3, 2–5, 2–6, 2–8, 2–9, 3–6, 3–9, 5–8, 5–9, 6–8, 6–9, 8–9
8	3	2	1–3, 1–7
9	1	0	none
10	3	2	2–3, 2–5

From this list, the elimination of nodes 6 or 9 will not result in any additional edges or fills. Since there is a tie, the node with the lowest degree is chosen.

Thus node 9 is chosen and eliminated. The number of fills and degrees is updated to apply the Scheme II algorithm again.

node	degree	fills if eliminated	edges created
1	3	3	2–4, 2–8, 4–8
2	3	3	1–7, 1–10, 7–10
3	5	6	4–7, 4–8, 4–10, 5–8, 7–10, 8–10
4	3	2	1–3, 1–5
5	5	6	3–6, 4–6, 4–7, 4–10, 6–10, 7–10
6	2	0	none
7	5	7	2–3, 2–5, 2–6, 2–8, 3–6, 5–8, 6–8
8	3	2	1–3, 1–7
10	3	2	2–3, 2–5

The next node to be eliminated is node 6 because it creates the fewest fills if eliminated. This node is therefore chosen and eliminated. The number of fills and degrees is again updated.

node	degree	fills if eliminated	edges created
1	3	3	2–4, 2–8, 4–8
2	3	3	1–7, 1–10, 7–10
3	5	6	4–7, 4–8, 4–10, 5–8, 7–10, 8–10
4	3	2	1–3, 1–5
5	5	6	3–6, 4–6, 4–7, 4–10, 6–10, 7–10
7	5	7	2–3, 2–5, 2–6, 2–8, 3–6, 5–8, 6–8
8	3	2	1–3, 1–7
10	3	2	2–3, 2–5

The two nodes that create the fewest fills are nodes 4 and 8. Both nodes have the same number of degrees; therefore, the node with the lowest natural ordering, node 4, is chosen and eliminated.

The Scheme II algorithm continues until all nodes have been added to the ordering scheme and subsequently eliminated. Scheme II results in the following ordering:

$$\text{Ordering II } = [9\ \ 6\ \ 4\ \ 8\ \ 2\ \ 1\ \ 3\ \ 5\ \ 7\ \ 10]$$

Reordering the matrix of Example 4.8 to reflect the ordering of Scheme II yields the ordering with fills shown in Figure 4.21. This ordering yields only 10 fills, leading to $\alpha = 84, \beta = 54$, and $\alpha + \beta = 138$. This represents a computational effort of only 68% of the original unordered system. ■

Scheme I endeavors to reduce the number of multiplications and divisions in the LU factorization process, whereas Scheme II focuses on reducing the multiplications and divisions in the forward/backward substitution process. Scheme 0 offers simplicity and speed of generation, but the performance improvement of Scheme I offsets the additional algorithm complexity [55].

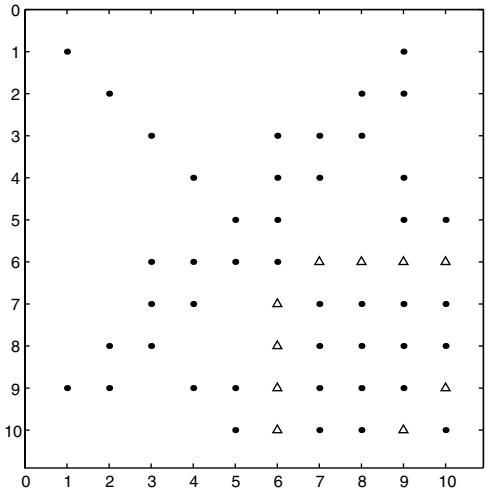


FIGURE 4.21
Matrix with fills for Example 4.13

Scheme II, however, frequently does not offer enough of an improvement to merit implementation. The decision of which scheme to implement is problem dependent and is best left up to the user.

4.3.4 Other Schemes

Modifications to these algorithms have been introduced to reduce computational requirements. These modifications are summarized below [19]. The first modification to the minimum-degree algorithm is the use of mass elimination, inspired by the concept of indistinguishable nodes [18]. This modification allows a subset of nodes to be eliminated at one time. If nodes x and y satisfy

$$Adj(y) \cup \{y\} = Adj(x) \cup \{x\} \tag{4.4}$$

where *Adj(y)* indicates the set of nodes adjacent to y, then nodes x and y are said to be indistinguishable and can be numbered consecutively in the ordering. This also reduces the number of nodes to be considered in an ordering, since only a representative node from each set of indistinguishable nodes needs to be considered. This accelerates the degree update step of the minimum-degree algorithm, which is typically the most computationally intensive step. Using mass elimination, the degree update is required only for the representative nodes.

The idea of incomplete degree update allows avoiding degree update for nodes that are known not to be minimum degree. Between two nodes u and

v, node v is said to be outmatched by u if [12]

$$Adj(u) \cup \{u\} \subseteq Adj(v) \cup \{v\} \tag{4.5}$$

Thus, if a node v becomes outmatched by u in the elimination process, the node u can be eliminated before v in the minimum-degree ordering algorithm. From this, it follows that it is not necessary to update the degree of v until node u has been eliminated. This further reduces the time-consuming degree update steps.

Another modification to the minimum-degree algorithm is one in which all possible nodes of minimum degree are eliminated before the degree update step. At a given step in the elimination process, the elimination of node y does not change the structure of the remaining nodes not in *Adj(y)*. The multiple-minimum-degree (MMD) algorithm delays degree update of the nodes in *Adj(y)* and chooses another node with the same degree as y to eliminate. This process continues until there are no more nodes left with the same degree as y. This algorithm was found to perform as well as the minimum-degree algorithm regarding the number of fills introduced [33]. In addition, it was found that the MMD algorithm performed faster. This was attributed to the identification of indistinguishable and outmatched nodes earlier in the algorithm, as well as the reduced number of degree updates.

Ties often occur for a given criteria (degrees or fills) in an ordering algorithm. The tie breaker often falls back on the natural ordering of the original matrix. It has been recognized that the natural ordering greatly affects the factorization in terms of number of fills and computation time. Thus it is often preferable to use a rapid "preconditioning" ordering before applying the ordering algorithm. Scheme 0 offers one such preordering, but to date no consistent optimum method for preordering has been developed that works well for all types of problems.

4.4 Power System Applications

Large sparse matrices occur frequently in power system applications, including state estimation, power flow analysis, and transient and dynamic stability simulations. Computational efficiency of these applications depends heavily on their formulation and the use of sparse matrix techniques. To better understand the impact of sparsity on power system problems, consider the power flow Jacobian of the IEEE 118 bus system shown in Figure 4.22.

The Jacobian of this system has 1051 nonzero elements and has the structure shown in Figure 4.23(a). Note the dominance of the main diagonal and then the two subdiagonals which result from the $\frac{\partial \Delta Q}{\partial \delta}$ and $\frac{\partial \Delta P}{\partial V}$ sub-Jacobians. The LU factorization of this Jacobian yields the structure shown in Figure

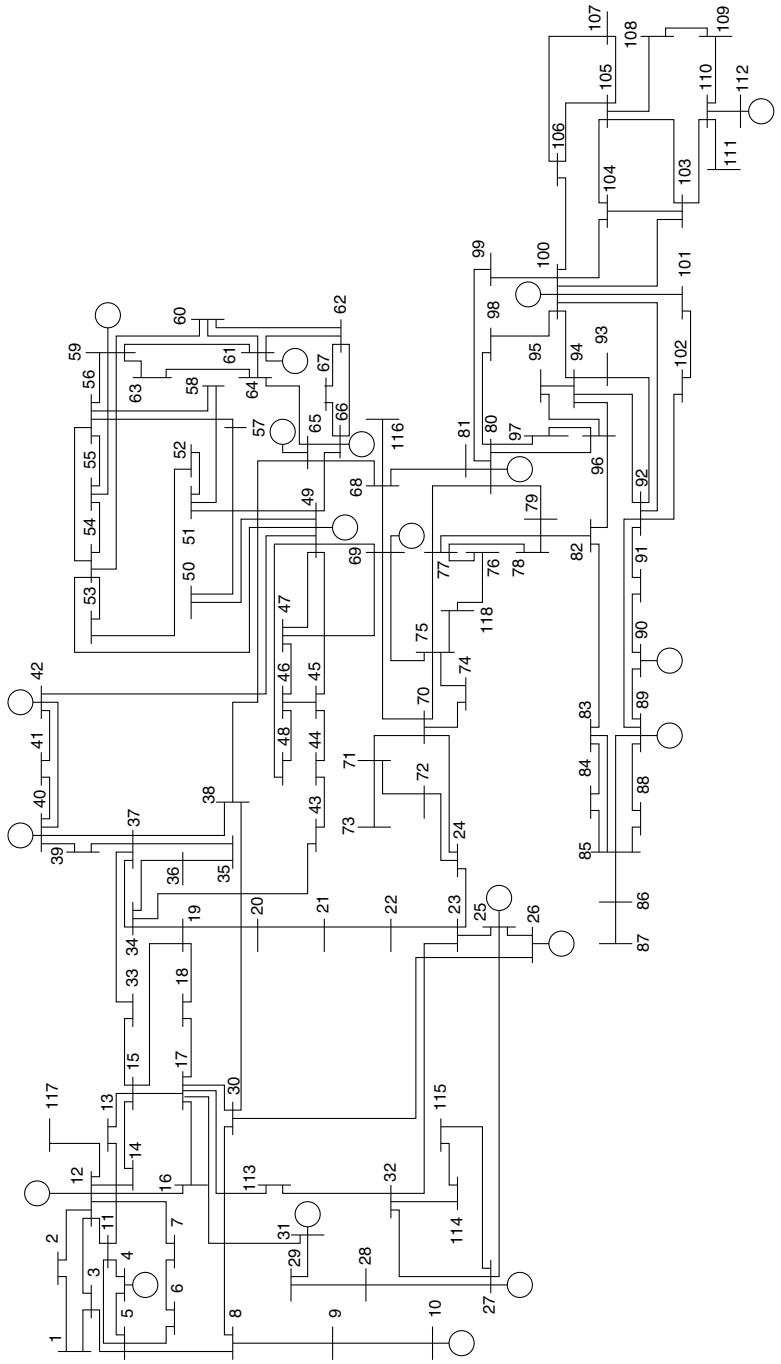


FIGURE 4.22
IEEE 118 bus system

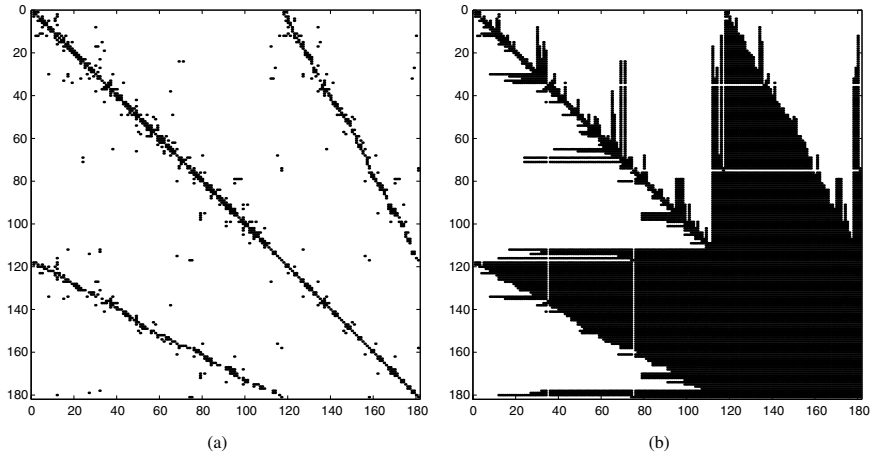


FIGURE 4.23
IEEE 118 bus system (a) Jacobian and (b) LU factors

4.23(b). This matrix has 14849 nonzero elements. Notice that the two subdiagonals have created a large number of fills extending between them and the main diagonal.

Figure 4.24(a) shows the structure of the Jacobian reordered according to Scheme 0. In this reordering, the presence of the subdiagonals is gone. The LU factorization of the Scheme 0 reordered Jacobian yields the structure shown in Figure 4.24(b). This matrix has only 1869 nonzero elements, which is almost an order of magnitude reduction from the nonordered Jacobian.

Figure 4.25(a) shows the structure of the power flow Jacobian reordered according to Scheme I. Note how the elements are gradually pulling into the main diagonal, which leads to a decrease in the number of fills. The LU factorization of the Scheme I ordering is shown in Figure 4.25(b), which has 1455 nonzero elements.

Finally, Figure 4.26(a) shows the structure of the Scheme II reordered Jacobian which yields the LU factorization in Figure 4.26(b). This ordering yields only 1421 nonzero elements, which is more than a full order of magnitude reduction. The LU factorization solution time for a sparse matrix is on the order of n^2 multiplications and divisions. The nonreordered power flow solution would require on the order of 220.5×10^6 multiplications and divisions per iteration, whereas the Scheme II reordered power flow solution would require only 2.02×10^6 multiplications and divisions. Thus the solution of the reordered system is over 100 times faster than the original system! When the solution time is multiplied by the number of iterations in a Newton–Raphson power flow or by the number of time steps in a time-domain integration, it would be computationally foolhardy not to use a reordering scheme.

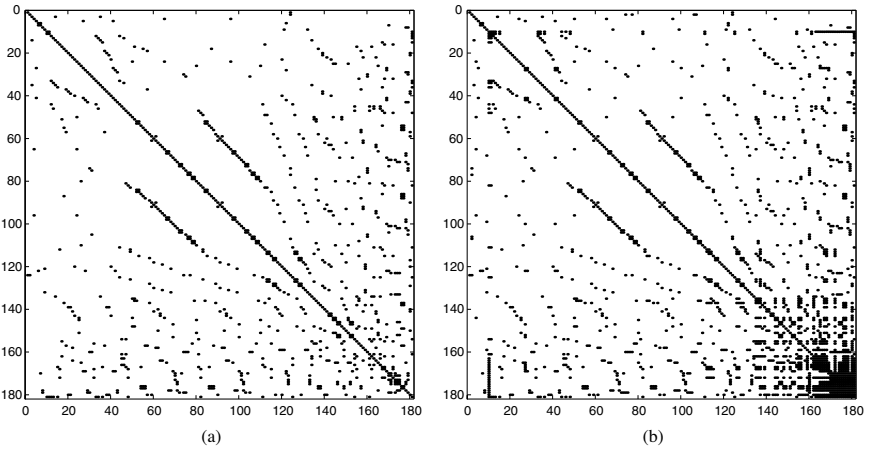


FIGURE 4.24
IEEE 118 bus system Scheme 0 (a) Jacobian and (b) LU factors

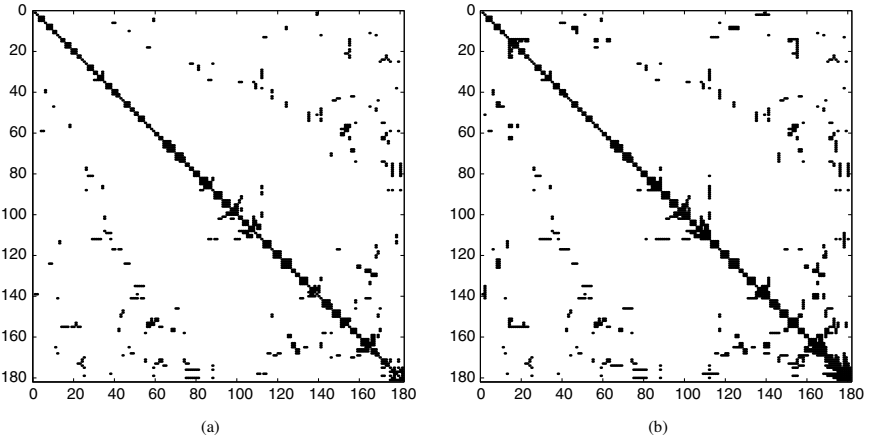


FIGURE 4.25
IEEE 118 bus system Scheme I (a) Jacobian and (b) LU factors

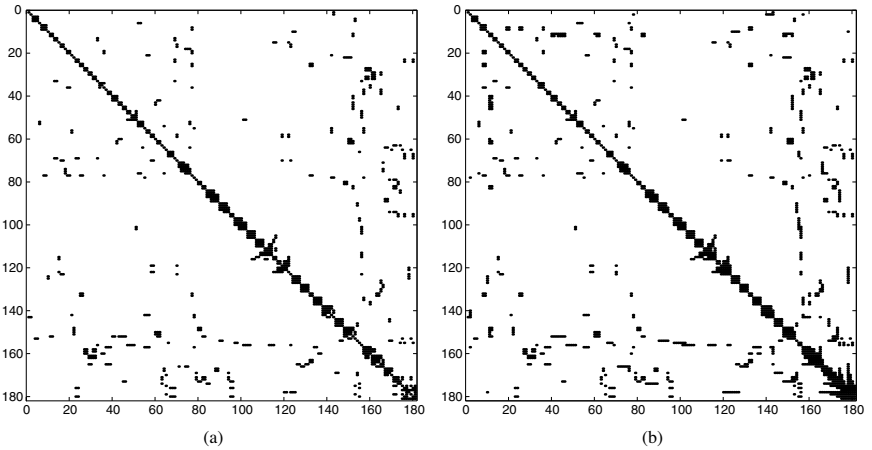


FIGURE 4.26
IEEE 118 bus system Scheme II (a) Jacobian and (b) LU factors

4.5 Problems

1. Verify Equations (4.1) and (4.2) for calculating α and β.

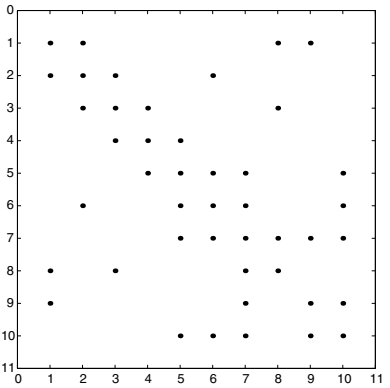
2. Let A and B be two sparse (square) matrices of the same dimension. How can the graph of $C = A + B$ be characterized with respect to the graphs of A and B?

3. Consider the matrix

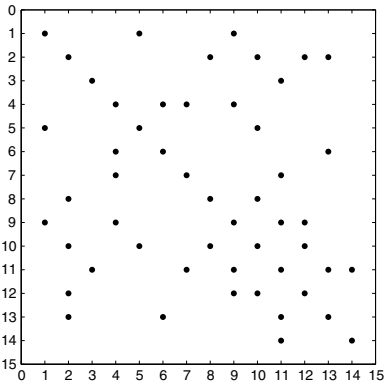
$$A = \begin{bmatrix} * & * & & & * & \\ * & * & * & & & * \\ & * & * & & & \\ & & & * & * & \\ * & & & * & * & * \\ & * & & & * & * \end{bmatrix}$$

   (a) Draw the graphical representation of A. How many multiplications and divisions will the LU factorization of A require?

   (b) Reorder the matrix using the new node numbering $\phi = [1, 3, 4, 2, 5, 6]$. Draw the graphical representation of the reordered matrix. How many multiplications and divisions will the LU factorization of the reordered matrix require?



(a)



(b)

FIGURE 4.27
Sparse Test Systems

4. For the matrices shown in Figure 4.27

   (a) Using the given ordering, compute $\alpha + \beta$.

   (b) Reorder the nodes in the network using Scheme 0 discussed in class. Compute $\alpha + \beta$ for this ordering.

   (c) Reorder the nodes in the network using Scheme I. Compute $\alpha + \beta$ for this ordering.

   (d) Reorder the nodes in the network using Scheme II. Compute $\alpha + \beta$ for this ordering.

5. For the sparse A given below:

	NROW	NCOL	NIR	NIC	Value	FIR	FIC
1	8	4	16	0	5.8824	27	27
2	9	9	0	0	–30.2732	34	34
3	10	7	9	0	21.2766	17	30
4	7	10	0	5	21.2766	33	14
5	8	10	0	21	10.0000	25	12
6	7	9	4	7	11.7647	26	22
7	8	9	5	2	18.5185	19	19
8	5	4	10	1	7.1429	13	18
9	10	8	21	0	10.0000	24	6
10	5	5	22	26	–24.5829	3	4
11	4	5	32	10	7.1429		
12	3	5	18	11	5.0000		
13	8	3	1	0	11.1111		
14	1	4	0	29	13.3333		
15	6	6	20	16	–23.8376		
16	8	6	28	0	11.3636		
17	3	2	31	0	8.3333		
18	3	8	0	32	11.1111		
19	7	7	6	24	–33.0413		
20	6	8	0	28	11.3636		
21	10	10	0	0	–31.2566		
22	5	6	0	15	12.5000		
23	9	8	2	9	18.5185		
24	9	7	23	3	11.7647		
25	5	3	8	13	5.0000		
26	6	5	15	0	12.5000		
27	1	1	14	33	–13.3258		
28	8	8	7	23	–56.8046		
29	4	4	11	8	–26.3260		
30	2	3	0	31	8.3333		
31	3	3	12	25	–24.3794		
32	4	8	0	20	5.8824		
33	4	1	29	0	13.3333		
34	2	2	30	17	-8.3183		

Complete the LU factors in the sparse matrix given below. Do *not* explicitly create the matrix A. Show fully how you arrived at your answer.

	NROW	NCOL	NIR	NIC	Value	FIR	FIC
1	8	4	36	0	5.8824	27	27
2	9	9	37	38	–13.9405	34	34
3	10	7	9	0	21.2766	17	30
4	7	10	0	5	–0.6439	33	14
5	8	10	0	37	–0.3541	25	12
6	7	9	4	7	–0.3561	26	22
7	8	9	5	2	–0.6558	19	19
8	5	4	10	1	7.1429	13	18
9	10	8	38	0	10.0000	24	6
10	5	5	22	26	–19.0943	3	4
11	4	5	32	10	–0.5501		
12	3	5	18	11	–0.3119		
13	8	3	1	0	11.1111		
14	1	4	0	29	–1.0006		
15	6	6	20	16	–15.6545		
16	8	6	28	0	15.7506		
17	3	2	31	0	8.3333		
18	3	8	0	32	–0.6931		
19	7	7	6	24	–33.0413		
20	6	8	0	28	–1.0061		
21	10	10	0	0	0.3143		
22	5	6					
23	9	8	2	9	18.5185		
24	9	7	23	3	11.7647		
25	5	3	8	13	5.0000		
26	6	5					
27	1	1	14	33	–13.3258		
28	8	8	7	23	–28.2397		
29	4	4	11	8	–12.9852		
30	2	3	0	31	–1.0018		
31	3	3	12	25	–16.0311		
32	4	8	0	35	–0.4530		
33	4	1	29	0	13.3333		
34	2	2	30	17	–8.3183		
35	5	8					
36	8	5					
37	9	10					
38	10	9					

6. Write a subroutine *sparmat* for sparse matrix storage that will

   - Read in data line by line in the format

     $$i \quad j \quad a_{ij}$$

     where the end of the data is signified by a 0 in the first column.
   - Sequentially build the vectors FIR, FIC, NIR, NIC, NROW, NCOL, and Value as defined in class. **Do not explicitly create the matrix** A.

7. Write a subroutine *sparvec* for sparse vector storage that will

   - Read in data line by line in the format

     $$i \quad b_i$$

     where the end of the data is signified by a 0 in the first column.
   - Sequentially build the vectors index, next, and Value. **Do not explicitly create the vector** b.

8. For the data given below, use *sparmat* and *sparvec* to create the sparse storage vectors.

A matrix			b vector	
i	j	a_{ij}	i	b_i
7	10	2.0	2	5
2	6	1.5	9	2
9	1	4.7	3	–1
5	5	–18.5		
8	7	2.8		
1	1	–15.0		
4	3	3.8		
6	7	6.1		
8	3	3.3		
5	7	4.4		
10	6	2.5		
6	5	1.1		
3	2	5.2		
7	8	2.9		
9	9	–12.1		
3	4	3.0		
7	6	5.6		
10	9	4.7		
8	8	–10.8		
1	9	4.5		
7	5	3.9		
5	6	7.2		
9	10	4.9		
5	4	0.8		
8	1	3.4		
5	10	4.5		
2	3	5.0		
6	6	–9.8		
7	9	1.8		
4	5	0.7		
7	7	–21.2		
1	2	4.4		
10	5	5.4		
3	8	3.1		
9	7	1.6		
4	4	–5.1		
6	10	2.7		
10	10	–16.9		
2	1	4.7		
3	3	–17.7		
1	8	3.5		
10	7	2.1		
2	2	–13.0		
6	2	1.2		

9. Write a subroutine *sparLU* that modifies your LU factorization routine to incorporate the sparse storage vectors of Problem 5 and apply it to the data of Problem 7 to compute the sparse LU factors (in sparse vector form).

10. Write a subroutine *sparsub* that modifies your forward/backward substitution routine *sub* to incorporate the sparse storage vectors of Problem 2 and apply it to the data of Problem 7 to solve the sparse linear system

$$Ax = b$$

11. Write a subroutine *scheme0* that will input the sparse vectors FIR, FIC, NIR, NIC, NROW, NCOL, and Value and will output the same vectors reordered according to Scheme 0, and calculate $\alpha + \beta$.

12. Write a subroutine *scheme1* that will input the sparse vectors FIR, FIC, NIR, NIC, NROW, NCOL, and Value and will output the same vectors reordered according to Scheme I, and calculate $\alpha + \beta$.

13. Write a subroutine *scheme2* that will input the sparse vectors FIR, FIC, NIR, NIC, NROW, NCOL, and Value and will output the same vectors reordered according to Scheme II, and calculate $\alpha + \beta$.