CHAPTER 3

Special linear systems

As a general principle, we want to take advantage of any special structure that may be present in the problem under consideration, to increase the efficiency of our computations. This increase of efficiency might be translated into faster computations using less memory space, and/or more reliable results. Next we will study matrices exhibiting properties like *symmetry*, *definiteness* and *bandedness*.

3.1. Symmetric matrices

An $n \times n$ matrix A is *symmetric* if

$$A^T = A.$$

Since $a_{j,i} = a_{i,j}$ for all i, j, to represent A we only need to specify the $\frac{n\,(n+1)}{2}$ entries $a_{i,j}$ for $i \geq j$, instead of the usual n^2 ones, and so to store a symmetric matrix we need approximately half the space that for general one. Consequently, in this case we

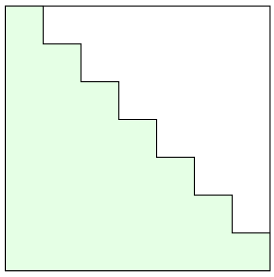

FIGURE 3.1.1. Storage of a symmetric matrix

would like to solve the linear equation

$$A\,x = b$$

with half the complexity of the general case, that is, using (approximately) $\frac{n^2}{2}$ memory slots instead of n^2, and $\frac{n^3}{3}$ flops instead of $\frac{2\,n^3}{3}$.

Permuting rows destroys symmetry, as can be seen already when $n = 2$:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} b & c \\ a & b \end{bmatrix}.$$

Hence to preserve symmetry, we will avoid using any pivoting strategy, and consequently we will only consider its LU factorization, whenever it exists.

From now on, suppose that A is a symmetric nonsingular $n \times n$ matrix that has a factorization

$$A = L\,U \tag{3.1}$$

with L unit lower triangular and U upper triangular. When this occurs, these factors are connected: for instance, when $n = 2$ we have that

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{b}{a} & 1 \end{bmatrix} \begin{bmatrix} a & b \\ 0 & c - \frac{b^2}{a} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{b}{a} & 1 \end{bmatrix} \begin{bmatrix} a & 0 \\ 0 & c - \frac{b^2}{a} \end{bmatrix} \begin{bmatrix} 1 & \frac{b}{a} \\ 0 & 1 \end{bmatrix},$$

and so U is a row scaling of L. This is a general fact: in the factorization (3.1) we have that

$$U = D\, L^T$$

with $D = \operatorname{diag}(u_{1,1}, \dots, u_{n,n})$, and so this factorization is equivalent to

$$A = L\, D\, L^T$$

with L unit lower triangular and D diagonal.

This LDLT factorization also allows to solve the linear equation $A\,x = b$ following the steps:

(1) solve $L\,z = b$ by forward substitution,
(2) solve $D\,y = z$ scaling each entry,
(3) solve $L^T\,x = y$ by backward substitution.

To compute it, we proceed similarly as we did before for the PLU factorization: consider the 2×2 block decomposition

$$A = \begin{array}{cc} \begin{array}{cc} \scriptstyle 1 & \scriptstyle n-1 \end{array} & \\ \begin{bmatrix} a_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} & \begin{array}{l} \scriptstyle 1 \\ \scriptstyle n-1 \end{array} \end{array}$$

where $a_{1,1}$ is the $(1,1)$-entry of A, $A_{1,2}$ and $A_{2,1}$ are the rest of its first row and columns respectively, and $A_{2,2}$ is the remaining $(n-1) \times (n-1)$ block. Since we assume that A admits an LU factorization we have that $a_{1,1} \neq 0$, and since it is symmetric we also have that $A_{1,2} = A_{2,1}^T$. Then we can compute the the block LDLT factorization

$$\begin{bmatrix} a_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} = \begin{bmatrix} 1 & \mathbb{0} \\ L_{2,1} & \mathbb{1}_{n-1} \end{bmatrix} \begin{bmatrix} d_1 & \mathbb{0} \\ \mathbb{0} & A_1 \end{bmatrix} \begin{bmatrix} 1 & L_{2,1}^T \\ \mathbb{0} & \mathbb{1}_{n-1} \end{bmatrix}$$

by setting

$$d_1 = a_{1,1}, \quad L_{2,1} = a_{1,1}^{-1}\, A_{2,1}, \quad D_1 = A_{2,2} - A_{2,1}\, L_{2,1}^T. \tag{3.2}$$

The Schur complement A_1 is a symmetric nonsingular $(n-1) \times (n-1)$ matrix and so we can repeat the procedure on it, leading to a symmetric version of Algorithm 1.3.1.

For convenience, we write the Schur complements defined by the successive application of the formulae in (3.2) as

$$A_j = \big[a_{i,k}^{(j)}\big]_{j+1 \leq i,k \leq n} \qquad \text{for } j = 0, \dots, n-1.$$

As it stands, this algorithm is inefficient both from the point of view of memory usage and speed of execution. To fully profit from the symmetry of the input matrix A, first recall that it can be represented keeping only its (i,j)-th entries for $i \geq j$. At the j-th step of Algorithm 3.1.1, the (j,j)-entry of the $(j-1)$-th Schur complement A_{j-1} is assigned to the j-th diagonal entry of D and never used again. On the other hand, its (i,j)-th entry is used both to compute the (i,j)-th entry of L and the i-th row of the j-th Schur complement. Moreover, the (i,k)-entries of A_{j-1} are only used to compute this next Schur complement. Moreover, since it is symmetric, we only need to compute its lower triangular part.

Algorithm 3.1.1 (LDLT algorithm)

```
for $j = 1, \dots, n$
    $d_j \leftarrow a_{j,j}^{(j-1)}$        (compute the $j$-th diagonal entry of $D$)
    for $i = j+1, \dots, n$
        $l_{i,j} \leftarrow a_{i,j}^{(j-1)} / a_{j,j}^{(j-1)}$        (compute the $j$-th column of $L$)
    for $i, k = j+1, \dots, n$
    $a_{i,k}^{(j)} \leftarrow a_{i,j}^{(j-1)} - a_{i,j}^{(j-1)} \, l_{k,j}$        (compute the $j$-th Schur complement)
$d_n \leftarrow a_{n,n}^{(n-1)}$        (compute the $n$-th diagonal entry of $D$)
```

Hence we can safely rewrite the lower triangular part of A with the j-th diagonal entry of D, the nontrivial entries in the j-th column of L, and the lower triangular part of the j-th Schur complement. Hence we need no extra space to store them apart from an auxiliary $(n-1)$-th vector c to keep the values of the j-th column of A. At the end of the procedure, the lower triangular part of A would contain all the diagonal entries of D and the nontrivial entries of L.

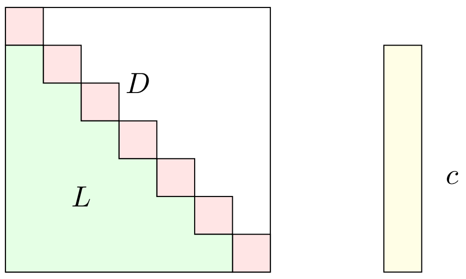

FIGURE 3.1.2. Storage distribution for LDLT

Algorithm 3.1.2 (LDLT algorithm with storage management)

```
for $j = 1, \dots, n$
    for $i = j+1, \dots, n$
        $c_i \leftarrow a_{i,j}$
        $a_{i,j} \leftarrow a_{i,j} / a_{i,j}$
    for $k = j+1, \dots, n$ and $i = k, \dots, n$
    $a_{i,k} \leftarrow a_{i,j} - c_i \, a_{k,j}$
```

As shown in Figure 3.1.2, this algorithm uses an (essentially) optimal amount of space, namely

$$\frac{n\,(n+1)}{2} + n - 1 = \frac{n^2}{2} \text{ memory slots.}$$

On the other hand, its time complexity is

$$\sum_{j=1}^{n-1}\Bigg(\sum_{i=j+1}^{n} 1 + \sum_{j+1 \le k \le i \le n} 2\Bigg) = \sum_{j=1}^{n-1}((n-j) + (n-j)\,(n-j+1))$$
$$= \sum_{j=1}^{n-1}((n-j)^2 + O(n-j)) = \frac{n^3}{3} + O(n^2),$$

which is asymptotically half the time complexity of the PLU factorization.

EXAMPLE 3.1.1. Consider the 3×3 matrix

$$A = \begin{bmatrix} 1 & -1 & 2 \\ -1 & 5 & 2 \\ 2 & 2 & 17 \end{bmatrix}.$$

Applying Algorithm 3.1.2, for $j = 1$ we have that

$$c = \begin{bmatrix} -1 \\ 2 \end{bmatrix} \quad \text{and} \quad A = \begin{bmatrix} 1 & & \\ -1/1 & 5-(-1)\cdot(-1) & \\ 2/1 & 2-2\cdot(-1) & 17-2\cdot 2 \end{bmatrix} = \begin{bmatrix} 1 & & \\ -1 & 4 & \\ 2 & 4 & 13 \end{bmatrix}.$$

For $j = 2$ we have that

$$c = \begin{bmatrix} * \\ 4 \end{bmatrix} \quad \text{and} \quad A = \begin{bmatrix} 1 & & \\ -1 & 4 & \\ 2 & 4/4 & 13-4\cdot 1 \end{bmatrix} = \begin{bmatrix} 1 & & \\ -1 & 4 & \\ 2 & 1 & 9 \end{bmatrix},$$

from where we extract the matrices in the LDLT factorization:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & 1 & 1 \end{bmatrix} \quad \text{and} \quad D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 9 \end{bmatrix}.$$

In general, the LDLT of a symmetric matrix can be numerically unstable, as shown already by the matrix

$$A = \begin{bmatrix} \eta & 1 \\ 1 & 1 \end{bmatrix}$$

for $0 < \eta$ smaller than the machine epsilon (Example 2.5.1). Still this factorization can be useful in practice, whenever the successive pivots are not too small.

3.2. Symmetric positive definite systems

Let A be a symmetric $n \times n$ matrix with real coefficients. We say that A is a *positive-definite* if for all $x \in \mathbb{R}^n \setminus \{0\}$ we have that

$$x^T A\, x > 0.$$

Positive definite symmetric (PDS) matrices play an important role in convex optimization. For instance, given a function of several real variables that is twice differentiable, if its Hessian matrix (the matrix of its second partial derivatives) is positive-definite then the function is convex in a neighborhood of that point.

PDS matrices can be characterized in several ways. First of all, there is an important result from linear algebra saying that a real $n \times n$ matrix A is symmetric if and only if it is orthogonally similar to a diagonal matrix, that is

$$A = Q^T \Lambda\, Q \tag{3.3}$$

with Q orthogonal and Λ diagonal, and then A is positive-definite if and only if the diagonal entries of Λ are positive.

In the factorization (3.3), we have that $Q^{-1} = Q^T$ and that the diagonal entries of Λ coincide with the eigenvalues of the given symmetric matrix. Hence PDS matrices can also be characterized as the symmetric real matrices whose eigenvalues are positive.

A third characterization of PDS matrices is given by the existence of a *Cholesky factorization*, a variant of the LU factorization with a positivity property. Namely, a real $n \times n$ matrix A is PDS if and only if there is a lower triangular $n \times n$ matrix G with positive diagonal entries such that

$$A = G\, G^T. \tag{3.4}$$

When it exists, this lower triangular matrix is unique [**Dem97**, Proposition 2.2].

This factorization is closely related to the LDLT factorization of a symmetric matrix studied in §3.1. Indeed, consider the diagonal matrix $\Lambda = \mathrm{diag}(g_{1,1}, \dots, g_{n,n})$ and set

$$L = G\, \Lambda^{-1} \quad \text{and} \quad D = \Lambda^2.$$

Then L is unit lower triangular and D is diagonal, and the Cholesky factorization (3.4) translates into the LDLT factorization

$$A = L\, D\, L^T.$$

Indeed, one way of computing the Cholesky factorization a PDS matrix proceeds by computing its LDLT factorization with Algorithm 1.3.2. The diagonal entries of D are positive and we might then obtain the lower triangular matrix G in (3.4) by setting

$$G = L\, \mathrm{diag}(d_{1,1}^{1/2}, \dots, d_{n,n}^{1/2}).$$

As a matter of fact, it will be easier to proceed in a different way. Given the PDS $n \times n$ matrix A, the matrix G is the solution of a system of $\frac{n\,(n+1)}{2}$ equations (one per each entry of A in its lower triangular part) in $\frac{n\,(n+1)}{2}$ variables (the nontrivial entries of G).

These equations are nonlinear but nevertheless, they can be easily solved when appropriately ordered: for each $i \geq j$ we have that $a_{i,j} = \sum_{k=0}^{j} g_{j,k}\, g_{i,k}$, or equivalently

$$g_{j,j}\, g_{i,j} = a_{i,j} - \sum_{k=1}^{j-1} g_{j,k}\, g_{i,k}.$$

If the first $j-1$ columns of G are already known, we can use this equation to compute the diagonal entry $g_{j,j}$ and then the rest of the entries in this column. Similarly as with GEPP (and to a great extent with the LDLT factorization) each (i,j)-th entry of A is used only to compute the corresponding entry of G, and so we can safely overwrite it. At the end of the procedure, the matrix A would contain all the nontrivial entries of G in its lower triangular part.

Algorithm 3.2.1 (Cholesky algorithm))

```
for $j = 1, \dots, n$
    $a_{j,j} \leftarrow (a_{j,j} - \sum_{k=1}^{j-1} a_{j,k}^2)^{1/2}$
    for $i = j+1, \dots, n$
        $a_{i,j} \leftarrow (a_{i,j} - \sum_{k=1}^{j-1} a_{i,k}\, a_{j,k})/a_{j,j}$
```

Thanks to the characterization of PDS matrices in (3.4), this algorithm might be applied to an arbitrary real symmetric A: if this matrix is PDS then we would obtain its Cholesky factorization as explained and if it is not, then the computation algorithm would break down because of a forbidden operation, like taking the square root of a negative number or dividing by zero. Indeed, this is the cheapest way to test if a given real symmetric matrix is definite-positive or not.

EXAMPLE 3.2.1. Consider again the 3×3 matrix

$$A = \begin{bmatrix} 1 & -1 & 2 \\ -1 & 5 & 2 \\ 2 & 2 & 17 \end{bmatrix}$$

as in Example 3.1.1. Applying the Cholesky algorithm, for $j = 1$ we obtain

$$A = \begin{bmatrix} 1^{1/2} & & \\ -1/1 & 5 & \\ 2/1 & 2 & 17 \end{bmatrix} = \begin{bmatrix} 1 & & \\ -1 & 5 & \\ 2 & 2 & 17 \end{bmatrix}.$$

For $j = 2$ we have that

$$A = \begin{bmatrix} 1 & & \\ -1 & (5 - (-1) \cdot (-1))^{1/2} & \\ 2 & (2 - (-1) \cdot 2)/2 & 17 \end{bmatrix} = \begin{bmatrix} 1 & & \\ -1 & 2 & \\ 2 & 2 & 17 \end{bmatrix}$$

Finally, for $j = 3$ we obtain

$$A = \begin{bmatrix} 1 & & \\ -1 & 2 & \\ 2 & 2 & (17 - (2 \cdot 2 + 2 \cdot 2))^{1/2} \end{bmatrix} = \begin{bmatrix} 1 & & \\ -1 & 2 & \\ 2 & 2 & 3 \end{bmatrix}.$$

Hence A is positive-definite, and the matrix in its Cholesky factorization is

$$G = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 2 & 0 \\ 2 & 2 & 3 \end{bmatrix}.$$

Algorithm 3.2.1 uses an optimal amount of space, namely

$$\frac{n\,(n+1)}{2} \text{ memory slots,}$$

exactly the space needed to represent A.

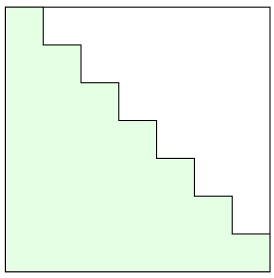

FIGURE 3.2.1. Storage distribution for the Cholesky algorithm

Its time complexity is

$$\begin{aligned}\sum_{j=1}^{n}\Big(2\,j-1+\sum_{i=j+1}^{n}(2\,j-1)\Big) &=\sum_{j=1}^{n}(n-j+1)\,(2\,j-1)\\ &=\sum_{j=1}^{n}2\,n\,j-\sum_{j=1}^{n}2\,j^2+O(n^2)\\ &=(n^3+O(n^2))-\Big(\frac{2}{3}\,n^3+O(n^2)\Big)+O(n^2)\\ &=\frac{1}{3}\,n^3+O(n^2),\end{aligned}$$

which is also asymptotically half the time complexity of the PLU factorization.

Furthermore, the Cholesky algorithm is also backward stable. Indeed, let A be a PDS matrix and b a real n-vector, and let x_{Ch} be the computed solution of the equation $A\,x=b$ using the Cholesky factorization, combined with forward and backward substitution for solving the equations $G\,y=b$ and $G^T x=y$ respectively. The same analysis of GEPP in § 2.6 shows that this solution satisfies

$$(A+\delta A)\,x_{\mathrm{Ch}}=b$$

with

$$|\delta A|\le 3\,n\,\varepsilon\,|G|\,|G^T| \tag{3.5}$$

with ε the machine epsilon and where the bars indicate the matrices whose entries are the absolute values of those in the indicated ones, and where the inequality occurs at every entry.

By the Cauchy-Schwartz inequality, for each i,j we have that

$$(|G|\,|G^T|)_{i,j}\le\sum_{k=1}^{n}|g_{i,k}|\,|g_{j,k}|\le\Big(\sum_{k=1}^{n}g_{i,k}^2\Big)^{1/2}\Big(\sum_{k=1}^{n}g_{j,k}^2\Big)^{1/2}=a_{i,i}^{1/2}a_{j,j}^{1/2}\le\max_{i,j}|a_{i,j}|.$$

Hence $\||G|\,|G^T|\|_\infty\le n\,\|A\|_\infty$, and so we deduce from (3.5) the upper bound for the relative backward error

$$\frac{\|\delta\,A\|_\infty}{\|A\|_\infty}\le 3\,n^2\,\varepsilon.$$

Properly done, all the elements in this section can be extended to the complex case. An $n\times n$ matrix A with complex entries is *Hermitian* if it coincides with its conjugate transpose that is, if

$$A^*=A.$$

A Hermitian matrix is *positive-definite* if for all $x\in\mathbb{C}^n\setminus\{0\}$ we have that

$$x^*A\,x>0.$$

Hermitian matrices enjoy similar properties as those of symmetric real matrices: for instance, a complex $n\times n$ matrix A is Hermitian if and only if

$$A=Q^*\Lambda\,Q$$

with Q a unitary matrix and Λ a diagonal matrix with real entries. When this is the case, the Hermitian matrix A is positive-definite if and only if the diagonal entries of Λ are positive.

The Cholesky factorization also extends to this setting: a complex $n \times n$ matrix A is positive-definite Hermitian if and only if it can be factored as

$$A = G\,G^*$$

with G lower triangular con positive diagonal entries, and Algorithm 3.2.1 can be easily modified to compute this matrix.

3.3. Band matrices

Roughly speaking, a *band matrix* is a matrix whose entries are concentrated around the diagonal. Such matrices arise from systems of (scalar) linear equations that can be ordered in such a way that each i-th variable only appears in a neighborhood of the i-th equation, like those appearing when discretizing ordinary differential equations (ODE's).

More precisely, the *lower bandwidth* and the *upper bandwidth* of an $n \times n$ matrix A are respectively defined as the smallest integers b_L and b_U such that

$$a_{i,j} = 0$$

whenever $i > j - b_L$ or $i < j + b_U$. For instance, a nonsingular lower (respectively upper) triangular matrix has lower (respectively upper) bandwidth equal to zero, a tridiagonal matrix with nonzero subdiagonal and supdiagonal entries has lower and upper bandwidths equal to one, and so on.

A nonsingular matrix A with lower bandwidth b_L and upper band with b_U has the shape

$$\begin{bmatrix} a_{1,1} & \cdots & a_{1,b_U+1} & & \\ \vdots & & & \ddots & \\ a_{b_L+1,1} & & & & a_{n-b_U,n} \\ & \ddots & & & \vdots \\ & & a_{n,n-b_L} & \cdots & a_{n,n} \end{bmatrix}$$

and so we need at most $(b_L + b_U)\,n$ entries to represent it.

Its LU factorization (whenever it exists!) preserves its band structure: we have that

$$A = L\,U$$

with L unit lower triangular with lower bandwidth b_L and U upper triangular with upper bandwidth b_U.

Indeed, this factorization can be computed iteratively using Algorithm 1.3.2 skipping the pivoting strategy. At the j-th step, the j-th Schur complement is computed with the formula

$$a_{i,k} \leftarrow a_{i,k} - a_{i,j}\,a_{j,k} \qquad \text{for } i,k = j+1,\dots,n.$$

Hence the lower and upper bandwidths of this Schur complement are bounded by those of the previous one, and so by those of the input matrix A.

Thus this factorization can be computed by overwriting the nontrivial entries of A and so with at most $(b_L + b_U)\,n$ memory slots, and this can be done using at most

$$2\,n\,b_L\,b_U + O(n\,(b_L + b_U)) \text{ flops.}$$

EXAMPLE 3.3.1. Consider the 4×4 matrix

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

which has lower and upper bandwidths equal to 1. Applying Algorithm 1.3.2 without pivoting, for $j = 1$ we obtain

$$A = \begin{bmatrix} 2 & -1 & & \\ 4/2 & -1 - 2 \cdot (-1) & 3 & \\ & -1 & -2 & 1 \\ & & 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & -1 & & \\ 2 & 1 & 3 & \\ & -1 & -2 & 1 \\ & & 3 & 4 \end{bmatrix}.$$

For $j = 2$ we have that

$$A = \begin{bmatrix} 2 & -1 & & \\ 2 & 1 & 3 & \\ & -1/1 & -2 - (-1) \cdot 3 & 1 \\ & & 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & -1 & & \\ 2 & 1 & 3 & \\ & -1 & 1 & 1 \\ & & 3 & 4 \end{bmatrix}.$$

Finally, for $j = 3$ we obtain that

$$A = \begin{bmatrix} 2 & -1 & & \\ 2 & 1 & 3 & \\ & -1 & 1 & 1 \\ & & 3/1 & 4 - 3 \cdot 1 \end{bmatrix} = \begin{bmatrix} 2 & -1 & & \\ 2 & 1 & 3 & \\ & -1 & 1 & 1 \\ & & 3 & 1 \end{bmatrix}.$$

We conclude that

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 3 & 1 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} 2 & -1 & 0 & 0 \\ 0 & 1 & 3 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

GEPP can exploit band structure, but the band properties of the factors L and U are less simple. Indeed, GEPP produces a factorization

$$A = P\,L\,U$$

where U is banded with upper bandwidth $b_L + b_U$, and L has at most $b_L + 1$ nonzero entries per column.

Indeed, at the j-th step of Algorithm 1.3.2, pivoting can only be done within the first b_L rows, because the others have only zeros on the j-th column. Hence the corresponding permutation can increase the upper bandwidth by b_L, and it can also reorder the entries of the previously computed columns of L.

EXAMPLE 3.3.2. Consider again the matrix in Example 3.3.1

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

Applying GEPP (Algorithm 1.3.2), for $j = 1$ we first swap rows 1 and 2 and then we compute the first column of L, the first row of U, and the first Schur complement:

$$A = \begin{bmatrix} 4 & -1 & 3 & 0 \\ 2/4 & -1 - \frac{1}{2} \cdot (-1) & 0 - \frac{1}{2} \cdot 3 & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 4 & -1 & 3 & 0 \\ \frac{1}{2} & \frac{-1}{2} & \frac{-3}{2} & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 3 & 4 \end{bmatrix}.$$

For $j = 2$ we swap the rows 2 and 3 of A and then compute the second column of L, row of U, and Schur complement:

$$A = \begin{bmatrix} 4 & -1 & 3 & 0 \\ 0 & -1 & -2 & 1 \\ \frac{1}{2} & \frac{-1}{2}/(-1) & \frac{-3}{2} - \frac{1}{2} \cdot (-2) & 0 - \frac{1}{2} \cdot 1 \\ 0 & 0 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 4 & -1 & 3 & 0 \\ 0 & -1 & -2 & 1 \\ \frac{1}{2} & \frac{1}{2} & \frac{-1}{2} & \frac{-1}{2} \\ 0 & 0 & 3 & 4 \end{bmatrix}.$$

Finally, for $j = 3$ we swap rows 3 and 4 and we compute the third column of L, row of U and Schur complement to obtain

$$A = \begin{bmatrix} 4 & -1 & 3 & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 3 & 4 \\ \frac{1}{2} & \frac{1}{2} & \frac{-1}{2}/3 & \frac{-1}{2} - \frac{-1}{6} \cdot 4 \end{bmatrix} = \begin{bmatrix} 4 & -1 & 3 & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 3 & 4 \\ \frac{1}{2} & \frac{1}{2} & \frac{-1}{6} & \frac{1}{6} \end{bmatrix}.$$

We conclude that

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & & & \\ 0 & 1 & & \\ 0 & 0 & 1 & \\ \frac{1}{2} & \frac{1}{2} & \frac{-1}{6} & 1 \end{bmatrix}, \quad U \begin{bmatrix} 4 & -1 & 3 & 0 \\ & -1 & -2 & 1 \\ & & 3 & 4 \\ & & & \frac{1}{6} \end{bmatrix}.$$

In this case, U has upper bandwidth 2 and L has at most 2 nonzero elements per column.