Chapter 3
Introduction to Matrix Factorizations

If numerical analysts understand anything, surely it must be Gaussian elimination. This is the oldest and truest of numerical algorithms . . . This algorithm has been so successful that to many of us, Gaussian elimination and $Ax = b$ are more or less synonymous. – Trefethen (1985).

Gaussian elimination is the standard method for solving a system of linear equations. As such, it is one of the most ubiquitous numerical algorithms and plays a fundamental role in scientific computation. – Higham (2011)

This chapter introduces the basic concepts of Gaussian elimination and its formulation as a matrix factorization that can be expressed in a number of mathematically equivalent but algorithmically different ways.

Using unweighted graphs to capture the sparsity structures of matrices during Gaussian elimination is simplified by assuming that the result of adding, subtracting, or multiplying two nonzeros is nonzero. It follows that if $A = LU$ and $\mathcal{E}_L$ denotes the set of (directed) edges of the digraph $\mathcal{G}(L)$, then for $i > j$

$$a_{ij} \neq 0 \text{ implies } (i \to j) \in \mathcal{E}_L.$$

This is the **non-cancellation assumption**. It allows the following observation.

Observation 3.1 *The sparsity structures of the LU factors of A satisfy*

$$\mathcal{S}\{A\} \subseteq \mathcal{S}\{L + U\}.$$

*That is, the factors may contain entries that lie outside the sparsity structure of A. Such entries are termed **filled entries**, and together the filled entries are called the **fill-in**. The graph obtained from $\mathcal{G}(A)$ by adding the fill-in is called the **filled graph**.*

Numerical cancellations in LU factorizations rarely happen, and in general, they are difficult to predict, particularly in floating-point arithmetic. Thus, such

J. Scott, M. Tůma, *Algorithms for Sparse Linear Systems*, Nečas Center Series,
https://doi.org/10.1007/978-3-031-25820-6_3

accidental zeros are not normally exploited in implementations, and we will ignore the possibility of their occurrence.

3.1 Gaussian Elimination: An Overview

The traditional way of describing Gaussian elimination is based on the systematic column-by-column annihilation of the entries in the lower triangular part of A. Assuming A is factorizable, this can be written formally as sequential multiplications by **column elimination matrices** that yield the **elimination sequence**

$$A = A^{(1)}, A^{(2)}, \ldots, A^{(n)} \tag{3.1}$$

of partially eliminated matrices as follows:

$$A^{(1)} \to A^{(2)} = C_1 A^{(1)} \to A^{(3)} = C_2 C_1 A^{(1)} \to \ldots \to A^{(n)} = C_{n-1} \ldots C_2 C_1 A^{(1)}.$$

The unit lower triangular matrices C_i $(1 \le i \le n-1)$ are the column elimination matrices. Elementwise, assuming $a_{11} = a_{11}^{(1)} \neq 0$, the first step $C_1 A^{(1)} = A^{(2)}$ is

$$\begin{pmatrix} 1 & & & \\ -a_{21}^{(1)}/a_{11}^{(1)} & 1 & & \\ -a_{31}^{(1)}/a_{11}^{(1)} & & 1 & \\ \vdots & & & 1 \\ -a_{n1}^{(1)}/a_{11}^{(1)} & & & 1 \end{pmatrix} \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \ldots & a_{1n}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \ldots & a_{2n}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & \ldots & a_{3n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & \ldots & a_{nn}^{(1)} \end{pmatrix} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \ldots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \ldots & a_{2n}^{(2)} \\ 0 & a_{32}^{(2)} & \ldots & a_{3n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(2)} & \ldots & a_{nn}^{(2)} \end{pmatrix},$$

and provided $a_{22}^{(2)} \neq 0$, the second step $C_2 A^{(2)} = A^{(3)}$ is

$$\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & -a_{32}^{(2)}/a_{22}^{(2)} & 1 & & \\ & \vdots & & 1 & \\ & -a_{n2}^{(2)}/a_{22}^{(2)} & & & 1 \end{pmatrix} \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \ldots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \ldots & a_{2n}^{(2)} \\ 0 & a_{32}^{(2)} & \ldots & a_{3n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(2)} & \ldots & a_{nn}^{(2)} \end{pmatrix} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \ldots & \ldots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \ldots & \ldots & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \ldots & a_{3n}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & a_{n3}^{(3)} & \ldots & a_{nn}^{(3)} \end{pmatrix}.$$

The k-th **partially eliminated matrix** is $A^{(k)}$. The **active entries** in $A^{(k)}$ are denoted by $a_{ij}^{(k)}$, $1 \le k \le i, j \le n$ (in the sparse case, many of the entries are zero), and the $(n-k+1) \times (n-k+1)$ submatrix of $A^{(k)}$ containing the active entries is termed its **active submatrix**. The graph associated with the active submatrix is the

k-th **elimination graph** and is denoted by $\mathcal{G}^k$. If $\mathcal{S}\{A\}$ is nonsymmetric, then $\mathcal{G}^k$ is a digraph.

The inverse of each C_k is the unit lower triangular matrix that is obtained by changing the sign of all the off-diagonal entries, and because the product of unit lower triangular matrices is a unit lower triangular matrix, it is clear that provided $a_{kk}^{(k)} \neq 0\ (1 \leq k < n)$

$$A = A^{(1)} = C_1^{-1} C_2^{-1} \ldots C_{n-1}^{-1} A^{(n)} = LU,$$

where the unit lower triangular matrix L is the product $C_1^{-1} C_2^{-1} \ldots C_{n-1}^{-1}$ and $U = A^{(n)}$ is an upper triangular matrix. The subdiagonal entries of L are the negative of the subdiagonal entries of the matrix $C_1 + C_2 + \ldots + C_{n-1}$. If A is a symmetric positive definite (SPD) matrix, then setting $U = DL^T$, the LU factorization can be written as

$$A = LDL^T,$$

which is the square root-free Cholesky factorization. Alternatively, it can be expressed as the Cholesky factorization

$$A = (LD^{1/2})(LD^{1/2})^T,$$

where the lower triangular matrix $LD^{1/2}$ has positive diagonal entries.

The process of performing an LU factorization can be rewritten in the **generic form** given in Algorithm 3.1. Here each l_{ik} is called a **multiplier**, and the $a_{kk}^{(k)}$ are called **pivots**. The assumption that A is factorizable implies $a_{kk}^{(k)} \neq 0$ for all k. Algorithm 3.1 comprises three nested loops. There are six ways of assigning the indices to the loops, with the loops having different ranges. The performance of the variants can differ significantly depending on the computer architecture. The key difference is the way the data are accessed from the factorized part of matrix and

ALGORITHM 3.1 Generic LU factorization
Input: Factorizable matrix A.
Output: LU factorization $A = LU$.

```
1: for ———— do
2:     for ———— do
3:         for ———— do
4:             l_ik = a_ik^(k) / a_kk^(k)
5:             a_ij^(k+1) = a_ij^(k) − l_ik a_kj^(k)
6:         end for
7:     end for
8: end for
```

applied to the part that is not yet factorized. But in exact arithmetic, they result in the same L and U, which allows any of them to be used to demonstrate theoretical properties of LU factorizations. To identify the variants, names that derive from the order in which the indices are assigned to the loops can be used. The kij and kji variants are called **submatrix LU factorizations**. The schemes jik and jki compute the factors by columns and are called **column factorizations**. The final two are **row factorizations** because they proceed by rows. A row factorization can be considered as a column LU factorization applied to A^T.

3.1.1 Submatrix LU Factorizations

Each outermost step of the submatrix LU variants computes one row of U and one column of L. The first step ($k = 1$) is

$$C_1 A = \begin{pmatrix} 1 & \\ -A_{2:n,1}/a_{11} & I \end{pmatrix} \begin{pmatrix} a_{11} & A_{1,2:n} \\ A_{2:n,1} & A_{2:n,2:n} \end{pmatrix} = \begin{pmatrix} a_{11} & A_{1,2:n} \\ & S \end{pmatrix},$$

where the $(n-1) \times (n-1)$ active submatrix

$$S = A_{2:n,2:n} - A_{2:n,1} A_{1,2:n}/a_{11} = A_{2:n,2:n} - L_{2:n,1} U_{1,2:n}$$

is the **Schur complement** of A with respect to a_{11}. If A is factorizable, then so too is S and the process can be repeated.

More generally, the operations performed at each step k correspond to a sequence of rank-one updates. The resulting Schur complement can be written in terms of entries of the matrices from the elimination sequence and entries of the computed factors. After $k-1$ steps ($1 < k \le n$), the $(n-k+1) \times (n-k+1)$ Schur complement of A with respect to its $(k-1) \times (k-1)$ principal leading submatrix is the active submatrix of the partially eliminated matrix $A^{(k)}$ given by

$$\begin{aligned} S^{(k)} &= \begin{pmatrix} a_{kk} & \dots & a_{kn} \\ \vdots & \ddots & \vdots \\ a_{nk} & \dots & a_{nn} \end{pmatrix} - \sum_{j=1}^{k-1} \begin{pmatrix} l_{kj} \\ \vdots \\ l_{nj} \end{pmatrix} \begin{pmatrix} u_{jk} & \dots & u_{jn} \end{pmatrix} \\ &= A_{k:n,k:n} - \sum_{j=1}^{k-1} L_{k:n,j} U_{j,k:n} \\ &= \begin{pmatrix} a_{kk}^{(k)} & \dots & a_{kn}^{(k)} \\ \vdots & \ddots & \vdots \\ a_{nk}^{(k)} & \dots & a_{nn}^{(k)} \end{pmatrix} = A_{k:n,k:n}^{(k)}. \end{aligned} \tag{3.2}$$

If A is SPD, then the Cholesky and LDLT factorizations that are special cases of the submatrix approach are termed **right-looking** (fan-out) factorizations.

3.1.2 Column LU Factorizations

In the column LU factorization, the outermost index in Algorithm 3.1 is j. For $j = 1$, $l_{11} = 1$, and the off-diagonal entries in column 1 of L are obtained by dividing the corresponding entries in column 1 of A by $u_{11} = a_{11}$. Assume $j - 1$ columns $(1 < j \leq n)$ of L and U have been computed. The partial column factorization can be expressed as

$$\begin{pmatrix} L_{1:j-1,1:j-1} \\ L_{j:n,1:j-1} \end{pmatrix} U_{1:j-1,1:j-1} = \begin{pmatrix} A_{1:j-1,1:j-1} \\ A_{j:n,1:j-1} \end{pmatrix}.$$

Column j of U and then column j of L are computed using the identities

$$U_{1:j-1,j} = L^{-1}_{1:j-1,1:j-1} A_{1:j-1,j}, \quad u_{jj} = a_{jj} - L_{j,1:j-1} U_{1:j-1,j},$$

and

$$l_{jj} = 1,\ L_{j+1:n,j} = (A_{j+1:n,j} - L_{j+1:n,1:j-1} U_{1:j-1,j})/u_{jj}.$$

Thus the strictly upper triangular part of column j of U is determined by solving the triangular system

$$L_{1:j-1,1:j-1} U_{1:j-1,j} = A_{1:j-1,j},$$

and the strictly lower triangular part of column j of L is computed as a linear combination of column $A_{j+1:n,j}$ of A and previously computed columns of L.

If A is symmetric and the pivots can be used in the order $1, 2, \ldots$ without modification, then there is the following link between its column LU and LDLT factorizations.

Observation 3.2 *The j-th diagonal entry d_{jj} $(1 \leq j \leq n)$ of the LDLT factorization of the symmetric matrix A is*

$$d_{jj} = u_{jj} = a_{jj} - \sum_{k=1}^{j-1} d_{kk} l_{jk}^2.$$

The L factor is the same as is computed by the column LU factorization; its computation can be written as

ALGORITHM 3.2 Basic column LU factorization with partial pivoting
Input: Nonsingular nonsymmetric matrix A.
Output: LU factorization $PA = LU$, where P is a row permutation matrix.

1: Interchange rows of A so that $|a_{11}| = \max\{|a_{i1}| \,|\, 1 \le i \le n\}$
2: $l_{11} = 1,\ u_{11} = a_{11},\ L_{2:n,1} = A_{2:n,1}/a_{11}$
3: **for** $j = 2 : n$ **do**
4: Solve $L_{1:j-1,1:j-1}U_{1:j-1,j} = A_{1:j-1,j}$
5: $z_{1:n-j+1} = A_{j:n,j} - L_{j:n,1:j-1}U_{1:j-1,j}$
6: Apply row interchanges to z, A and L so that $|z_1| = \max\{|z_i| \,|\, 1 \le i \le n - j + 1\}$.
7: $l_{jj} = 1$, $u_{jj} = z_1$ and $L_{j+1:n,j} = z_{2:n-j+1}/z_1$
8: **end for**

$$d_{jj}L_{j+1:n,j} = A_{j+1:n,j} - \sum_{k=1}^{j-1} L_{j+1:n,k}\, d_{kk}\, l_{jk}.$$

The U factor is equal to DL^T. Computing L and D in this way is called the ***left-looking*** *(fan-in) factorization.*

So far, we have assumed that A is factorizable. If A is nonsingular, then there exists a row permutation matrix P such that PA is factorizable (Theorem 1.1), and if there are zeros on the diagonal, then the rows can always be permuted to achieve a nonzero diagonal. Consider the simple 2×2 matrix A and its LU factorization

$$A = \begin{pmatrix} \delta & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & \\ \delta^{-1} & 1 \end{pmatrix} \begin{pmatrix} \delta & 1 \\ & 1 - \delta^{-1} \end{pmatrix}.$$

If $\delta = 0$, this factorization does not exist, and if δ is very small, then the entries in the factors involving δ^{-1} are very large. But interchanging the rows of A, we have

$$PA = \begin{pmatrix} 1 & 1 \\ \delta & 1 \end{pmatrix} = \begin{pmatrix} 1 & \\ \delta & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ & 1 - \delta \end{pmatrix},$$

which is valid for all $\delta \neq 1$. Algorithm 3.2 presents a basic column LU factorization scheme for nonsingular A. The interchanging of rows at each elimination step to select the entry of largest absolute value in its column as the next pivot is called **partial pivoting**. It avoids small pivots and results in an LU factorization of a row permuted matrix PA in which the absolute value of each entry of L is at most 1. In practice, partial pivoting (or another pivoting strategy) is incorporated into all LU factorization variants. Pivoting strategies are discussed in Chapter 7.

3.1.3 *Factorizations by Bordering*

The generic LU factorization scheme does not cover all possible approaches. An alternative is **factorization by bordering**. Set all diagonal entries of L to 1, and assume the first $k-1$ rows of L and first $k-1$ columns of U ($1 < k \leq n$) have been computed (that is, $L_{1:k-1,1:k-1}$ and $U_{1:k-1,1:k-1}$). At step k, the factors must satisfy

$$A_{1:k,1:k} = \begin{pmatrix} A_{1:k-1,1:k-1} & A_{1:k-1,k} \\ A_{k,1:k-1} & a_{kk} \end{pmatrix} = \begin{pmatrix} L_{1:k-1,1:k-1} & 0 \\ L_{k,1:k-1} & 1 \end{pmatrix} \begin{pmatrix} U_{1:k-1,1:k-1} & U_{1:k-1,k} \\ 0 & u_{kk} \end{pmatrix}.$$

Equating terms, the lower triangular part of row k of L and the upper triangular part of column k of U are obtained by solving

$$L_{k,1:k-1}U_{1:k-1,1:k-1} = A_{k,1:k-1},$$
$$L_{1:k-1,1:k-1}U_{1:k-1,k} = A_{1:k-1,k}.$$

The diagonal entry u_{kk} is then given by

$$u_{kk} = a_{kk} - L_{k,1:k-1}U_{1:k-1,k} \quad (\text{with } u_{11} = a_{11}).$$

3.2 Fill-in in Sparse Gaussian Elimination

Here we give some simple results that describe fill-in in the matrix factors; strategies to limit fill-in will be presented in Chapter 8. We start by looking at the rules that establish the positions of the entries in the factors. Assume $\mathcal{S}\{A\}$ is symmetric, and consider the elimination graph $\mathcal{G}^k$ at step k. Its vertices are the $n-k+1$ uneliminated vertices. Its edge set contains the edges in $\mathcal{G}(A)$ connecting these vertices and additional edges corresponding to filled entries produced during the first $k-1$ elimination steps. The sequence of graphs $\mathcal{G}^1 \equiv \mathcal{G}(A), \mathcal{G}^2, \ldots$ is generated recursively using **Parter's rule**:

To obtain the elimination graph $\mathcal{G}^{k+1}$ from $\mathcal{G}^k$, delete vertex k and add all possible edges between vertices that are adjacent to vertex k in $\mathcal{G}^k$.

Denoting $\mathcal{G}^k = (\mathcal{V}^k, \mathcal{E}^k)$ and $\mathcal{G}^{k+1} = (\mathcal{V}^{k+1}, \mathcal{E}^{k+1})$, this can be written as

$$\mathcal{V}^{k+1} = \mathcal{V}^k \setminus \{k\}, \ \mathcal{E}^{k+1} = \mathcal{E}^k \cup \{(i,j) \,|\, i,j \in adj_{\mathcal{G}^k}\{k\}\} \setminus \{(i,k) \,|\, i \in adj_{\mathcal{G}^k}\{k\}\}.$$

If $\mathcal{S}\{A\}$ is nonsymmetric, then the elimination graphs are digraphs and Parter's rule generalizes as follows:

To obtain the elimination graph $\mathcal{G}^{k+1}$ from $\mathcal{G}^k$, delete vertex k and add all edges $(i \xrightarrow{\mathcal{G}^{k+1}} j)$ such that $(i \xrightarrow{\mathcal{G}^k} k)$ and $(k \xrightarrow{\mathcal{G}^k} j)$.

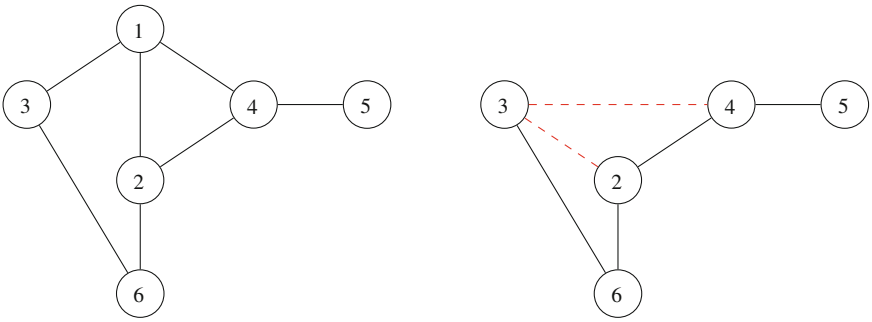

Figure 3.1 Illustration of Parter's rule. The original undirected graph $\mathcal{G} = \mathcal{G}^1$ and the elimination graph $\mathcal{G}^2$ that results from eliminating vertex 1 are shown on the left and right, respectively. The red dashed lines denote fill edges. The vertices $\{2, 3, 4\}$ become a clique.

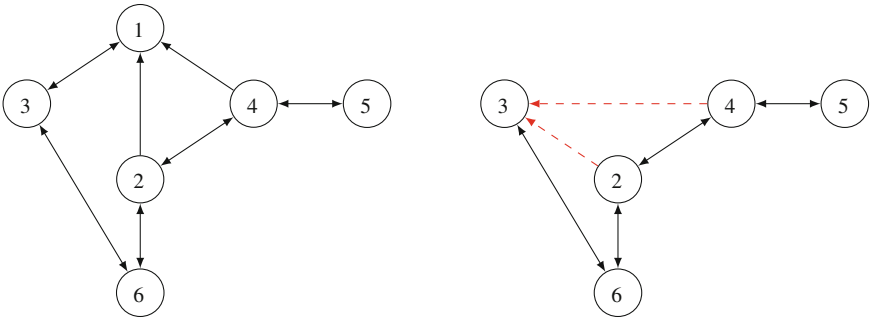

Figure 3.2 Illustration of Parter's rule for a nonsymmetric $\mathcal{S}\{A\}$. The original digraph $\mathcal{G} = \mathcal{G}^1$ and the directed elimination graph $\mathcal{G}^2$ that results from eliminating vertex 1 are shown on the left and right, respectively. The red dashed lines denote fill edges.

Simple examples are given in Figures 3.1 and 3.2.

In terms of graph theory, if $\mathcal{S}\{A\}$ is symmetric, then Parter's rule says that the adjacency set of vertex k becomes a clique when k is eliminated. Thus, Gaussian elimination systematically generates cliques. As the elimination process progresses, cliques grow or more than one clique join to form larger cliques, a process known as **clique amalgamation**. A clique with m vertices has $m(m-1)/2$ edges, but it can be represented by storing a list of its vertices, without any reference to edges. This enables important savings in both storage and data movement to be achieved during the symbolic phase of a direct solver.

The repeated application of Parter's rule specifies all the edges in $\mathcal{G}(L + L^T)$:

(i, j) is an edge of $\mathcal{G}(L + L^T)$ if and only if (i, j) is an edge of $\mathcal{G}(A)$ or (i, k) and (k, j) are edges of $\mathcal{G}(L + L^T)$ for some $k < i, j$.

This generalizes to a nonsymmetric matrix A and its LU factorization:

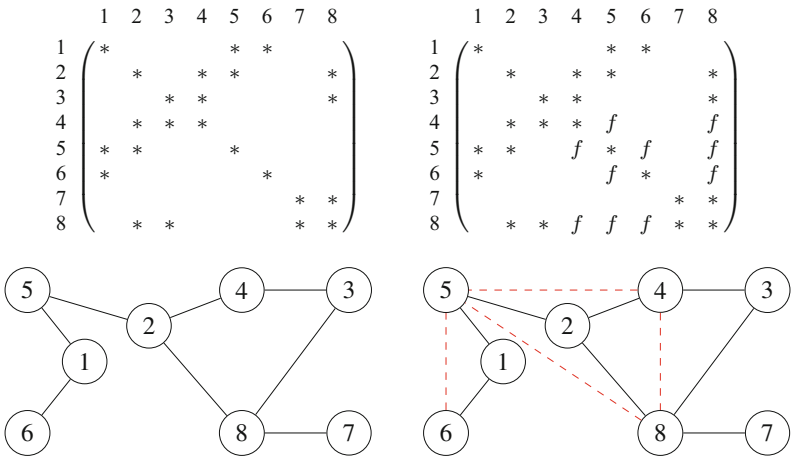

Figure 3.3 Example to illustrate fill-in during the factorization of a symmetric matrix, with the eliminations performed in the natural order. $\mathcal{S}\{A\}$ and $\mathcal{S}\{L + L^T\}$ are on the left and right, respectively, with the corresponding undirected graphs $\mathcal{G}(A)$ and $\mathcal{G}(L + L^T)$. Filled entries in $L + L^T$ are denoted by f. The red dashed lines in the filled graph $\mathcal{G}(L + L^T)$ correspond to filled entries.

$(i \rightarrow j)$ is an edge of the digraph $\mathcal{G}(L+U)$ if and only if $(i \rightarrow j)$ is an edge of the digraph $\mathcal{G}(A)$ or $(i \rightarrow k)$ and $(k \rightarrow j)$ are edges of $\mathcal{G}(L + U)$ for some $k < i, j$.

Parter's rule is a local rule that uses the dependency on nonzeros obtained in previous steps of the factorization. The following result, which uses the path notation of Section 2.2, fully characterizes the nonzero entries in the factors using only paths in $\mathcal{G}(A)$.

Theorem 3.1 (Rose et al. 1976; Rose & Tarjan 1978)

(a) Let $\mathcal{S}\{A\}$ be symmetric and $A = LL^T$. Then $(L + L^T)_{ij} \neq 0$ if and only if there is a fill-path $i \overset{\mathcal{G}(A)}{\underset{min}{\Longleftrightarrow}} j$.

(b) Let $\mathcal{S}\{A\}$ be nonsymmetric and $A = LU$. Then $(L + U)_{ij} \neq 0$ if and only if there is a fill-path $i \overset{\mathcal{G}(A)}{\underset{min}{\Longrightarrow}} j$.

The fill-paths may not be unique.

Figure 3.3 illustrates Theorem 3.1 for symmetric $\mathcal{S}\{A\}$. There is a filled entry in position $(8, 6)$ of L because there is a fill-path $8 \overset{\mathcal{G}(A)}{\underset{min}{\Longleftrightarrow}} 6$ given by the sequence of (undirected) edges $8 \longleftrightarrow 2 \longleftrightarrow 5 \longleftrightarrow 1 \longleftrightarrow 6$.

Corollary 3.2 characterizes edges of $\mathcal{G}^k$ in terms of reachable sets in the original graph $\mathcal{G}(A)$.

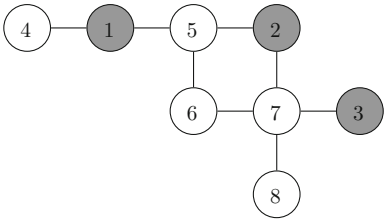

Figure 3.4 An example to illustrate reachable sets in $\mathcal{G}(A)$. The grey vertices 1, 2, and 3 are eliminated in the first three elimination steps ($\mathcal{V}^4 = \{1, 2, 3\}$).

Corollary 3.2 (Rose et al., 1976; George & Liu, 1980b)
Assume $\mathcal{S}\{A\}$ is symmetric. Let $\mathcal{V}^k$ be the set of $k-1$ vertices of $\mathcal{G}(A)$ that have already been eliminated, and let v be a vertex in the elimination graph $\mathcal{G}^k$. Then the set of vertices adjacent to v in $\mathcal{G}^k$ is the set $\mathcal{R}each(v, \mathcal{V}^k)$ of vertices reachable from v through $\mathcal{V}^k$ in $\mathcal{G}(A)$.

Proof The proof is by induction on k. The result holds trivially for $k = 1$ because $\mathcal{R}each(v, \mathcal{V}^1) = adj_{\mathcal{G}(A)}\{v\}$. Assume the result holds for $\mathcal{G}^1, \ldots, \mathcal{G}^k$ with $k \geq 1$, and let v be a vertex in the graph $\mathcal{G}^{k+1}$ that is obtained after eliminating v_k from $\mathcal{G}^k$. If v is not adjacent to v_k in $\mathcal{G}^k$, then $\mathcal{R}each(v, \mathcal{V}^{k+1}) = \mathcal{R}each(v, \mathcal{V}^k)$. Otherwise, if v is adjacent to v_k in $\mathcal{G}^k$, then $adj_{\mathcal{G}^{k+1}}\{v\} = \mathcal{R}each(v, \mathcal{V}^k) \cup \mathcal{R}each(v_k, \mathcal{V}^k)$. In both cases, Parter's rule implies that the new adjacency set is exactly equal to the vertices that are reachable from v through $\mathcal{V}^{k+1}$, that is, $\mathcal{R}each(v, \mathcal{V}^{k+1})$. □

Figure 3.4 depicts a graph $\mathcal{G}(A)$. The adjacency sets of the vertices in $\mathcal{G}^4$ that result from eliminating vertices $\mathcal{V}^4 = \{1, 2, 3\}$ are $adj_{\mathcal{G}^4}\{4\} = \mathcal{R}each(4, \mathcal{V}^4) = \{5\}$, $adj_{\mathcal{G}^4}\{5\} = \mathcal{R}each(5, \mathcal{V}^4) = \{4, 6, 7\}$, $adj_{\mathcal{G}^4}\{6\} = \mathcal{R}each(6, \mathcal{V}^4) = \{5, 7\}$, $adj_{\mathcal{G}^4}\{7\} = \mathcal{R}each(7, \mathcal{V}^4) = \{5, 6, 8\}$, and $adj_{\mathcal{G}^4}\{8\} = \mathcal{R}each(8, \mathcal{V}^4) = \{7\}$.

We remark that neither the local characterization of filled entries using Parter's rule nor Theorem 3.1 provides a direct answer as to whether a certain edge belongs to $\mathcal{G}(L+L^T)$ (or $\mathcal{G}(L+U)$); without performing the eliminations, they do not tell us whether a given entry of a factor of A is nonzero. Such questions are addressed by deeper theoretical and algorithmic results that are presented in subsequent chapters.

3.3 Triangular Solves

Once an LU factorization has been computed, the solution x of the linear system $Ax = b$ is computed by solving the lower triangular system

$$Ly = b, \tag{3.3}$$

followed by the upper triangular system

$$Ux = y. \tag{3.4}$$

Solving a system with a triangular matrix and dense right-hand side vector is straightforward. The solution of (3.3) can be computed using **forward substitution** in which the component y_1 is determined from the first equation, substitute it into the second equation to obtain y_2, and so on. Once y is available, the solution of (3.4) can be obtained by **back substitution** in which the last equation is used to obtain x_n, which is then substituted into equation $n-1$ to obtain x_{n-1}, and so on. Algorithm 3.3 is a simple lower triangular solve for dense b. If L is unit lower triangular, step 3 is not needed.

ALGORITHM 3.3 Forward substitution: lower triangular solve $Ly = b$ with b dense
Input: Lower triangular matrix L with nonzero diagonal entries and dense right-hand side b.
Output: The dense solution vector y.

```
1: Initialise y = b
2: for j = 1 : n do
3:     y_j = y_j / l_jj
4:     for i = j + 1 : n do
5:         if l_ij ≠ 0 then
6:             y_i = y_i − l_ij y_j
7:         end if
8:     end for
9: end for
```

When b is sparse, the solution y is also sparse. In particular, if in Algorithm 3.3 $y_k = 0$, then the outer loop with $j = k$ can be skipped. Furthermore, if $b_1 = b_2 = \ldots = b_k = 0$ and $b_{k+1} \neq 0$, then $y_1 = y_2 = \ldots = y_k = 0$. Scanning y to check for zeros adds $O(n)$ to the complexity. But if the set of indices $\mathcal{J} = \{j \mid y_j \neq 0\}$ is known beforehand, then Algorithm 3.3 can be replaced by Algorithm 3.4. A possible way to determine $\mathcal{J}$ is discussed later (Theorem 5.2).

Note that the combined effect of forward substitution (3.3) followed by back substitution (3.4) often results in the final solution vector x being dense. This is the case if $y_n \neq 0$ and U has an entry in each off-diagonal row i ($1 \leq i < n$).

3.4 Reducibility and Block Triangular Forms

The performance of algorithms for computing factorizations of sparse matrices can frequently be significantly enhanced by first permuting A to have a block form or by

ALGORITHM 3.4 Forward substitution: lower triangular solve $Ly = b$ with b sparse
Input: Lower triangular matrix L with nonzero diagonal entries, sparse vector b and the set $\mathcal{J}$ of indices j such that $y_j \neq 0$.
Output: The sparse solution vector y.

```
1: Initialise y = b
2: for j ∈ J do                          ▷ Take indices from J in increasing order
3:     y_j = y_j / l_jj
4:     for i = j + 1 : n do
5:         if l_ij ≠ 0 then
6:             y_i = y_i − l_ij y_j
7:         end if
8:     end for
9: end for
```

partitioning A into blocks. Permuting to block form is closely connected to matrix reducibility. A is said to be **reducible** if there is a permutation matrix P such that

$$PAP^T = \begin{pmatrix} A_{p_1,p_1} & A_{p_1,p_2} \\ 0 & A_{p_2,p_2} \end{pmatrix},$$

where A_{p_1,p_1} and A_{p_2,p_2} are nontrivial square matrices (that is, they are of order at least 1). If A is not reducible, it is **irreducible**. If A is structurally symmetric, then $A_{p_1,p_2} = 0$ and PAP^T is block diagonal. The following example illustrates that a one-sided permutation can transform an irreducible matrix A into a reducible matrix AQ.

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & \\ 1 & & \end{pmatrix}, \quad Q = \begin{pmatrix} & & 1 \\ & 1 & \\ 1 & & \end{pmatrix}, \quad AQ = \begin{pmatrix} 1 & 1 & 1 \\ & 1 & 1 \\ & & 1 \end{pmatrix}.$$

A matrix A is said to be a **Hall matrix** (or has the **Hall property**) if every set of k columns has nonzeros in at least k rows ($1 \leq k \leq n$). A is a **strong Hall matrix** (or has the **strong Hall property**) if every set of k columns ($1 \leq k < n$) has nonzeros in at least $k + 1$ rows. The strong Hall property trivially implies the Hall property. The Hall property applies to rectangular $m \times n$ matrices with $m \geq n$. If A is square, then A has the strong Hall property if and only if the directed graph $\mathcal{G}(A)$ is strongly connected.

The following theorem is an important consequence of reducibility.

Theorem 3.3 (Brualdi & Ryser 1991)
Given a nonsingular nonsymmetric matrix A, there exists a permutation matrix P such that

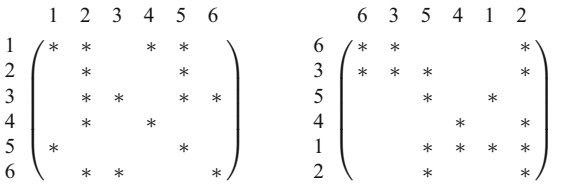

Figure 3.5 The sparsity patterns of A (left) and the upper block triangular form PAP^T with two blocks $A_{ib,ib}$, $i = 1, 2$, of orders 2 and 4 (right).

$$PAP^T = \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,nb} \\ 0 & A_{2,2} & \cdots & A_{2,nb} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{nb,nb} \end{pmatrix}, \tag{3.5}$$

where the square matrices $A_{ib,ib}$ on the diagonal are irreducible. The set $\{A_{ib,ib} \mid 1 \le ib \le nb\}$ is uniquely determined (but the blocks may appear on the diagonal in a different order). The order of the rows and columns within each $A_{ib,ib}$ may not be unique.

The **upper block triangular form** (3.5) is also known as the **Frobenius normal form**. It is said to be nontrivial if $nb > 1$, and this is the case if A does not have the strong Hall property. An example of a matrix that can be symmetrically permuted to block triangular form with $nb = 2$ is given in Figure 3.5.

In practice, many of the blocks in (3.5) are either sparse or zero blocks. Assuming the blocks $A_{ib,ib}$ on the diagonal are all nonsingular, an LU factorization of each can be computed independently. These can then be used to solve the permuted system $PAP^T y = c$ as a sequence of nb smaller problems, as outlined in Algorithm 3.5. The solution of the original system $Ax = b$ follows by setting $c = Pb$ and $x = P^T y$. Because the algorithms used to transform A into a block triangular form are typically graph-based (and do not use the numerical values of the entries of A), pivoting needs to be incorporated within the factorization of the diagonal blocks. Algorithm 3.5 employs partial pivoting for this.

The transversal of a matrix A is the set of its nonzero diagonal elements. A has a **full** or **maximum transversal** if all its diagonal entries are nonzero. There exist permutation matrices P and Q such that PAQ has a full transversal matrix if and only if A has the Hall property. Moreover, if A is nonsingular, then it can be nonsymmetrically permuted to have a full transversal. However, the converse is clearly not true (for example, a matrix with all its entries equal to one has a full transversal, but it is singular). Permuting A to have a full transversal will be discussed in Section 6.3.

If A has a full transversal, then there exists a permutation matrix P_s such that $P_s A P_s^T$ has the form (3.5). In other words, once A has a full transversal, a symmetric permutation is sufficient to obtain the form (3.5). Finding P_s is identical

ALGORITHM 3.5 Solve a sparse linear system in upper block triangular form
Input: Upper block triangular matrix (3.5) and a conformally partitioned right-hand side vector c.
Output: The conformally partitioned solution vector y.

1: **for** $ib = 1 : nb$ **do** ▷ LU factorizations of the $A_{ib,ib}$ blocks can be performed in parallel
2: Compute $P_{ib}A_{ib,ib} = L_{ib}U_{ib}$ ▷ Sparse LU factorization with partial pivoting
3: **end for**
4: Solve $L_{nb}U_{nb}\, y_{nb} = P_{nb}c_{nb}$ ▷ Perform forward and back substitutions
5: **for** $ib = nb - 1 : 1$ **do**
6: **for** $jb = ib + 1 : nb$ **do**
7: $c_{ib} = c_{ib} - A_{ib,jb}y_{jb}$ ▷ Sparse matrix-vector operation (skip if $A_{ib,jb} = 0$)
8: **end for**
9: Solve $L_{ib}U_{ib}\, y_{ib} = P_{ib}c_{ib}$ ▷ Perform forward and back substitutions
10: **end for**

to finding the strongly connected components (SCCs) of the digraph $\mathcal{G}(A) = (\mathcal{V}, \mathcal{E})$ (Section 2.3). To find the SCCs, $\mathcal{V}$ is partitioned into non-empty subsets $\mathcal{V}_i$ with each vertex belonging to exactly one subset. Each vertex i in the **quotient graph** corresponds to a subset $\mathcal{V}_i$, and there is an edge in the quotient graph with endpoints i and j if $\mathcal{E}$ contains at least one edge with one endpoint in $\mathcal{V}_i$ and the other in $\mathcal{V}_j$. The **condensation** (or component graph) of a digraph is a quotient graph in which the SCCs form the subsets of the partition, that is, each SCC is contracted to a single vertex. This reduction provides a simplified view of the connectivity between components. An example is given in Figure 3.6. It has five SCCs: $\{p, q, r\}$, $\{s, t, u\}$, $\{v\}$, $\{w\}$, and $\{x\}$.

The following result gives the relationship between SCCs and DAGs.

Theorem 3.4 (Sharir 1981; Cormen et al. 2009)
The condensation $\mathcal{G}_C$ of a digraph is a DAG (directed acyclic graph).

Because any DAG can be topologically ordered, $\mathcal{G}_C = (\mathcal{V}_C, \mathcal{E}_C)$ can be topologically ordered, and if $\mathcal{V}_i$ and $\mathcal{V}_j$ are contracted to s_i and s_j and $(s_i \longrightarrow s_j) \in \mathcal{E}_C$, then $s_i < s_j$. It follows that to permute A to block triangular form it is sufficient to find the SCCs of $\mathcal{G}(A)$. That is, topologically ordering the vertices of the condensation $\mathcal{G}_C$ induced by the SCCs is the quotient graph that implies the block triangular form. There are many ways to find SCCs, one of which is Tarjan's algorithm (Algorithm 3.6). The key idea here is that vertices of an SCC form a subtree in the DFS spanning tree of the graph. The algorithm performs depth-first searches, keeping track of two properties for each vertex v: when v was first

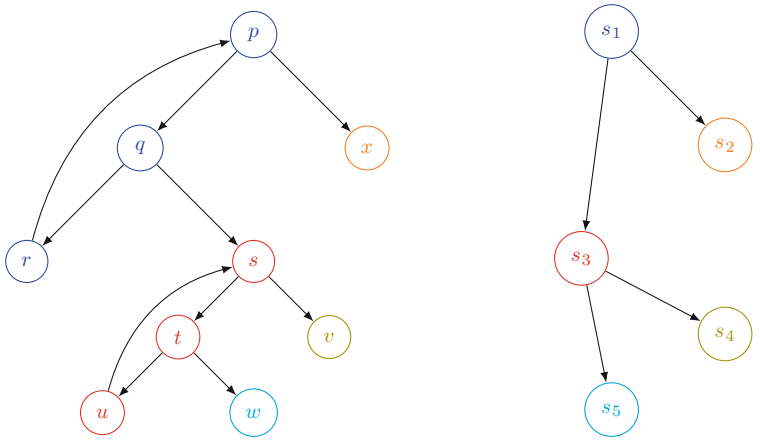

Figure 3.6 An illustration of the strong components of a digraph. On the left, the five SCCs are denoted using different colours and on the right is the condensation DAG $\mathcal{G}_C$ formed by the SCCs.

encountered (held in $invorder(v)$) and the lowest numbered vertex that is reachable from v (called the low-link value and held in $lowlink(v)$). It pushes vertices onto a stack as it goes and outputs a SCC when it finds a vertex for which $invorder(v)$ and $lowlink(v)$ are the same. The value $lowlink(v)$ is computed during the DFS from v, as this finds the vertices that are reachable from v.

In Algorithm 3.6, the variable $index$ is the DFS vertex number counter that is incremented when an unvisited vertex is visited. S is the vertex stack. It is initially empty and is used to store the history of visited vertices that are not yet committed to an SCC. Vertices are added to the stack in the order in which they are visited. The outermost loop of the algorithm visits each vertex that has not yet been visited, ensuring vertices that are not reachable from the starting vertex are eventually visited. The recursive function **scomp_step** performs a single DFS, finding all descendants of vertex v, and reporting all SCCs for that subgraph. When a vertex v finishes recursing, if $lowlink(v) = invorder(v)$, then it is the root vertex of an SCC comprising all of the vertices above it on the stack. The algorithm pops the stack up to and including v; these popped vertices form an SCC. The algorithm is linear in the number of edges and vertices, that is, it is of complexity $O(|\mathcal{V}|+|\mathcal{E}|)$.

3.5 Block Partitioning

In this section, we assume that $\mathcal{S}\{A\}$ is symmetric and $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is the adjacency graph of A.

ALGORITHM 3.6 Tarjan's algorithm to find the strongly connected components (SCCs) of a digraph
Input: Digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.
Output: Strongly connected components of $\mathcal{G}$, determined one-by-one.

```
1: V_v = ∅, S = (), index = 0,                ▷ Each vertex is initially unvisited
2: for each v ∈ V do
3:     if v ∉ V_v then
4:         scomp_step(v)
5:     end if
6: end for
7: recursive function (scomp_step(v))
8:     V_v = V_v ∪ {v}                        ▷ Add v to the set of visited vertices
9:     index = index + 1                      ▷ Set the index for v to smallest unused index
10:    invorder(v) = index, lowlink(v) = index
11:    push(S, v)                             ▷ Put v on the stack
12:    Set v = head(S)                        ▷ v is the current head of S.
13:    for each (v → w) ∈ E do                ▷ Look in the adjacency list of v
14:        if w ∉ V_v then                    ▷ w not yet been visited; recurse on it
15:            scomp_step(w)
16:            lowlink(v) = min(lowlink(v), lowlink(w))
17:        else if w ∈ S then                 ▷ w is in the stack and hence in current SCC
18:            lowlink(v) = min(lowlink(v), invorder(w))
19:        end if
20:    end for
21:    if lowlink(v) = invorder(v) then
22:        pop all vertices down to v from S to obtain a new SCC
23:    end if
24: end recursive function
```

3.5.1 Block Structure Based on Supervariables

Sets of columns of A frequently have identical sparsity patterns. For instance, when A arises from a finite element discretization, the columns corresponding to variables that belong to the same set of finite elements have the same pattern, and this occurs as a result of each node of the finite element mesh having multiple degrees of freedom associated with it. This repetition of the sparsity patterns can be used to substantially enhance performance.

Adjacent vertices u and v in an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ are said to be **indistinguishable** if they have the same neighbours, that is, $adj_{\mathcal{G}}\{u\} \cup \{u\} =$

$adj_{\mathcal{G}}\{v\} \cup \{v\}$. A set of mutually indistinguishable vertices is called an **indistinguishable vertex set**. If $\mathcal{U} \subseteq \mathcal{V}$ is an indistinguishable vertex set, then $\mathcal{U}$ is **maximal** if $\mathcal{U} \cup \{w\}$ is not indistinguishable for any $w \in \mathcal{V} \setminus \mathcal{U}$.

Indistinguishability is an equivalence relation on $\mathcal{V}$, and maximal indistinguishable vertex sets represent its classes. This implies a partitioning of $\mathcal{V}$ into $nsup \geq 1$ non-empty disjoint subsets

$$\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \cup \ldots \cup \mathcal{V}_{nsup}. \tag{3.6}$$

An indistinguishable vertex set can be represented by a single vertex, called a **supervariable**.

If the vertices belonging to each subset $\mathcal{V}_1, \ldots, \mathcal{V}_{nsup}$ are numbered consecutively, with those in $\mathcal{V}_i$ preceding those in $\mathcal{V}_{i+1}$ ($1 \leq i < nsup$), and if P is the permutation matrix corresponding to this ordering, then the permuted matrix PAP^T has a block structure in which the blocks are dense (with the possible exception of the diagonal entries, which can be zero); the dimensions of the blocks are equal to the sizes of the indistinguishable sets.

One approach for identifying supervariables is outlined in Algorithm 3.7. Initially, all the vertices are placed in a single vertex set (that is, into a single supervariable). This is split into two supervariables by taking the first vertex $j = 1$ and moving vertices in the adjacency set of j into a new vertex set (a new supervariable). Each vertex j is considered in turn, and each vertex set $\mathcal{V}_{sv}$ that contains a vertex in $adj_{\mathcal{G}}\{j\} \cup j$ is split into two by moving the vertices in $adj_{\mathcal{G}}\{j\} \cup j$ that belong to $\mathcal{V}_{sv}$ into a new vertex set. Note that as a result of the splitting and moving of vertices, a vertex set can become empty, in which case it is discarded. Once the supervariables have been determined, the permuted matrix PAP^T can be condensed to a matrix of order equal to $nsup$; the corresponding graph is called the **supervariable** graph. If the average number of variables in each supervariable is k, using the supervariable graph will reduce the amount of integer data that is read during the symbolic phase by a factor of about k^2.

As an illustration, consider the following 5×5 matrix

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} * & * & & & * \\ * & * & & & * \\ & & * & * & * \\ & & * & * & * \\ * & * & * & * & * \end{pmatrix} \end{array}.$$

Initially, 1, 2, 3, 4, 5 are put into a single vertex set $\mathcal{V}_1$. Consider $j = 1$. Vertices $i = 1, 2$ and 5 belong to $adj_{\mathcal{G}}\{1\} \cup \{1\}$; they are moved from $\mathcal{V}_1$ into a new vertex set. There is no further splitting of the vertex sets for $j = 2$. For $j = 3$, $adj_{\mathcal{G}}\{3\} \cup \{3\} = \{3, 4, 5\}$. Vertices $i = 3$ and 4 are moved from $\mathcal{V}_1$ into a new vertex set. $\mathcal{V}_1$ is now empty and can be discarded. Vertex $i = 5$ is moved from the vertex set that holds vertices 1 and 2 into a new vertex set. For $j = 4$ and 5, no additional splitting is performed. Thus, three supervariables are found, namely $\{1, 2\}$, $\{3, 4\}$, and $\{5\}$.

ALGORITHM 3.7 Find the supervariables of an undirected graph
Input: Graph $\mathcal{G}$ of a symmetrically structured matrix.
Output: Partitioning of $\mathcal{V}$ into indistinguishable vertex sets.

$\mathcal{V}_1 = \{1, 2, \ldots, n\}$
for $j = 1 : n$ **do**
  for $i \in adj_{\mathcal{G}}\{j\} \cup j$ **do**
    Find sv such that $i \in \mathcal{V}_{sv}$
    if this is the first occurrence of sv for the current index j **then**
      Establish a new vertex set $\mathcal{V}_{nsv}$ and move i from $\mathcal{V}_{sv}$ to $\mathcal{V}_{nsv}$
    else
      Move i from $\mathcal{V}_{sv}$ to $\mathcal{V}_{nsv}$
    end if
    Discard $\mathcal{V}_{sv}$ if it is empty
  end for
end for

3.5.2 *Block Structure Using Symbolic Dot Products*

An alternative way to find a block structure uses symbolic dot products between the rows of the matrix. While fully dense blocks can be found this way, it can also be used to determine an approximate block structure in which blocks are classified as dense or sparse based on a chosen threshold; this can be useful in preconditioning iterative methods. Although we assume that $\mathcal{S}\{A\}$ is symmetric, modifications can extend the approach to general nonsymmetric A.

Rewrite A as row vectors

$$A = \left(a_1^T, \ldots, a_n^T\right)^T, \text{ where } a_i^T = A_{i,1:n},$$

and consider $\mathcal{G}(A) = (\mathcal{V}, \mathcal{E})$. A partition $\mathcal{V} = \mathcal{V}_1 \cup \ldots \cup \mathcal{V}_{nb}$ is constructed using row products $a_i^T a_k$ between different rows of A. These express the level of orthogonality between the rows; if $a_i^T a_k$ is small, then i and k are assigned to different vertex sets. Algorithm 3.8 treats all entries of A as unity, and the symbolic row products can be considered as a generalization of the angles between rows expressed by their cosines, hence the notation *cosine* for the vector that stores these products. The vertex sets are described using the vector *adjmap*. On output, if $adjmap(i_1) = adjmap(i_2)$, then vertices i_1 and i_2 belong to the same vertex set. Symmetry of $\mathcal{S}\{A\}$ simplifies the computation of the symbolic row products because for row i only $k > i$ is considered, that is, only the symbolic row products that correspond to one triangle of $A^T A$ are checked.

The procedure outlined in Algorithm 3.8 and illustrated in Figure 3.7 is controlled by a threshold parameter $\tau \in (0, 1]$. j is added to the subset to which i

ALGORITHM 3.8 Find approximately indistinguishable vertex sets in an undirected graph

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of a symmetrically structured matrix A, the number nz_i of entries in row i of A ($1 \le i \le n$), and a threshold parameter $\tau \in (0, 1]$.

Output: Partitioning of $\mathcal{V}$ into nb disjoint approximately indistinguishable vertex sets.

```
1: nb = 0, adjmap(1 : n) = 0, cosine(1 : n) = 0
2: for i = 1 : n do
3:     if adjmap(i) = 0 then
4:         nb = nb + 1                                   ▷ Start a new set
5:         adjmap(i) = ib
6:         for (i, j) ∈ E do                  ▷ Corresponds to an entry in A_{i,1:n}
7:             for (k, j) ∈ E with k > i do   ▷ Both rows i and k have an entry in
                                                 column j
8:                 if adjmap(k) = 0 then      ▷ k has not been yet added to some
                                                 partitioning set
9:                     cosine(k) = cosine(k) + 1   ▷ Increase partial dot product
10:                end if
11:            end for
12:            for k with cosine(k) ≠ 0 do
13:                if cosine(k)^2 ≥ τ^2 * nz_i * nz_k then   ▷ Test similarity of row
                                                               patterns
14:                    adjmap(k) = nb
15:                end if
16:                cosine(k) = 0
17:            end for
18:        end for
19:    end if
20: end for
```

belongs if the cosine of the angle between them exceeds τ. If $\tau < 1$, the block structure depends on the order in which the rows are processed, while $\tau = 1$ gives the exact indistinguishable vertex sets because, in this case, the row patterns being compared must be the identical for the rows to be assigned to the same set.

3.6 Notes and References

A standard description of LU factorizations based on the generic scheme given in Algorithm 3.1 can be found in the classical book by Ortega (1988b); this includes the

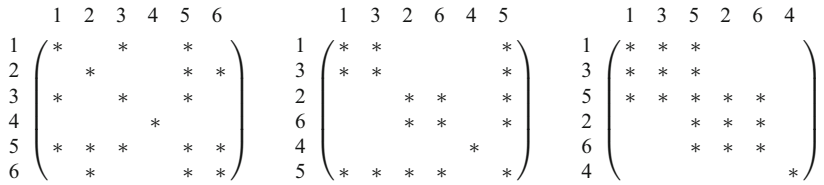

Figure 3.7 An example to illustrate Algorithm 3.8. The original matrix is given (left) together with the permuted matrix with indistinguishable vertex sets $\mathcal{V} = \{1, 3\} \cup \{2, 6\} \cup \{4\} \cup \{5\}$ obtained using $\tau = 1$ (centre) and the permuted matrix with approximately indistinguishable vertex sets $\mathcal{V} = \{1, 3, 5\} \cup \{2, 6\} \cup \{4\}$ obtained using $\tau = 0.5$ (right). The threshold $\tau = 0.5$ results in putting row 5 into the same set as row 1, making the vertex sets only approximately indistinguishable. The permuted matrix on the right has an approximate block form.

symmetric case and discusses early parallelization issues (which are also considered in the review of Dongarra et al. (1984)). A more algorithmically oriented approach is given in Golub & Van Loan (1996). For the column variant with partial pivoting, we recommend the detailed description of the sparse case in Gilbert & Peierls (1988). Many results for sparse LU factorizations are surveyed by Gilbert & Ng (1993) and Gilbert (1994). Pothen & Toledo (2004) consider both symmetric and nonsymmetric matrices in their survey of graph models of sparse elimination. The review by Davis et al. (2016) provides many further references.

Parter (1961) presents Parter's rule, and its nonsymmetric version is given in Haskins & Rose (1973). Building on the paper of Rose et al. (1976), Rose & Tarjan (1978) were the first to methodically consider the symbolic structure of Gaussian elimination for nonsymmetric matrices. Related work is included in the seminal paper on Cholesky factorizations by Liu (1986). Fill-in rules in the general context of Schur complements in LU factorizations can be found in Eisenstat & Liu (1993b).

Classical and detailed treatments of triangular solves that also cover sparse issues are given in the papers Brayton et al. (1970), Gilbert & Peierls (1988), and Gilbert (1994). For reducibility theory that is closely connected to the general theory of matrices, see Brualdi & Ryser (1991), which includes, for example, a proof of Theorem 3.4.

Algorithm 3.6 for computing strongly connected components of a digraph is introduced in Tarjan (1972); see also Sharir (1981) and Duff & Reid (1978) for an early implementation.

For identifying supervariables, Algorithm 3.7 follows Reid & Scott (1999), but see also Ashcraft (1995) and Hogg & Scott (2013a) (the latter presents an efficient variant that employs a stack). The approximate block partitioning of Section 3.5.2 is from the paper by Saad (2003a), which also describes some modifications of the basic approach; more sophisticated schemes with overlapping blocks are given in Fritzsche et al. (2013).