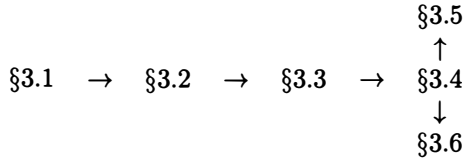
Chapter 3

General Linear Systems

The problem of solving a linear system $Ax = b$ is central to scientific computation. In this chapter we focus on the method of Gaussian elimination, the algorithm of choice if A is square, dense, and unstructured. Other methods are applicable if A does not fall into this category, see Chapter 4, Chapter 11, §12.1, and §12.2. Solution procedures for triangular systems are discussed first. These are followed by a derivation of Gaussian elimination that makes use of Gauss transformations. The process of eliminating unknowns from equations is described in terms of the factorization $A = LU$ where L is lower triangular and U is upper triangular. Unfortunately, the derived method behaves poorly on a nontrivial class of problems. An error analysis pinpoints the difficulty and sets the stage for a discussion of pivoting, a permutation strategy that keeps the numbers "nice" during the elimination. Practical issues associated with scaling, iterative improvement, and condition estimation are covered. A framework for computing the LU factorization in parallel is developed in the final section.

Reading Notes

Familiarity with Chapter 1, §§2.1–2.5, and §2.7 is assumed. The sections within this chapter depend upon each other as follows:



Useful global references include Forsythe and Moler (SLAS), Stewart(MABD), Higham (ASNA), Watkins (FMC), Trefethen and Bau (NLA), Demmel (ANLA), and Ipsen (NMA).

3.1 Triangular Systems

Traditional factorization methods for linear systems involve the conversion of the given square system to a triangular system that has the same solution. This section is about the solution of triangular systems.

3.1.1 Forward Substitution

Consider the following 2-by-2 lower triangular system:

$$\begin{bmatrix} \ell_{11} & 0 \\ \ell_{21} & \ell_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}.$$

If $\ell_{11}\ell_{22} \neq 0$, then the unknowns can be determined sequentially:

$$x_1 = b_1/\ell_{11},$$
$$x_2 = (b_2 - \ell_{21}x_1)/\ell_{22}.$$

This is the 2-by-2 version of an algorithm known as *forward substitution*. The general procedure is obtained by solving the ith equation in $Lx = b$ for x_i:

$$x_i = \left(b_i - \sum_{j=1}^{i-1} \ell_{ij}x_j \right) \Big/ \ell_{ii}.$$

If this is evaluated for $i = 1{:}n$, then a complete specification of x is obtained. Note that at the ith stage the dot product of $L(i, 1{:}i-1)$ and $x(1{:}i-1)$ is required. Since b_i is involved only in the formula for x_i, the former may be overwritten by the latter.

Algorithm 3.1.1 (Row-Oriented Forward Substitution) If $L \in \mathbb{R}^{n\times n}$ is lower triangular and $b \in \mathbb{R}^n$, then this algorithm overwrites b with the solution to $Lx = b$. L is assumed to be nonsingular.

$b(1) = b(1)/L(1,1)$
for $i = 2{:}n$
  $b(i) = (b(i) - L(i, 1{:}i-1)\cdot b(1{:}i-1))/L(i,i)$
end

This algorithm requires n^2 flops. Note that L is accessed by row. The computed solution $\hat{x}$ can be shown to satisfy

$$(L+F)\hat{x} = b \qquad |F| \leq n\mathbf{u}|L| + O(\mathbf{u}^2). \tag{3.1.1}$$

For a proof, see Higham (ASNA, pp. 141-142). It says that the computed solution exactly satisfies a slightly perturbed system. Moreover, each entry in the perturbing matrix F is small relative to the corresponding element of L.

3.1.2 Back Substitution

The analogous algorithm for an upper triangular system $Ux = b$ is called *back substitution*. The recipe for x_i is prescribed by

$$x_i = \left(b_i - \sum_{j=i+1}^{n} u_{ij} x_j \right) \Big/ u_{ii}$$

and once again b_i can be overwritten by x_i.

Algorithm 3.1.2 (Row-Oriented Back Substitution) If $U \in \mathbb{R}^{n \times n}$ is upper triangular and $b \in \mathbb{R}^n$, then the following algorithm overwrites b with the solution to $Ux = b$. U is assumed to be nonsingular.

$b(n) = b(n)/U(n,n)$
for $i = n-1\colon -1\colon 1$
    $b(i) = (b(i) - U(i, i+1\colon n) \cdot b(i+1\colon n))/U(i,i)$
end

This algorithm requires n^2 flops and accesses U by row. The computed solution $\hat{x}$ obtained by the algorithm can be shown to satisfy

$$(U + F)\hat{x} = b, \qquad |F| \leq n\mathbf{u}|U| + O(\mathbf{u}^2). \tag{3.1.2}$$

3.1.3 Column-Oriented Versions

Column-oriented versions of the above procedures can be obtained by reversing loop orders. To understand what this means from the algebraic point of view, consider forward substitution. Once x_1 is resolved, it can be removed from equations 2 through n leaving us with the reduced system

$$L(2\colon n, 2\colon n)x(2\colon n) = b(2\colon n) - x(1) \cdot L(2\colon n, 1).$$

We next compute x_2 and remove it from equations 3 through n, etc. Thus, if this approach is applied to

$$\begin{bmatrix} 2 & 0 & 0 \\ 1 & 5 & 0 \\ 7 & 9 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 2 \\ 5 \end{bmatrix},$$

we find $x_1 = 3$ and then deal with the 2-by-2 system

$$\begin{bmatrix} 5 & 0 \\ 9 & 8 \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \end{bmatrix} - 3 \begin{bmatrix} 1 \\ 7 \end{bmatrix} = \begin{bmatrix} -1 \\ -16 \end{bmatrix}.$$

Here is the complete procedure with overwriting.

Algorithm 3.1.3 (Column-Oriented Forward Substitution) If the matrix $L \in \mathbb{R}^{n \times n}$ is lower triangular and $b \in \mathbb{R}^n$, then this algorithm overwrites b with the solution to $Lx = b$. L is assumed to be nonsingular.

for $j = 1{:}n-1$
  $b(j) = b(j)/L(j,j)$
  $b(j+1{:}n) = b(j+1{:}n) - b(j)\cdot L(j+1{:}n, j)$
end
$b(n) = b(n)/L(n,n)$

It is also possible to obtain a column-oriented saxpy procedure for back substitution.

Algorithm 3.1.4 (Column-Oriented Back Substitution) If $U \in \mathbb{R}^{n\times n}$ is upper triangular and $b \in \mathbb{R}^n$, then this algorithm overwrites b with the solution to $Ux = b$. U is assumed to be nonsingular.

for $j = n{:}-1{:}2$
  $b(j) = b(j)/U(j,j)$
  $b(1{:}j-1) = b(1{:}j-1) - b(j)\cdot U(1{:}j-1, j)$
end
$b(1) = b(1)/U(1,1)$

Note that the dominant operation in both Algorithms 3.1.3 and 3.1.4 is the saxpy operation. The roundoff behavior of these implementations is essentially the same as for the dot product versions.

3.1.4 Multiple Right-Hand Sides

Consider the problem of computing a solution $X \in \mathbb{R}^{n\times q}$ to $LX = B$ where $L \in \mathbb{R}^{n\times n}$ is lower triangular and $B \in \mathbb{R}^{n\times q}$. This is the *multiple-right-hand-side* problem and it amounts to solving q separate triangular systems, i.e., $LX(:,j) = B(:,j)$, $j = 1{:}q$. Interestingly, the computation can be blocked in such a way that the resulting algorithm is rich in matrix multiplication, assuming that q and n are large enough. This turns out to be important in subsequent sections where various block factorization schemes are discussed.

It is sufficient to consider just the lower triangular case as the derivation of block back substitution is entirely analogous. We start by partitioning the equation $LX = B$ as follows:

$$\begin{bmatrix} L_{11} & 0 & \cdots & 0 \\ L_{21} & L_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ L_{N1} & L_{N2} & \cdots & L_{NN} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{bmatrix}. \tag{3.1.3}$$

Assume that the diagonal blocks are square. Paralleling the development of Algorithm 3.1.3, we solve the system $L_{11}X_1 = B_1$ for X_1 and then remove X_1 from block equations 2 through N:

$$\begin{bmatrix} L_{22} & 0 & \cdots & 0 \\ L_{32} & L_{33} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ L_{N2} & L_{N3} & \cdots & L_{NN} \end{bmatrix} \begin{bmatrix} X_2 \\ X_3 \\ \vdots \\ X_N \end{bmatrix} = \begin{bmatrix} B_2 \\ B_3 \\ \vdots \\ B_N \end{bmatrix} - \begin{bmatrix} L_{21} \\ L_{31} \\ \vdots \\ L_{N1} \end{bmatrix} X_1 .$$

Continuing in this way we obtain the following block forward elimination scheme:

```
for j = 1:N
    Solve L_jj X_j = B_j
    for i = j + 1:N                                   (3.1.4)
        B_i = B_i − L_ij X_j
    end
end
```

Notice that the i-loop oversees a single block saxpy update of the form

$$\begin{bmatrix} B_{j+1} \\ \vdots \\ B_N \end{bmatrix} = \begin{bmatrix} B_{j+1} \\ \vdots \\ B_N \end{bmatrix} - \begin{bmatrix} L_{j+1,j} \\ \vdots \\ L_{N,j} \end{bmatrix} X_j.$$

To realize level-3 performance, the submatrices in (3.1.3) must be sufficiently large in dimension.

3.1.5 The Level-3 Fraction

It is handy to adopt a measure that quantifies the amount of matrix multiplication in a given algorithm. To this end we define the *level-3 fraction* of an algorithm to be the fraction of flops that occur in the context of matrix multiplication. We call such flops *level-3 flops.*

Let us determine the level-3 fraction for (3.1.4) with the simplifying assumption that $n = rN$. (The same conclusions hold with the unequal blocking described above.) Because there are N applications of r-by-r forward elimination (the level-2 portion of the computation) and n^2 flops overall, the level-3 fraction is approximately given by

$$1 - \frac{Nr^2}{n^2} = 1 - \frac{1}{N}.$$

Thus, for large N almost all flops are level-3 flops. It makes sense to choose N as large as possible subject to the constraint that the underlying architecture can achieve a high level of performance when processing block saxpys that have width $r = n/N$ or greater.

3.1.6 Nonsquare Triangular System Solving

The problem of solving nonsquare, m-by-n triangular systems deserves some attention. Consider the lower triangular case when $m \geq n$, i.e.,

$$\begin{bmatrix} L_{11} \\ L_{21} \end{bmatrix} x = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \qquad \begin{matrix} L_{11} \in \mathbb{R}^{n\times n}, & b_1 \in \mathbb{R}^n, \\ L_{21} \in \mathbb{R}^{(m-n)\times n}, & b_2 \in \mathbb{R}^{m-n}. \end{matrix}$$

Assume that L_{11} is lower triangular and nonsingular. If we apply forward elimination to $L_{11}x = b_1$, then x solves the system provided $L_{21}(L_{11}^{-1}b_1) = b_2$. Otherwise, there is no solution to the overall system. In such a case least squares minimization may be appropriate. See Chapter 5.

Now consider the lower triangular system $Lx = b$ when the number of columns n exceeds the number of rows m. We can apply forward substitution to the square

system $L(1{:}m,1{:}m)x(1{:}m,1{:}m) = b$ and prescribe an arbitrary value for $x(m+1{:}n)$. See §5.6 for additional comments on systems that have more unknowns than equations. The handling of nonsquare upper triangular systems is similar. Details are left to the reader.

3.1.7 The Algebra of Triangular Matrices

A *unit* triangular matrix is a triangular matrix with 1's on the diagonal. Many of the triangular matrix computations that follow have this added bit of structure. It clearly poses no difficulty in the above procedures.

For future reference we list a few properties about products and inverses of triangular and unit triangular matrices.

- The inverse of an upper (lower) triangular matrix is upper (lower) triangular.
- The product of two upper (lower) triangular matrices is upper (lower) triangular.
- The inverse of a unit upper (lower) triangular matrix is unit upper (lower) triangular.
- The product of two unit upper (lower) triangular matrices is unit upper (lower) triangular.

Problems

P3.1.1 Give an algorithm for computing a nonzero $z \in \mathbf{R}^n$ such that $Uz = 0$ where $U \in \mathbf{R}^{n\times n}$ is upper triangular with $u_{nn} = 0$ and $u_{11}\cdots u_{n-1,n-1} \neq 0$.

P3.1.2 Suppose $L = I_n - N$ is unit lower triangular where $N \in \mathbf{R}^{n\times n}$. Show that

$$L^{-1} = I_n + N + N^2 + \cdots + N^{n-1}.$$

What is the value of $\| L^{-1} \|_F$ if $N_{ij} = 1$ for all $i > j$?

P3.1.3 Write a detailed version of (3.1.4). Do not assume that N divides n.

P3.1.4 Prove all the facts about triangular matrices that are listed in §3.1.7.

P3.1.5 Suppose $S, T \in \mathbf{R}^{n\times n}$ are upper triangular and that $(ST - \lambda I)x = b$ is a nonsingular system. Give an $O(n^2)$ algorithm for computing x. Note that the explicit formation of $ST - \lambda I$ requires $O(n^3)$ flops. Hint: Suppose

$$S_+ = \begin{bmatrix} \sigma & u^T \\ 0 & S_c \end{bmatrix}, \quad T_+ = \begin{bmatrix} \tau & v^T \\ 0 & T_c \end{bmatrix}, \quad b_+ = \begin{bmatrix} \beta \\ b_c \end{bmatrix},$$

where $S_+ = S(k-1{:}n, k-1{:}n)$, $T_+ = T(k-1{:}n, k-1{:}n)$, $b_+ = b(k-1{:}n)$, and $\sigma, \tau, \beta \in \mathbf{R}$. Show that if we have a vector x_c such that

$$(S_cT_c - \lambda I)x_c = b_c$$

and $w_c = T_cx_c$ is available, then

$$x_+ = \begin{bmatrix} \gamma \\ x_c \end{bmatrix}, \qquad \gamma = \frac{\beta - \sigma v^T x_c - u^T w_c}{\sigma\tau - \lambda}$$

solves $(S_+T_+ - \lambda I)x_+ = b_+$. Observe that x_+ and $w_+ = T_+x_+$ each require $O(n-k)$ flops.

P3.1.6 Suppose the matrices $R_1, \ldots, R_p \in \mathbf{R}^{n\times n}$ are all upper triangular. Give an $O(pn^2)$ algorithm for solving the system $(R_1 \cdots R_p - \lambda I)x = b$ assuming that the matrix of coefficients is nonsingular. Hint. Generalize the solution to the previous problem.

P3.1.7 Suppose $L, K \in \mathbf{R}^{n\times n}$ are lower triangular and $B \in \mathbf{R}^{n\times n}$. Give an algorithm for computing $X \in \mathbf{R}^{n\times n}$ so that $LXK = B$.

Notes and References for §3.1

The accuracy of a computed solution to a triangular system is often surprisingly good, see:

N.J. Higham (1989). "The Accuracy of Solutions to Triangular Systems," *SIAM J. Numer. Anal. 26*, 1252–1265.

Solving systems of the form $(T_p \cdots T_1 - \lambda I)x = b$ where each T_i is triangular is considered in:

C.D. Martin and C.F. Van Loan (2002). "Product Triangular Systems with Shift," *SIAM J. Matrix Anal. Applic. 24*, 292–301.

The trick to obtaining an $O(pn^2)$ procedure that does not involve any matrix-matrix multiplications is to look carefully at the back-substitution recursions. See P3.1.6.

A survey of parallel triangular system solving techniques and their stabilty is given in:

N.J. Higham (1995). "Stability of Parallel Triangular System Solvers," *SIAM J. Sci. Comput. 16*, 400–413.

3.2 The LU Factorization

Triangular system solving is an easy $O(n^2)$ computation. The idea behind Gaussian elimination is to convert a given system $Ax = b$ to an equivalent triangular system. The conversion is achieved by taking appropriate linear combinations of the equations. For example, in the system

$$\begin{aligned} 3x_1 + 5x_2 &= 9, \\ 6x_1 + 7x_2 &= 4, \end{aligned}$$

if we multiply the first equation by 2 and subtract it from the second we obtain

$$\begin{aligned} 3x_1 + 5x_2 &= 9, \\ -3x_2 &= -14. \end{aligned}$$

This is $n = 2$ Gaussian elimination. Our objective in this section is to describe the procedure in the language of matrix factorizations. This means showing that the algorithm computes a unit lower triangular matrix L and an upper triangular matrix U so that $A = LU$, e.g.,

$$\begin{bmatrix} 3 & 5 \\ 6 & 7 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 3 & 5 \\ 0 & -3 \end{bmatrix}.$$

The solution to the original $Ax = b$ problem is then found by a two-step triangular solve process:

$$Ly = b, \quad Ux = y \qquad \Longrightarrow \qquad Ax = LUx = Ly = b. \tag{3.2.1}$$

The LU factorization is a "high-level" algebraic description of Gaussian elimination. Linear equation solving is not about the matrix vector product $A^{-1}b$ but about computing LU and using it effectively; see §3.4.9. Expressing the outcome of a matrix algorithm in the "language" of matrix factorizations is a productive exercise, one that is repeated many times throughout this book. It facilitates generalization and highlights connections between algorithms that can appear very different at the scalar level.

3.2.1 Gauss Transformations

To obtain a factorization description of Gaussian elimination as it is traditionally presented, we need a matrix description of the zeroing process. At the $n = 2$ level, if $v_1 \neq 0$ and $\tau = v_2/v_1$, then

$$\begin{bmatrix} 1 & 0 \\ -\tau & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} v_1 \\ 0 \end{bmatrix}.$$

More generally, suppose $v \in \mathbb{R}^n$ with $v_k \neq 0$. If

$$\tau^T = [\underbrace{0, \ldots, 0}_{k}, \tau_{k+1}, \ldots, \tau_n], \qquad \tau_i = \frac{v_i}{v_k}, \quad i = k+1{:}n,$$

and we define

$$M_k = I_n - \tau e_k^T, \tag{3.2.2}$$

then

$$M_k v = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & & 1 & 0 & & 0 \\ 0 & & -\tau_{k+1} & 1 & & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -\tau_n & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_k \\ v_{k+1} \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} v_1 \\ \vdots \\ v_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

A matrix of the form $M_k = I_n - \tau e_k^T \in \mathbb{R}^{n \times n}$ is a *Gauss transformation* if the first k components of $\tau \in \mathbb{R}^n$ are zero. Such a matrix is unit lower triangular. The components of $\tau(k+1{:}n)$ are called *multipliers*. The vector τ is called the *Gauss vector*.

3.2.2 Applying Gauss Transformations

Multiplication by a Gauss transformation is particularly simple. If $C \in \mathbb{R}^{n \times r}$ and $M_k = I_n - \tau e_k^T$ is a Gauss transformation, then

$$M_k C = (I_n - \tau e_k^T)C = C - \tau\,(e_k^T C) = C - \tau\, C(k,:)$$

is an outer product update. Since $\tau(1{:}k) = 0$ only $C(k+1{:}n,:)$ is affected and the update $C = M_k C$ can be computed row by row as follows:

for $i = k+1{:}n$
  $C(i,:) = C(i,:) - \tau_i \cdot C(k,:)$
end

This computation requires $2(n-k)r$ flops. Here is an example:

$$C = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{bmatrix}, \quad \tau = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix} \quad \Longrightarrow \quad (I - \tau e_1^T)C = \begin{bmatrix} 1 & 4 & 7 \\ 1 & 1 & 1 \\ 4 & 10 & 17 \end{bmatrix}.$$

3.2.3 Roundoff Properties of Gauss Transformations

If $\hat{\tau}$ is the computed version of an exact Gauss vector τ, then it is easy to verify that

$$\hat{\tau} = \tau + e, \qquad |e| \leq \mathbf{u}|\tau|.$$

If $\hat{\tau}$ is used in a Gauss transform update and $\mathrm{fl}((I_n - \hat{\tau}e_k^T)C)$ denotes the computed result, then

$$\mathrm{fl}\left((I_n - \hat{\tau}e_k^T)C\right) = (I - \tau e_k^T)C + E,$$

where

$$|E| \leq 3\mathbf{u}\left(|C| + |\tau||C(k,:)|\right) + O(\mathbf{u}^2).$$

Clearly, if τ has large components, then the errors in the update may be large in comparison to $|C|$. For this reason, care must be exercised when Gauss transformations are employed, a matter that is pursued in §3.4.

3.2.4 Upper Triangularizing

Assume that $A \in \mathbb{R}^{n\times n}$. Gauss transformations $M_1, \ldots, M_{n-1}$ can usually be found such that $M_{n-1}\cdots M_2 M_1 A = U$ is upper triangular. To see this we first look at the $n = 3$ case. Suppose

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{bmatrix}$$

and note that

$$M_1 = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -3 & 0 & 1 \end{bmatrix} \Rightarrow M_1 A = \begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & -6 & -11 \end{bmatrix}.$$

Likewise, in the second step we have

$$M_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix} \Rightarrow M_2(M_1 A) = \begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{bmatrix}.$$

Extrapolating from this example to the general n case we conclude two things.

- At the start of the kth step we have a matrix $A^{(k-1)} = M_{k-1}\cdots M_1 A$ that is upper triangular in columns 1 through $k-1$.
- The multipliers in the kth Gauss transform M_k are based on $A^{(k-1)}(k+1{:}n, k)$ and $a_{kk}^{(k-1)}$ must be nonzero in order to proceed.

Noting that complete upper triangularization is achieved after $n-1$ steps, we obtain the following rough draft of the overall process:

$A^{(1)} = A$

for $k = 1{:}n-1$

For $i = k+1{:}n$, determine the multipliers $\tau_i^{(k)} = a_{ik}^{(k)}/a_{kk}^{(k)}$. (3.2.3)

Apply $M_k = I - \tau^{(k)}e_k^T$ to obtain $A^{(k+1)} = M_k A^{(k)}$.

end

For this process to be well-defined, the matrix entries $a_{11}^{(1)}, a_{22}^{(2)}, \ldots, a_{n-1,n-1}^{(n-1)}$ must be nonzero. These quantities are called *pivots*.

3.2.5 Existence

If no zero pivots are encountered in (3.2.3), then Gauss transformations $M_1, \ldots, M_{n-1}$ are generated such that $M_{n-1} \cdots M_1 A = U$ is upper triangular. It is easy to check that if $M_k = I_n - \tau^{(k)} e_k^T$, then its inverse is prescribed by $M_k^{-1} = I_n + \tau^{(k)} e_k^T$ and so

$$A = LU \tag{3.2.4}$$

where

$$L = M_1^{-1} \cdots M_{n-1}^{-1}. \tag{3.2.5}$$

It is clear that L is a unit lower triangular matrix because each M_k^{-1} is unit lower triangular. The factorization (3.2.4) is called the *LU factorization*.

The LU factorization may not exist. For example, it is impossible to find l_{ij} and u_{ij} so

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 7 \\ 3 & 5 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}.$$

To see this, equate entries and observe that we must have $u_{11} = 1$, $u_{12} = 2$, $\ell_{21} = 2$, $u_{22} = 0$, and $\ell_{31} = 3$. But then the (3,2) entry gives us the contradictory equation $5 = \ell_{31} u_{12} + \ell_{32} u_{22} = 6$. For this example, the pivot $a_{22}^{(1)} = a_{22} - (a_{21}/a_{11}) a_{12}$ is zero.

It turns out that the kth pivot in (3.2.3) is zero if $A(1{:}k, 1{:}k)$ is singular. A submatrix of the form $A(1{:}k, 1{:}k)$ is called a *leading principal submatrix*.

Theorem 3.2.1. (LU Factorization). *If $A \in \mathbb{R}^{n \times n}$ and $\det(A(1{:}k, 1{:}k)) \neq 0$ for $k = 1{:}n-1$, then there exists a unit lower triangular $L \in \mathbb{R}^{n \times n}$ and an upper triangular $U \in \mathbb{R}^{n \times n}$ such that $A = LU$. If this is the case and A is nonsingular, then the factorization is unique and $\det(A) = u_{11} \cdots u_{nn}$.*

Proof. Suppose $k-1$ steps in (3.2.3) have been executed. At the beginning of step k the matrix A has been overwritten by $M_{k-1} \cdots M_1 A = A^{(k-1)}$. Since Gauss transformations are unit lower triangular, it follows by looking at the leading k-by-k portion of this equation that

$$\det(A(1{:}k, 1{:}k)) = a_{11}^{(k-1)} \cdots a_{kk}^{(k-1)}. \tag{3.2.6}$$

Thus, if $A(1{:}k, 1{:}k)$ is nonsingular, then the kth pivot $a_{kk}^{(k-1)}$ is nonzero.

As for uniqueness, if $A = L_1 U_1$ and $A = L_2 U_2$ are two LU factorizations of a nonsingular A, then $L_2^{-1} L_1 = U_2 U_1^{-1}$. Since $L_2^{-1} L_1$ is unit lower triangular and $U_2 U_1^{-1}$ is upper triangular, it follows that both of these matrices must equal the identity. Hence, $L_1 = L_2$ and $U_1 = U_2$. Finally, if $A = LU$, then

$$\det(A) = \det(LU) = \det(L)\det(U) = \det(U).$$

It follows that $\det(A) = u_{11} \cdots u_{nn}$. □

3.2.6 L Is the Matrix of Multipliers

It turns out that the construction of L is not nearly so complicated as Equation (3.2.5) suggests. Indeed,

$$\begin{aligned} L &= M_1^{-1} \cdots M_{n-1}^{-1} \\ &= \left(I_n - \tau^{(1)} e_1^T\right)^{-1} \cdots \left(I_n - \tau^{(n-1)} e_{n-1}^T\right)^{-1} \\ &= \left(I_n + \tau^{(1)} e_1^T\right) \cdots \left(I_n + \tau^{(n-1)} e_{n-1}^T\right) \\ &= I_n + \sum_{k=1}^{n-1} \tau^{(k)} e_k^T \end{aligned}$$

showing that

$$L(k+1{:}n, k) = \tau^{(k)}(k+1{:}n) \qquad k = 1{:}n-1. \tag{3.2.7}$$

In other words, the kth column of L is defined by the multipliers that arise in the k-th step of (3.2.3). Consider the example in §3.2.4:

$$\tau^{(1)} = \begin{bmatrix} 0 \\ 2 \\ 3 \end{bmatrix}, \quad \tau^{(2)} = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{bmatrix}.$$

3.2.7 The Outer Product Point of View

Since the application of a Gauss transformation to a matrix involves an outer product, we can regard (3.2.3) as a sequence of outer product updates. Indeed, if

$$A = \begin{bmatrix} \alpha & w^T \\ v & B \end{bmatrix} \begin{matrix} 1 \\ n-1 \end{matrix}$$
$$\qquad\quad 1 \quad n-1$$

then the first step in Gaussian elimination results in the decomposition

$$\begin{bmatrix} \alpha & w^T \\ z & B \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ z/\alpha & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & B - zw^T/\alpha \end{bmatrix} \begin{bmatrix} \alpha & w^T \\ 0 & I_{n-1} \end{bmatrix}.$$

Steps 2 through $n-1$ compute the LU factorization

$$B - zw^T/\alpha = L_1 U_1$$

for then

$$A = \begin{bmatrix} 1 & 0 \\ z/\alpha & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & L_1 U_1 \end{bmatrix} \begin{bmatrix} \alpha & w^T \\ 0 & I_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ z/\alpha & L_1 \end{bmatrix} \begin{bmatrix} \alpha & w^T \\ 0 & U_1 \end{bmatrix} \equiv LU.$$

3.2.8 Practical Implementation

Let us consider the efficient implementation of (3.2.3). First, because zeros have already been introduced in columns 1 through $k-1$, the Gauss transformation update need only be applied to columns k through n. Of course, we need not even apply the kth Gauss transform to $A(:,k)$ since we know the result. So the efficient thing to do is simply to update $A(k+1{:}n, k+1{:}n)$. Also, the observation (3.2.7) suggests that we can overwrite $A(k+1{:}n, k)$ with $L(k+1{:}n, k)$ since the latter houses the multipliers that are used to zero the former. Overall we obtain:

Algorithm 3.2.1 (Outer Product LU) Suppose $A \in \mathbb{R}^{n\times n}$ has the property that $A(1{:}k, 1{:}k)$ is nonsingular for $k = 1{:}n-1$. This algorithm computes the factorization $A = LU$ where L is unit lower triangular and U is upper triangular. For $i = 1{:}n-1$, $A(i, i{:}n)$ is overwritten by $U(i, i{:}n)$ while $A(i+1{:}n, i)$ is overwritten by $L(i+1{:}n, i)$.

```
for k = 1:n − 1
    ρ = k + 1:n
    A(ρ, k) = A(ρ, k)/A(k, k)
    A(ρ, ρ) = A(ρ, ρ) − A(ρ, k)·A(k, ρ)
end
```

This algorithm involves $2n^3/3$ flops and it is one of several formulations of *Gaussian elimination*. Note that the k-th step involves an $(n-k)$-by-$(n-k)$ outer product.

3.2.9 Other Versions

Similar to matrix-matrix multiplication, Gaussian elimination is a triple-loop procedure that can be arranged in several ways. Algorithm 3.2.1 corresponds to the "kij" version of Gaussian elimination if we compute the outer product update row by row:

```
for k = 1:n − 1
    A(k + 1:n, k) = A(k + 1:n, k)/A(k, k)
    for i = k + 1:n
        for j = k + 1:n
            A(i, j) = A(i, j) − A(i, k)·A(k, j)
        end
    end
end
```

There are five other versions: kji, ikj, ijk, jik, and jki. The last of these results in an implementation that features a sequence of gaxpys and forward eliminations which we now derive at the vector level.

The plan is to compute the jth columns of L and U in step j. If $j = 1$, then by comparing the first columns in $A = LU$ we conclude that

$$L(2{:}n, j) = A(2{:}n, 1)/A(1,1)$$

and $U(1,1) = A(1,1)$. Now assume that $L(:, 1{:}j-1)$ and $U(1{:}j-1, 1{:}j-1)$ are known. To get the jth columns of L and U we equate the jth columns in the equation $A = LU$

and infer from the vector equation $A(:,j) = LU(:,j)$ that

$$A(1{:}j-1,j) = L(1{:}j-1,1{:}j-1)\cdot U(1{:}j-1,j)$$

and

$$A(j{:}n,j) = \sum_{k=1}^{j} L(j{:}n,k)\cdot U(k,j).$$

The first equation is a lower triangular linear system that can be solved for the vector $U(1{:}j-1,j)$. Once this is accomplished, the second equation can be rearranged to produce recipes for $U(j,j)$ and $L(j+1{:}n,j)$. Indeed, if we set

$$\begin{aligned} v(j{:}n) &= A(j{:}n,j) - \sum_{k=1}^{j-1} L(j{:}n,k)U(k,j) \\ &= A(j{:}n,j) - L(j{:}n,1{:}j-1)\cdot U(1{:}j-1,j), \end{aligned}$$

then $L(j+1{:}n,j) = v(j+1{:}n)/v(j)$ and $U(j,j) = v(j)$. Thus, $L(j+1{:}n,j)$ is a scaled gaxpy and we obtain the following alternative to Algorithm 3.2.1:

Algorithm 3.2.2 (Gaxpy LU) Suppose $A \in \mathbb{R}^{n\times n}$ has the property that $A(1{:}k,1{:}k)$ is nonsingular for $k = 1{:}n-1$. This algorithm computes the factorization $A = LU$ where L is unit lower triangular and U is upper triangular.

```
Initialize L to the identity and U to the zero matrix.
for j = 1:n
    if j = 1
        v = A(:,1)
    else
        ã = A(:,j)
        Solve L(1:j−1,1:j−1)·z = ã(1:j−1) for z ∈ R^{j−1}.
        U(1:j−1,j) = z
        v(j:n) = ã(j:n) − L(j:n,1:j−1)·z
    end
    U(j,j) = v(j)
    L(j+1:n,j) = v(j+1:n)/v(j)
end
```

(We chose to have separate arrays for L and U for clarity; it is not necessary in practice.) Algorithm 3.2.2 requires $2n^3/3$ flops, the same volume of floating point work required by Algorithm 3.2.1. However, from §1.5.2 there is less memory traffic associated with a gaxpy than with an outer product, so the two implementations could perform differently in practice. Note that in Algorithm 3.2.2, the original $A(:,j)$ is untouched until step j.

The terms *right-looking* and *left-looking* are sometimes applied to Algorithms 3.2.1 and 3.2.2. In the outer-product implementation, after $L(k{:}n,k)$ is determined, the columns to the right of $A(:,k)$ are updated so it is a right-looking procedure. In contrast, subcolumns to the left of $A(:,k)$ are accessed in gaxpy LU before $L(k+1{:}n,k)$ is produced so that implementation left-looking.

3.2.10 The LU Factorization of a Rectangular Matrix

The LU factorization of a rectangular matrix $A \in \mathbb{R}^{n\times r}$ can also be performed. The $n > r$ case is illustrated by

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 3 & 1 \\ 5 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 0 & -2 \end{bmatrix}$$

while

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \end{bmatrix}$$

depicts the $n < r$ situation. The LU factorization of $A \in \mathbb{R}^{n\times r}$ is guaranteed to exist if $A(1{:}k, 1{:}k)$ is nonsingular for $k = 1{:}\min\{n, r\}$.

The square LU factorization algorithms above needs only minor alterations to handle the rectangular case. For example, if $n > r$, then Algorithm 3.2.1 modifies to the following:

```
for k = 1:r
    ρ = k + 1:n
    A(ρ,k) = A(ρ,k)/A(k,k)
    if k < r
        μ = k + 1:r
        A(ρ,μ) = A(ρ,μ) − A(ρ,k)·A(k,μ)
    end
end
```
(3.2.8)

This calculation requires $nr^2 - r^3/3$ flops. Upon completion, A is overwritten by the strictly lower triangular portion of $L \in \mathbb{R}^{n\times r}$ and the upper triangular portion of $U \in \mathbb{R}^{r\times r}$.

3.2.11 Block LU

It is possible to organize Gaussian elimination so that matrix multiplication becomes the dominant operation. Partition $A \in \mathbb{R}^{n\times n}$ as follows:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{matrix} r \\ n-r \end{matrix}$$
$$\qquad\quad r \quad n-r$$

where r is a blocking parameter. Suppose we compute the LU factorization

$$\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} = \begin{bmatrix} L_{11} \\ L_{21} \end{bmatrix} U_{11}.$$

Here, $L_{11} \in \mathbb{R}^{r\times r}$ is unit lower triangular and $U_{11} \in \mathbb{R}^{r\times r}$ is upper triangular and assumed to be nonsingular. If we solve $L_{11}U_{12} = A_{12}$ for $U_{12} \in \mathbb{R}^{r\times n-r}$, then

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I_{n-r} \end{bmatrix} \begin{bmatrix} I_r & 0 \\ 0 & \tilde{A} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & I_{n-r} \end{bmatrix},$$

where

$$\tilde{A} = A_{22} - L_{21}U_{12} = A_{22} - A_{21}A_{11}^{-1}A_{12} \tag{3.2.9}$$

is the *Schur complement* of A_{11} in A. Note that if

$$\tilde{A} = L_{22}U_{22}$$

is the LU factorization of $\tilde{A}$, then

$$A = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$$

is the LU factorization of A. This lays the groundwork for a recursive implementation.

Algorithm 3.2.3 (Recursive Block LU) Suppose $A \in \mathbb{R}^{n\times n}$ has an LU factorization and r is a positive integer. The following algorithm computes unit lower triangular $L \in \mathbb{R}^{n\times n}$ and upper triangular $U \in \mathbb{R}^{n\times n}$ so $A = LU$.

function $[L,U] = \mathsf{BlockLU}(A,n,r)$
  if $n \le r$
    Compute the LU factorization $A = LU$ using (say) Algorithm 3.2.1.
  else
    Use (3.2.8) to compute the LU factorization $A(:,1{:}r) = \begin{bmatrix} L_{11} \\ L_{21} \end{bmatrix} U_{11}$.
    Solve $L_{11}U_{12} = A(1{:}r, r+1{:}n)$ for U_{12}.
    $\tilde{A} = A(r+1{:}n, r+1{:}n) - L_{21}U_{12}$
    $[L_{22}, U_{22}] = \mathsf{BlockLU}(\tilde{A}, n-r, r)$
    $L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}$, $U = \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$
  end
end

The following table explains where the flops come from:

Activity	Flops
L_{11}, L_{21}, U_{11}	$nr^2 - r^3/3$
U_{12}	$(n-r)r^2$
$\tilde{A}$	$2(n-r)^2$

If $n \gg r$, then there are a total of about $2n^3/3$ flops, the same volume of atithmetic as Algorithms 3.2.1 and 3.2.2. The vast majority of these flops are the level-3 flops associated with the production of $\tilde{A}$.

The actual level-3 fraction, a concept developed in §3.1.5, is more easily derived from a nonrecursive implementation. Assume for clarity that $n = Nr$ where N is a positive integer and that we want to compute

$$\begin{bmatrix} A_{11} & \cdots & A_{1N} \\ \vdots & \ddots & \vdots \\ A_{N1} & \cdots & A_{NN} \end{bmatrix} = \begin{bmatrix} L_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ L_{N1} & \cdots & L_{NN} \end{bmatrix} \begin{bmatrix} U_{11} & \cdots & U_{1N} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & U_{NN} \end{bmatrix} \tag{3.2.10}$$

where all blocks are r-by-r. Analogously to Algorithm 3.2.3 we have the following.

Algorithm 3.2.4 (Nonrecursive Block LU) Suppose $A \in \mathbb{R}^{n\times n}$ has an LU factorization and r is a positive integer. The following algorithm computes unit lower triangular $L \in \mathbb{R}^{n\times n}$ and upper triangular $U \in \mathbb{R}^{n\times n}$ so $A = LU$.

for $k = 1{:}N$

Rectangular Gaussian elimination:

$$\begin{bmatrix} A_{kk} \\ \vdots \\ A_{Nk} \end{bmatrix} = \begin{bmatrix} L_{kk} \\ \vdots \\ L_{Nk} \end{bmatrix} U_{kk}$$

Multiple right hand side solve:

$$L_{kk}\left[\; U_{k,k+1} \mid \cdots \mid U_{kN} \;\right] = \left[\; A_{k,k+1} \mid \cdots \mid A_{kN} \;\right]$$

Level-3 updates:

$$A_{ij} = A_{ij} - L_{ik}U_{kj}, \quad i = k+1{:}N,\ j = k+1{:}N$$

end

Here is the flop situation during the kth pass through the loop:

Activity	Flops
Gaussian elimination	$(N-k+1)r^3 - r^3/3$
Multiple RHS solve	$(N-k)r^3$
Level-3 updates	$2(N-k)^2r^2$

Summing these quantities for $k = 1{:}N$ we find that the level-3 fraction is approximately

$$\frac{2n^3/3}{2n^3/3 + n^2r} = 1 - \frac{3}{2N}.$$

Thus, for large N almost all arithmetic takes place in the context of matrix multiplication. This ensures a favorable amount of data reuse as discussed in §1.5.4.

Problems

P3.2.1 Verify Equation (3.2.6).

P3.2.2 Suppose the entries of $A(\epsilon) \in \mathbb{R}^{n\times n}$ are continuously differentiable functions of the scalar ϵ. Assume that $A \equiv A(0)$ and all its principal submatrices are nonsingular. Show that for sufficiently small ϵ, the matrix $A(\epsilon)$ has an LU factorization $A(\epsilon) = L(\epsilon)U(\epsilon)$ and that $L(\epsilon)$ and $U(\epsilon)$ are both continuously differentiable.

P3.2.3 Suppose we partition $A \in \mathbb{R}^{n\times n}$

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

where A_{11} is r-by-r and nonsingular. Let S be the Schur complement of A_{11} in A as defined in (3.2.9). Show that after r steps of Algorithm 3.2.1, $A(r+1{:}n, r+1{:}n)$ houses S. How could S be obtained after r steps of Algorithm 3.2.2?

P3.2.4 Suppose $A \in \mathbb{R}^{n \times n}$ has an LU factorization. Show how $Ax = b$ can be solved without storing the multipliers by computing the LU factorization of the n-by-$(n+1)$ matrix $[A\ b]$.

P3.2.5 Describe a variant of Gaussian elimination that introduces zeros into the columns of A in the order, $n\text{:} -1\text{:}2$ and which produces the factorization $A = UL$ where U is unit upper triangular and L is lower triangular.

P3.2.6 Matrices in $\mathbb{R}^{n \times n}$ of the form $N(y,k) = I - ye_k^T$ where $y \in \mathbb{R}^n$ are called *Gauss-Jordan transformations.* (a) Give a formula for $N(y,k)^{-1}$ assuming it exists. (b) Given $x \in \mathbb{R}^n$, under what conditions can y be found so $N(y,k)x = e_k$? (c) Give an algorithm using Gauss-Jordan transformations that overwrites A with A^{-1}. What conditions on A ensure the success of your algorithm?

P3.2.7 Extend Algorithm 3.2.2 so that it can also handle the case when A has more rows than columns.

P3.2.8 Show how A can be overwritten with L and U in Algorithm 3.2.2. Give a 3-loop specification so that unit stride access prevails.

P3.2.9 Develop a version of Gaussian elimination in which the innermost of the three loops oversees a dot product.

Notes and References for §3.2

The method of Gaussian elimination has a long and interesting history, see:

J.F. Grcar (2011). "How Ordinary Elimination Becamc Gaussian Elimination," *Historica Mathematica, 38,* 163–218.
J.F. Grcar (2011). "Mathematicians of Gaussian Elimination," *Notices of the AMS 58,* 782–792.

Schur complements (3.2.9) arise in many applications. For a survey of both practical and theoretical interest, see:

R.W. Cottle (1974). "Manifestations of the Schur Complement," *Lin. Alg. Applic. 8,* 189–211.

Schur complements are known as "Gauss transforms" in some application areas. The use of Gauss-Jordan transformations (P3.2.6) is detailed in Fox (1964). See also:

T. Dekker and W. Hoffman (1989). "Rehabilitation of the Gauss-Jordan Algorithm," *Numer. Math. 54,* 591–599.

As we mentioned, inner product versions of Gaussian elimination have been known and used for some time. The names of Crout and Doolittle are associated with these techniques, see:

G.E. Forsythe (1960). "Crout with Pivoting," *Commun. ACM 3,* 507–508.
W.M. McKeeman (1962). "Crout with Equilibration and Iteration," *Commun. ACM. 5,* 553–555.

Loop orderings and block issues in LU computations are discussed in:

J.J. Dongarra, F.G. Gustavson, and A. Karp (1984). "Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine," *SIAM Review 26,* 91–112.
J.M. Ortega (1988). "The *ijk* Forms of Factorization Methods I: Vector Computers," *Parallel Comput. 7,* 135–147.
D.H. Bailey, K.Lee, and H.D. Simon (1991). "Using Strassen's Algorithm to Accelerate the Solution of Linear Systems," *J. Supercomput. 4,* 357–371.
J.W. Demmel, N.J. Higham, and R.S. Schreiber (1995). "Stability of Block LU Factorization," *Numer. Lin. Alg. Applic. 2,* 173–190.

Suppose $A = LU$ and $A+\Delta A = (L+\Delta L)(U+\Delta U)$ are LU factorizations. Bounds on the perturbations ΔL and ΔU in terms of ΔA are given in:

G.W. Stewart (1997). "On the Perturbation of LU and Cholesky Factors," *IMA J. Numer. Anal. 17,* 1–6.
X.-W. Chang and C.C. Paige (1998). "On the Sensitivity of the LU factorization," *BIT 38,* 486–501.

In certain limited domains, it is possible to solve linear systems exactly using rational arithmetic. For a snapshot of the challenges, see:

P. Alfeld and D.J. Eyre (1991). "The Exact Analysis of Sparse Rectangular Linear Systems," *ACM Trans. Math. Softw. 17*, 502–518.

P. Alfeld (2000). "Bivariate Spline Spaces and Minimal Determining Sets," *J. Comput. Appl. Math. 119*, 13–27.

3.3 Roundoff Error in Gaussian Elimination

We now assess the effect of rounding errors when the algorithms in the previous two sections are used to solve the linear system $Ax = b$. A much more detailed treatment of roundoff error in Gaussian elimination is given in Higham (ASNA).

3.3.1 Errors in the LU Factorization

Let us see how the error bounds for Gaussian elimination compare with the ideal bounds derived in §2.7.11. We work with the infinity norm for convenience and focus our attention on Algorithm 3.2.1, the outer product version. The error bounds that we derive also apply to the gaxpy formulation (Algorithm 3.2.2). Our first task is to quantify the roundoff errors associated with the computed triangular factors.

Theorem 3.3.1. *Assume that A is an n-by-n matrix of floating point numbers. If no zero pivots are encountered during the execution of Algorithm 3.2.1, then the computed triangular matrices $\hat{L}$ and $\hat{U}$ satisfy*

$$\hat{L}\hat{U} = A + H, \tag{3.3.1}$$

$$|H| \leq 2(n-1)\mathbf{u}\left(|A| + |\hat{L}||\hat{U}|\right) + O(\mathbf{u}^2)\,. \tag{3.3.2}$$

Proof. The proof is by induction on n. The theorem obviously holds for $n = 1$. Assume that $n \geq 2$ and that the theorem holds for all $(n-1)$-by-$(n-1)$ floating point matrices. If A is partitioned as follows

$$A = \begin{bmatrix} \alpha & w^T \\ v & B \end{bmatrix} \begin{matrix} 1 \\ n-1 \end{matrix}$$

$$\qquad\quad 1 \quad n-1$$

then the first step in Algorithm 3.2.1 is to compute

$$\hat{z} = \mathrm{fl}(v/\alpha), \qquad \hat{C} = \mathrm{fl}(\hat{z}w^T), \qquad \hat{A}_1 = \mathrm{fl}(B - \hat{C}),$$

from which we conclude that

$$\hat{z} = v/\alpha + f, \tag{3.3.3}$$

$$|f| \leq \mathbf{u}|v/\alpha|, \tag{3.3.4}$$

$$\hat{C} = \hat{z}w^T + F_1, \tag{3.3.5}$$

$$|F_1| \leq \mathbf{u}|\hat{z}||w^T|, \tag{3.3.6}$$

$$\hat{A}_1 = B - (\hat{z}w^T + F_1) + F_2, \tag{3.3.7}$$

$$|F_2| \leq \mathbf{u}\left(|B| + |\hat{z}||w^T|\right) + O(\mathbf{u}^2), \tag{3.3.8}$$

$$|\hat{A}_1| \leq |B| + |\hat{z}||w^T| + O(\mathbf{u}). \tag{3.3.9}$$

The algorithm proceeds to compute the LU factorization of $\hat{A}_1$. By induction, the computed factors $\hat{L}_1$ and $\hat{U}_1$ satisfy

$$\hat{L}_1\hat{U}_1 = \hat{A}_1 + H_1 \tag{3.3.10}$$

where

$$|H_1| \leq 2(n-2)\mathbf{u}\left(|\hat{A}_1| + |\hat{L}_1||\hat{U}_1|\right) + O(\mathbf{u}^2). \tag{3.3.11}$$

If

$$\hat{L} = \begin{bmatrix} 1 & 0 \\ \hat{z} & \hat{L}_1 \end{bmatrix}, \qquad \hat{U} = \begin{bmatrix} \alpha & w^T \\ 0 & \hat{U}_1 \end{bmatrix},$$

then it is easy to verify that

$$\hat{L}\hat{U} = A + H$$

where

$$H = \begin{bmatrix} 0 & 0 \\ \alpha f & H_1 - F_1 + F_2 \end{bmatrix}. \tag{3.3.12}$$

To prove the theorem we must verify (3.3.2), i.e.,

$$|H| \leq 2(n-1)\mathbf{u}\begin{bmatrix} 2|\alpha| & 2|w^T| \\ |v| + |\alpha||f| & |B| + |\hat{L}_1||\hat{U}_1| + |\hat{z}||w^T| \end{bmatrix} + O(\mathbf{u}^2).$$

Considering (3.3.12), this is obviously the case if

$$|H_1| + |F_1| + |F_2| \leq 2(n-1)\mathbf{u}\left(|B| + |\hat{z}||w^T| + |\hat{L}_1||\hat{U}_1|\right) + O(\mathbf{u}^2). \tag{3.3.13}$$

Using (3.3.9) and (3.3.11) we have

$$|H_1| \leq 2(n-2)\mathbf{u}\left(|B| + |\hat{z}||w^T| + |\hat{L}_1||\hat{U}_1|\right) + O(\mathbf{u}^2),$$

while (3.3.6) and (3.3.8) imply

$$|F_1| + |F_2| \leq \mathbf{u}(|B| + 2|\hat{z}||w|) + O(\mathbf{u}^2).$$

These last two results establish (3.3.13) and therefore the theorem. □

We mention that if A is m-by-n, then the theorem applies with n replaced by the smaller of n and m in Equation 3.3.2.

3.3.2 Triangular Solving with Inexact Triangles

We next examine the effect of roundoff error when $\hat{L}$ and $\hat{U}$ are used by the triangular system solvers of §3.1.

Theorem 3.3.2. *Let $\hat{L}$ and $\hat{U}$ be the computed LU factors obtained by Algorithm 3.2.1 when it is applied to an n-by-n floating point matrix A. If the methods of §3.1 are used to produce the computed solution $\hat{y}$ to $\hat{L}y = b$ and the computed solution $\hat{x}$ to $\hat{U}x = \hat{y}$, then $(A + E)\hat{x} = b$ with*

$$|E| \leq n\mathbf{u}\left(2|A| + 4|\hat{L}||\hat{U}|\right) + O(\mathbf{u}^2)\,. \tag{3.3.14}$$

Proof. From (3.1.1) and (3.1.2) we have

$$\begin{aligned} (\hat{L} + F)\hat{y} &= b, & |F| &\leq n\mathbf{u}|\hat{L}| + O(\mathbf{u}^2), \\ (\hat{U} + G)\hat{x} &= \hat{y}, & |G| &\leq n\mathbf{u}|\hat{U}| + O(\mathbf{u}^2), \end{aligned}$$

and thus

$$(\hat{L} + F)(\hat{U} + G)\hat{x} = (\hat{L}\hat{U} + F\hat{U} + \hat{L}G + FG)\hat{x} = b.$$

If follows from Theorem 3.3.1 that $\hat{L}\hat{U} = A + H$ with

$$|H| \leq 2(n-1)\mathbf{u}(|A| + |\hat{L}||\hat{U}|) + O(\mathbf{u}^2),$$

and so by defining

$$E = H + F\hat{U} + \hat{L}G + FG$$

we find $(A + E)\hat{x} = b$. Moreover,

$$\begin{aligned} |E| &\leq |H| + |F|\,|\hat{U}| + |\hat{L}|\,|G| + O(\mathbf{u}^2) \\ &\leq 2n\mathbf{u}\left(|A| + |\hat{L}||\hat{U}|\right) + 2n\mathbf{u}\left(|\hat{L}||\hat{U}|\right) + O(\mathbf{u}^2), \end{aligned}$$

completing the proof of the theorem. □

If it were not for the possibility of a large $|\hat{L}||\hat{U}|$ term, (3.3.14) would compare favorably with the ideal bound (2.7.21). (The factor n is of no consequence, cf. the Wilkinson quotation in §2.7.7.) Such a possibility exists, for there is nothing in Gaussian elimination to rule out the appearance of small pivots. If a small pivot is encountered, then we can expect large numbers to be present in $\hat{L}$ and $\hat{U}$.

We stress that small pivots are not necessarily due to ill-conditioning as the example

$$A = \begin{bmatrix} \epsilon & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{bmatrix} \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix}$$

shows. Thus, Gaussian elimination can give arbitrarily poor results, even for well-conditioned problems. The method is unstable. For example, suppose 3-digit floating point arithmetic is used to solve

$$\begin{bmatrix} .001 & 1.00 \\ 1.00 & 2.00 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1.00 \\ 3.00 \end{bmatrix}.$$

(See §2.7.1.) Applying Gaussian elimination we get

$$\hat{L} = \begin{bmatrix} 1 & 0 \\ 1000 & 1 \end{bmatrix}, \qquad \hat{U} = \begin{bmatrix} .001 & 1 \\ 0 & -1000 \end{bmatrix},$$

and a calculation shows that

$$\hat{L}\hat{U} = \begin{bmatrix} .001 & 1 \\ 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & -2 \end{bmatrix} \equiv A + H.$$

If we go on to solve the problem using the triangular system solvers of §3.1, then using the same precision arithmetic we obtain a computed solution $\hat{x} = [0\ ,\ 1]^T$. This is in contrast to the exact solution $x = [1.002\ldots, .998\ldots]^T$.

Problems

P3.3.1 Show that if we drop the assumption that A is a floating point matrix in Theorem 3.3.1, then Equation 3.3.2 holds with the coefficient "2" replaced by "3."

P3.3.2 Suppose A is an n-by-n matrix and that $\hat{L}$ and $\hat{U}$ are produced by Algorithm 3.2.1. (a) How many flops are required to compute $\| \, |\hat{L}| \, |\hat{U}| \, \|_\infty$? (b) Show $\mathrm{fl}(|\hat{L}||\hat{U}|) \leq (1 + 2n\mathbf{u})|\hat{L}||\hat{U}| + O(\mathbf{u}^2)$.

Notes and References for §3.3

The original roundoff analysis of Gaussian elimination appears in:

J.H. Wilkinson (1961). "Error Analysis of Direct Methods of Matrix Inversion," *J. ACM 8*, 281–330.

Various improvements and insights regarding the bounds and have been made over the years, see:

B.A. Chartres and J.C. Geuder (1967). "Computable Error Bounds for Direct Solution of Linear Equations," *J. ACM 14*, 63–71.

J.K. Reid (1971). "A Note on the Stability of Gaussian Elimination," *J. Inst. Math. Applic. 8*, 374–75.

C.C. Paige (1973). "An Error Analysis of a Method for Solving Matrix Equations," *Math. Comput. 27*, 355–59.

H.H. Robertson (1977). "The Accuracy of Error Estimates for Systems of Linear Algebraic Equations," *J. Inst. Math. Applic. 20*, 409–14.

J.J. Du Croz and N.J. Higham (1992). "Stability of Methods for Matrix Inversion," *IMA J. Numer. Anal. 12*, 1–19.

J.M. Banoczi, N.C. Chiu, G.E. Cho, and I.C.F. Ipsen (1998). "The Lack of Influence of the Right-Hand Side on the Accuracy of Linear System Solution," *SIAM J. Sci. Comput. 20*, 203–227.

P. Amodio and F. Mazzia (1999). "A New Approach to Backward Error Analysis of LU Factorization *BIT 39*, 385–402.

An interesting account of von Neuman's contributions to the numerical analysis of Gaussian elimination is detailed in:

J.F. Grcar (2011). "John von Neuman's Analysis of Gaussian Elimination and the Origins of Modern Numerical Analysis," *SIAM Review 53*, 607–682.

3.4 Pivoting

The analysis in the previous section shows that we must take steps to ensure that no large entries appear in the computed triangular factors $\hat{L}$ and $\hat{U}$. The example

$$A = \begin{bmatrix} .0001 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 10000 & 1 \end{bmatrix} \begin{bmatrix} .0001 & 1 \\ 0 & -9999 \end{bmatrix} = LU$$

correctly identifies the source of the difficulty: relatively small pivots. A way out of this difficulty is to interchange rows. For example, if P is the permutation

$$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

then

$$PA = \begin{bmatrix} 1 & 1 \\ .0001 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ .0001 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & .9999 \end{bmatrix} = LU.$$

Observe that the triangular factors have modestly sized entries.

In this section we show how to determine a permuted version of A that has a reasonably stable LU factorization. There are several ways to do this and they each corresponds to a different pivoting strategy. Partial pivoting, complete pivoting, and rook pivoting are considered. The efficient implementation of these strategies and their properties are discussed. We begin with a few comments about permutation matrices that can be used to swap rows or columns.

3.4.1 Interchange Permutations

The stabilizations of Gaussian elimination that are developed in this section involve data movements such as the interchange of two matrix rows. In keeping with our desire to describe all computations in "matrix terms," we use permutation matrices to describe this process. (Now is a good time to review §1.2.8–§1.2.11.) *Interchange permutations* are particularly important. These are permutations obtained by swapping two rows in the identity, e.g.,

$$\Pi = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

Interchange permutations can be used to describe row and column swapping. If $A \in \mathbb{R}^{4\times 4}$, then $\Pi \cdot A$ is A with rows 1 and 4 interchanged while $A \cdot \Pi$ is A with columns 1 and 4 swapped.

If $P = \Pi_m \cdots \Pi_1$ and each Π_k is the identity with rows k and $piv(k)$ interchanged, then $piv(1{:}m)$ encodes P. Indeed, $x \in \mathbb{R}^n$ can be overwritten by Px as follows:

for $k = 1{:}m$
  $x(k) \leftrightarrow x(piv(k))$
end

Here, the "$\leftrightarrow$" notation means "swap contents." Since each Π_k is symmetric, we have $P^T = \Pi_1 \cdots \Pi_m$. Thus, the *piv* representation can also be used to overwrite x with $P^T x$:

for $k = m{:}-1{:}1$
  $x(k) \leftrightarrow x(piv(k))$
end

We remind the reader that although no floating point arithmetic is involved in a permutation operation, permutations move data and have a nontrivial effect upon performance.

3.4.2 Partial Pivoting

Interchange permutations can be used in LU computations to guarantee that no multiplier is greater than 1 in absolute value. Suppose

$$A = \begin{bmatrix} 3 & 17 & 10 \\ 2 & 4 & -2 \\ 6 & 18 & -12 \end{bmatrix}.$$

To get the smallest possible multipliers in the first Gauss transformation, we need a_{11} to be the largest entry in the first column. Thus, if Π_1 is the interchange permutation

$$\Pi_1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

then

$$\Pi_1 A = \begin{bmatrix} 6 & 18 & -12 \\ 2 & 4 & -2 \\ 3 & 17 & 10 \end{bmatrix}.$$

It follows that

$$M_1 = \begin{bmatrix} 1 & 0 & 0 \\ -1/3 & 1 & 0 \\ -1/2 & 0 & 1 \end{bmatrix} \quad \Longrightarrow \quad M_1\Pi_1 A = \begin{bmatrix} 6 & 18 & -12 \\ 0 & -2 & 2 \\ 0 & 8 & 16 \end{bmatrix}.$$

To obtain the smallest possible multiplier in M_2, we need to swap rows 2 and 3. Thus, if

$$\Pi_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{and} \quad M_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1/4 & 1 \end{bmatrix},$$

then

$$M_2\Pi_2 M_1\Pi_1 A = \begin{bmatrix} 6 & 18 & -12 \\ 0 & 8 & 16 \\ 0 & 0 & 6 \end{bmatrix}.$$

For general n we have

for $k = 1{:}n-1$

Find an interchange permutation $\Pi_k \in \mathbb{R}^{n\times n}$ that swaps

$A(k,k)$ with the largest element in $|A(k{:}n,k)|$.

$A = \Pi_k A$ (3.4.1)

Determine the Gauss transformation $M_k = I_n - \tau^{(k)} e_k^T$ such that if

v is the kth column of $M_k A$, then $v(k+1{:}n) = 0$.

$A = M_k A$

end

This particular row interchange strategy is called *partial pivoting* and upon completion, we have

$$M_{n-1}\Pi_{n-1}\cdots M_1\Pi_1 A = U \tag{3.4.2}$$

where U is upper triangular. As a consequence of the partial pivoting, no multiplier is larger than one in absolute value.

3.4.3 Where is L?

It turns out that (3.4.1) computes the factorization

$$PA = LU \tag{3.4.3}$$

where $P = \Pi_{n-1} \cdots \Pi_1$, U is upper triangular, and L is unit lower triangular with $|\ell_{ij}| \leq 1$. We show that $L(k+1{:}n, k)$ is a permuted version of M_k's multipliers. From (3.4.2) it can be shown that

$$\tilde{M}_{n-1} \cdots \tilde{M}_1 PA = U \tag{3.4.4}$$

where

$$\tilde{M}_k = (\Pi_{n-1} \cdots \Pi_{k+1}) M_k (\Pi_{k+1} \cdots \Pi_{n-1}) \tag{3.4.5}$$

for $k = 1{:}n-1$. For example, in the $n = 4$ case we have

$$\tilde{M}_3 \tilde{M}_2 \tilde{M}_1 PA = M_3 \cdot (\Pi_3 M_2 \Pi_3) \cdot (\Pi_3 \Pi_2 M_1 \Pi_2 \Pi_3) \cdot (\Pi_3 \Pi_2 \Pi_1) A$$

since the Π_i are symmetric. Moreover,

$$\tilde{M}_k = (\Pi_{n-1} \cdots \Pi_{k+1}) \cdot (I_n - \tau^{(k)} e_k^T) \cdot (\Pi_{k+1} \cdots \Pi_{n-1}) = I_n - \tilde{\tau}^{(k)} e_k^T$$

with $\tilde{\tau}^{(k)} = \Pi_{n-1} \cdots \Pi_{k+1} \tau^{(k)}$. This shows that $\tilde{M}_k$ is a Gauss transformation. The transformation from $\tau^{(k)}$ to $\tilde{\tau}^{(k)}$ is easy to implement in practice.

Algorithm 3.4.1 (Outer Product LU with Partial Pivoting) This algorithm computes the factorization $PA = LU$ where P is a permutation matrix encoded by $piv(1{:}n-1)$, L is unit lower triangular with $|\ell_{ij}| \leq 1$, and U is upper triangular. For $i = 1{:}n$, $A(i, i{:}n)$ is overwritten by $U(i, i{:}n)$ and $A(i+1{:}n, i)$ is overwritten by $L(i+1{:}n, i)$. The permutation P is given by $P = \Pi_{n-1} \cdots \Pi_1$ where Π_k is an interchange permutation obtained by swapping rows k and $piv(k)$ of I_n.

```
for k = 1:n − 1
    Determine μ with k ≤ μ ≤ n so |A(μ, k)| = ‖ A(k:n, k) ‖∞
    piv(k) = μ
    A(k, :) ↔ A(μ, :)
    if A(k, k) ≠ 0
        ρ = k + 1:n
        A(ρ, k) = A(ρ, k)/A(k, k)
        A(ρ, ρ) = A(ρ, ρ) − A(ρ, k)A(k, ρ)
    end
end
```

The *floating point* overhead associated with partial pivoting is minimal from the standpoint of arithmetic as there are only $O(n^2)$ comparisons associated with the search for the pivots. The overall algorithm involves $2n^3/3$ flops.

If Algorithm 3.4.1 is applied to

$$A = \begin{bmatrix} 3 & 17 & 10 \\ 2 & 4 & -2 \\ 6 & 18 & -12 \end{bmatrix},$$

then upon completion

$$A = \begin{bmatrix} 6 & 18 & -12 \\ 1/2 & 8 & 16 \\ 1/3 & -1/4 & 6 \end{bmatrix}$$

and $piv = [3, 3]$. These two quantities encode all the information associated with the reduction:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} A = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 1/3 & -1/4 & 1 \end{bmatrix} \begin{bmatrix} 6 & 18 & -12 \\ 0 & 8 & 16 \\ 0 & 0 & 6 \end{bmatrix}.$$

To compute the solution to $Ax = b$ after invoking Algorithm 3.4.1, we solve $Ly = Pb$ for y and $Ux = y$ for x. Note that b can be overwritten by Pb as follows

for $k = 1{:}n-1$
  $b(k) \leftrightarrow b(piv(k))$
end

We mention that if Algorithm 3.4.1 is applied to the problem,

$$\begin{bmatrix} .001 & 1.00 \\ 1.00 & 2.00 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1.00 \\ 3.00 \end{bmatrix},$$

using 3-digit floating point arithmetic, then

$$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \hat{L} = \begin{bmatrix} 1.00 & 0 \\ .001 & 1.00 \end{bmatrix}, \quad \hat{U} = \begin{bmatrix} 1.00 & 2.00 \\ 0 & 1.00 \end{bmatrix},$$

and $\hat{x} = [1.00, .996]^T$. Recall from §3.3.2 that if Gaussian elimination without pivoting is applied to this problem, then the computed solution has $O(1)$ error.

We mention that Algorithm 3.4.1 *always* runs to completion. If $A(k{:}n, k) = 0$ in step k, then $M_k = I_n$.

3.4.4 The Gaxpy Version

In §3.2 we developed outer product and gaxpy schemes for computing the LU factorization. Having just incorporated pivoting in the outer product version, it is equally straight forward to do the same with the gaxpy approach. Referring to Algorithm 3.2.2, we simply search the vector $|v(j{:}n)|$ in that algorithm for its maximal element and proceed accordingly.

Algorithm 3.4.2 (Gaxpy LU with Partial Pivoting) This algorithm computes the factorization $PA = LU$ where P is a permutation matrix encoded by $piv(1{:}n-1)$, L is unit lower triangular with $|\ell_{ij}| \leq 1$, and U is upper triangular. For $i = 1{:}n$, $A(i, i{:}n)$ is overwritten by $U(i, i{:}n)$ and $A(i+1{:}n, i)$ is overwritten by $L(i+1{:}n, i)$. The permutation P is given by $P = \Pi_{n-1} \cdots \Pi_1$ where Π_k is an interchange permutation obtained by swapping rows k and $piv(k)$ of I_n.

Initialize L to the identity and U to the zero matrix.
for $j = 1{:}n$
  if $j = 1$
    $v = A(:, 1)$
  else
    $\tilde{a} = \Pi_{j-1} \cdots \Pi_1 A(:, j)$
    Solve $L(1{:}j-1, 1{:}j-1)z = \tilde{a}(1{:}j-1)$ for $z \in \mathbb{R}^{j-1}$
    $U(1{:}j-1, j) = z,\ v(j{:}n) = \tilde{a}(j{:}n) - L(j{:}n, 1{:}j-1) \cdot z$
  end
  Determine μ with $j \leq \mu \leq n$ so $|v(\mu)| = \| v(j{:}n) \|_\infty$ and set $piv(j) = \mu$
  $v(j) \leftrightarrow v(\mu),\ L(j, 1{:}j-1) \leftrightarrow L(\mu, 1{:}j-1),\ U(j, j) = v(j)$
  if $v(j) \neq 0$
    $L(j+1{:}n, j) = v(j+1{:}n)/v(j)$
  end
end

As with Algorithm 3.4.1, this procedure requires $2n^3/3$ flops and $O(n^2)$ comparisons.

3.4.5 Error Analysis and the Growth Factor

We now examine the stability that is obtained with partial pivoting. This requires an accounting of the rounding errors that are sustained during elimination and during the triangular system solving. Bearing in mind that there are no rounding errors associated with permutation, it is not hard to show using Theorem 3.3.2 that the computed solution $\hat{x}$ satisfies $(A + E)\hat{x} = b$ where

$$|E| \leq n\mathbf{u}\left(2|A| + 4\hat{P}^T|\hat{L}||\hat{U}|\right) + O(\mathbf{u}^2). \tag{3.4.6}$$

Here we are assuming that $\hat{P}$, $\hat{L}$, and $\hat{U}$ are the computed analogs of P, L, and U as produced by the above algorithms. Pivoting implies that the elements of $\hat{L}$ are bounded by one. Thus $\| \hat{L} \|_\infty \leq n$ and we obtain the bound

$$\| E \|_\infty \leq n\mathbf{u}\left(2\| A \|_\infty + 4n\| \hat{U} \|_\infty\right) + O(\mathbf{u}^2). \tag{3.4.7}$$

The problem now is to bound $\| \hat{U} \|_\infty$. Define the *growth factor* ρ by

$$\rho = \max_{i,j,k} \frac{|\hat{a}_{ij}^{(k)}|}{\| A \|_\infty} \tag{3.4.8}$$

where $\hat{A}^{(k)}$ is the computed version of the matrix $A^{(k)} = M_k\Pi_k \cdots M_1\Pi_1 A$. It follows that

$$\| E \|_\infty \leq 6n^3\rho \| A \|_\infty \mathbf{u} + O(\mathbf{u}^2). \tag{3.4.9}$$

Whether or not this compares favorably with the ideal bound (2.7.20) hinges upon the size of the growth factor of ρ. (The factor n^3 is not an operating factor in practice and may be ignored in this discussion.)

The growth factor measures how large the A-entries become during the process of elimination. Whether or not we regard Gaussian elimination with partial pivoting is safe to use depends upon what we can say about this quantity. From an average-case point of view, experiments by Trefethen and Schreiber (1990) suggest that ρ is usually in the vicinity of $n^{2/3}$. However, from the worst-case point of view, ρ can be as large as 2^{n-1}. In particular, if $A \in \mathbb{R}^{n\times n}$ is defined by

$$a_{ij} = \begin{cases} 1 & \text{if } i = j \text{ or } j = n, \\ -1 & \text{if } i > j, \\ 0 & \text{otherwise,} \end{cases}$$

then there is no swapping of rows during Gaussian elimination with partial pivoting. We emerge with $A = LU$ and it can be shown that $u_{nn} = 2^{n-1}$. For example,

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

Understanding the behavior of ρ requires an intuition about what makes the U-factor large. Since $PA = LU$ implies $U = L^{-1}PA$ it would appear that the size of L^{-1} is relevant. However, Stewart (1997) discusses why one can expect the L-factor to be well conditioned.

Although there is still more to understand about ρ, the consensus is that serious element growth in Gaussian elimination with partial pivoting is *extremely* rare. *The method can be used with confidence.*

3.4.6 Complete Pivoting

Another pivot strategy called *complete pivoting* has the property that the associated growth factor bound is considerably smaller than 2^{n-1}. Recall that in partial pivoting, the kth pivot is determined by scanning the current subcolumn $A(k{:}n, k)$. In complete pivoting, the largest entry in the current submatrix $A(k{:}n, k{:}n)$ is permuted into the (k, k) position. Thus, we compute the upper triangularization

$$M_{n-1}\Pi_{n-1} \cdots M_1\Pi_1 A\Gamma_1 \cdots \Gamma_{n-1} = U.$$

In step k we are confronted with the matrix

$$A^{(k-1)} = M_{k-1}\Pi_{k-1} \cdots M_1\Pi_1 A\Gamma_1 \cdots \Gamma_{k-1}$$

and determine interchange permutations Π_k and Γ_k such that

$$\left|\left(\Pi_k A^{(k-1)}\Gamma_k\right)_{kk}\right| = \max_{k\leq i,j\leq n} \left|\left(\Pi_k A^{(k-1)}\Gamma_k\right)_{ij}\right|.$$

Algorithm 3.4.3 (Outer Product LU with Complete Pivoting) This algorithm computes the factorization $PAQ^T = LU$ where P is a permutation matrix encoded by $piv(1{:}n-1)$, Q is a permutation matrix encoded by $colpiv(1{:}n-1)$, L is unit lower triangular with $|\ell_{ij}| \leq 1$, and U is upper triangular. For $i = 1{:}n$, $A(i, i{:}n)$ is overwritten by $U(i, i{:}n)$ and $A(i+1{:}n, i)$ is overwritten by $L(i+1{:}n, i)$. The permutation P is given by $P = \Pi_{n-1} \cdots \Pi_1$ where Π_k is an interchange permutation obtained by swapping rows k and $rowpiv(k)$ of I_n. The permutation Q is given by $Q = \Gamma_{n-1} \cdots \Gamma_1$ where Γ_k is an interchange permutation obtained by swapping rows k and $colpiv(k)$ of I_n.

```
for k = 1:n − 1
    Determine μ with k ≤ μ ≤ n and λ with k ≤ λ ≤ n so
        |A(μ, λ)| = max{ |A(i, j)| : i = k:n, j = k:n }
    rowpiv(k) = μ
    A(k, 1:n) ↔ A(μ, 1:n)
    colpiv(k) = λ
    A(1:n, k) ↔ A(1:n, λ)
    if A(k, k) ≠ 0
        ρ = k + 1:n
        A(ρ, k) = A(ρ, k)/A(k, k)
        A(ρ, ρ) = A(ρ, ρ) − A(ρ, k)A(k, ρ)
    end
end
```

This algorithm requires $2n^3/3$ flops and $O(n^3)$ comparisons. Unlike partial pivoting, complete pivoting involves a significant floating point arithmetic overhead because of the two-dimensional search at each stage.

With the factorization $PAQ^T = LU$ in hand the solution to $Ax = b$ proceeds as follows:

Step 1. Solve $Lz = Pb$ for z.

Step 2. Solve $Uy = z$ for y.

Step 3. Set $x = Q^T y$.

The *rowpiv* and *colpiv* representations can be used to form Pb and Qy, respectively.

Wilkinson (1961) has shown that in exact arithmetic the elements of the matrix $A^{(k)} = M_k \Pi_k \cdots M_1 \Pi_1 A \Gamma_1 \cdots \Gamma_k$ satisfy

$$|a_{ij}^{(k)}| \leq k^{1/2}(2 \cdot 3^{1/2} \cdots k^{1/k-1})^{1/2} \max |a_{ij}|. \tag{3.4.10}$$

The upper bound is a rather slow-growing function of k. This fact coupled with vast empirical evidence suggesting that ρ is always modestly sized (e.g, $\rho = 10$) permit us to conclude that *Gaussian elimination with complete pivoting is stable.* The method solves a nearby linear system $(A+E)\hat{x} = b$ in the sense of (2.7.21). However, in general there is little reason to choose complete pivoting over partial pivoting. A possible exception is when A is rank deficient. In principal, complete pivoting can be used to reveal the rank of a matrix. Suppose $\mathsf{rank}(A) = r < n$. It follows that at the beginning of step

$r+1$, $A(r+1{:}n, r+1{:}n) = 0$. This implies that $\Pi_k = \Gamma_k = M_k = I$ for $k = r+1{:}n$ and so the algorithm can be terminated after step r with the following factorization in hand:

$$PAQ^T = LU = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I_{n-r} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & 0 \end{bmatrix}.$$

Here, L_{11} and U_{11} are r-by-r and L_{21} and U_{12}^T are $(n-r)$-by-r. Thus, Gaussian elimination with complete pivoting can in principle be used to determine the rank of a matrix. Nevertheless, roundoff errors make the probability of encountering an exactly zero pivot remote. In practice one would have to "declare" A to have rank k if the pivot element in step $k+1$ was sufficiently small. The numerical rank determination problem is discussed in detail in §5.5.

3.4.7 Rook Pivoting

A third type of LU stablization strategy called *rook pivoting* provides an interesting alternative to partial pivoting and complete pivoting. As with complete pivoting, it computes the factorization $PAQ = LU$. However, instead of choosing as pivot the largest value in $|A(k{:}n, k{:}n)|$, it searches for an element of that submatrix that is maximal in both its row *and* column. Thus, if

$$A(k{:}n, k{:}n) = \begin{bmatrix} 24 & 36 & 13 & 61 \\ 42 & 67 & 72 & 50 \\ 38 & 11 & 36 & 43 \\ 52 & 37 & 48 & 16 \end{bmatrix},$$

then "72" would be identified by complete pivoting while "52," "72," or "61" would be acceptable with the rook pivoting strategy. To implement rook pivoting, the scan-and-swap portion of Algorithm 3.4.3 is changed to

$\mu = k$, $\lambda = k$, $\tau = |a_{\mu\lambda}|$, $s = 0$
while $\tau < \| (A(k{:}n, \lambda) \|_\infty \quad \vee \quad \tau < \| (A(\mu, k{:}n) \|_\infty$
  if $\mathrm{mod}(s, 2) = 0$
    Update μ so that $|a_{\mu\lambda}| = \| (A(k{:}n, \lambda) \|_\infty$ with $k \le \mu \le n$.
  else
    Update λ so that $|a_{\mu\lambda}| = \| (A(\mu, k{:}n) \|_\infty$ with $k \le \lambda \le n$.
  end
  $s = s + 1$
end
$rowpiv(k) = \mu$, $A(k,:) \leftrightarrow A(\mu,:)$ $colpiv(k) = \lambda$, $A(:,k) \leftrightarrow A(:,\lambda)$

The search for a larger $|a_{\mu\lambda}|$ involves alternate scans of $A(k{:}n, \lambda)$ and $A(\mu, k{:}n)$. The value of τ is monotone increasing and that ensures termination of the **while**-loop. In theory, the exit value of s could be $O(n-k)^2)$, but in practice its value is $O(1)$. See Chang (2002). The bottom line is that rook pivoting represents the same $O(n^2)$ overhead as partial pivoting, but that it induces the same level of reliability as complete pivoting.

3.4.8 A Note on Underdetermined Systems

If $A \in \mathbb{R}^{m\times n}$ with $m < n$, rank$(A) = m$, and $b \in \mathbb{R}^m$, then the linear system $Ax = b$ is said to be *underdetermined.* Note that in this case there are an infinite number of solutions. With either complete or rook pivoting, it is possible to compute an LU factorization of the form

$$PAQ^T = L\,[\,U_1 \,|\, U_2\,] \tag{3.4.11}$$

where P and Q are permutations, $L \in \mathbb{R}^{m\times m}$ is unit lower triangular, and $U_1 \in \mathbb{R}^{m\times m}$ is nonsingular and upper triangular. Note that

$$Ax = b \;\Leftrightarrow\; (PAQ^T)(Qx) = (Pb) \;\Leftrightarrow\; L\,[\,U_1 \,|\, U_2\,]\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = L(U_1z_1 + U_2z_2) = c$$

where $c = Pb$ and

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = Qx.$$

This suggests the following solution procedure:

Step 1. Solve $Ly = Pb$ for $y \in \mathbb{R}^m$.

Step 2. Choose $z_2 \in \mathbb{R}^{n-m}$ and solve $U_1z_1 = y - U_2z_2$ for z_1.

Step 3. Set

$$x = Q^T\begin{bmatrix} z_1 \\ z_2 \end{bmatrix}.$$

Setting $z_2 = 0$ is a natural choice. We have more to say about underdetermined systems in §5.6.2.

3.4.9 The LU Mentality

We offer three examples that illustrate how to think in terms of the LU factorization when confronted with a linear equation situation.

Example 1. Suppose A is nonsingular and n-by-n and that B is n-by-p. Consider the problem of finding X (n-by-p) so $AX = B$. This is the *multiple right hand side problem.* If $X = [\,x_1 \,|\, \cdots \,|\, x_p\,]$ and $B = [\,b_1 \,|\, \cdots \,|\, b_p\,]$ are column partitions, then

```
Compute PA = LU
for k = 1:p
    Solve Ly = Pb_k and then Ux_k = y.
end
```
(3.4.12)

If $B = I_n$, then we emerge with an approximation to A^{-1} .

Example 2. Suppose we want to overwrite b with the solution to $A^kx = b$ where $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$, and k is a positive integer. One approach is to compute $C = A^k$ and then solve $Cx = b$. However, the matrix multiplications can be avoided altogether:

Compute $PA = LU$.
for $j = 1{:}k$
  Overwrite b with the solution to $Ly = Pb$. (3.4.13)
  Overwrite b with the solution to $Ux = b$.
end

As in Example 1, the idea is to get the LU factorization "outside the loop."

Example 3. Suppose we are given $A \in \mathbb{R}^{n\times n}$, $d \in \mathbb{R}^n$, and $c \in \mathbb{R}^n$ and that we want to compute $s = c^T A^{-1} d$. One approach is to compute $X = A^{-1}$ as discussed in (i) and then compute $s = c^T X d$. However, it is more economical to proceed as follows:

Compute $PA = LU$.
Solve $Ly = Pd$ and then $Ux = y$.
$s = c^T x$

An "A^{-1}" in a formula almost always means "solve a linear system" and almost never means "compute A^{-1}."

3.4.10 A Model Problem for Numerical Analysis

We are now in possession of a very important and well-understood algorithm (Gaussian elimination) for a very important and well-understood problem (linear equations). Let us take advantage of our position and formulate more abstractly what we mean by "problem sensitivity" and "algorithm stability." Our discussion follows Higham (ASNA, §1.5–1.6), Stewart (MA, §4.3), and Trefethen and Bau (NLA, Lectures 12, 14, 15, and 22).

A *problem* is a function $f{:}D \to S$ from "data/input space" D to "solution/output space" S. A *problem instance* is f together with a particular $d \in D$. We assume D and S are normed vector spaces. For linear systems, D is the set of matrix-vector pairs (A, b) where $A \in \mathbb{R}^{n\times n}$ is nonsingular and $b \in \mathbb{R}^n$. The function f maps (A, b) to $A^{-1}b$, an element of S. For a particular A and b, $Ax = b$ is a problem instance.

A *perturbation theory* for the problem f sheds light on the difference between $f(d)$ and $f(d + \Delta d)$ where $d \in D$ and $d + \Delta d \in D$. For linear systems, we discussed in §2.6 the difference between the solution to $Ax = b$ and the solution to $(A + \Delta A)(x + \Delta x) = (b + \Delta b)$. We bounded $\| \Delta x \|/\| x \|$ in terms of $\| \Delta A \|/\| A \|$ and $\| \Delta b \|/\| b \|$.

The *conditioning* of a problem refers to the behavior of f under perturbation at d. A *condition number* of a problem quantifies the rate of change of the solution with respect to the input data. If small changes in d induce relatively large changes in $f(d)$, then that problem instance is *ill-conditioned.* If small changes in d do not induce relatively large changes in $f(d)$, then that problem instance is *well-conditioned.* Definitions for "small" and "large" are required. For linear systems we showed in §2.6 that the magnitude of the condition number $\kappa(A) = \| A \| \| A^{-1} \|$ determines whether an $Ax = b$ problem is ill-conditioned or well-conditioned. One might say that a linear equation problem is well-conditioned if $\kappa(A) \approx O(1)$ and ill-conditioned if $\kappa(A) \approx O(1/\mathbf{u})$.

An *algorithm* for computing $f(d)$ produces an approximation $\tilde{f}(d)$. Depending on the situation, it may be necessary to identify a particular software implementation

of the underlying method. The $\tilde{f}$ function for Gaussian elimination with partial pivoting, Gaussian elimination with rook pivoting, and Gaussian elimination with complete pivoting are all different.

An algorithm for computing $f(d)$ is *stable* if for some small Δd, the computed solution $\tilde{f}(d)$ is close to $f(d+\Delta d)$. A stable algorithm *nearly* solves a nearby problem. An algorithm for computing $f(d)$ is *backward stable* if for some small Δd, the computed solution $\tilde{f}(d)$ satisfies $\tilde{f}(d) = f(d+\Delta d)$. A backward stable algorithm *exactly* solves a nearby problem. Applied to a given linear system $Ax = b$, Gaussian elimination with complete pivoting is backward stable because the computed solution $\tilde{x}$ satisfies

$$(A + \Delta)\tilde{x} = b$$

and $\| \Delta \|/\| A \| \approx O(\mathbf{u})$. On the other hand, if b is specified by a matrix-vector product $b = Mv$, then

$$(A + \Delta)\tilde{x} = Mv + \delta$$

where $\| \Delta \|/\| A \| \approx O(\mathbf{u})$ and $\delta/(\| M \| \| v \|) \approx O(\mathbf{u})$. Here, the underlying f is defined by $f{:}(A, M, v) \rightarrow A^{-1}(Mv)$. In this case the algorithm is stable but not backward stable.

Problems

P3.4.1 Let $A = LU$ be the LU factorization of n-by-n A with $|\ell_{ij}| \leq 1$. Let a_i^T and u_i^T denote the ith rows of A and U, respectively. Verify the equation

$$u_i^T = a_i^T - \sum_{j=1}^{i-1} \ell_{ij} u_j^T$$

and use it to show that $\| U \|_\infty \leq 2^{n-1} \| A \|_\infty$. (Hint: Take norms and use induction.)

P3.4.2 Show that if $PAQ = \text{LU}$ is obtained via Gaussian elimination with complete pivoting, then no element of $U(i, i{:}n)$ is larger in absolute value than $|u_{ii}|$. Is this true with rook pivoting?

P3.4.3 Suppose $A \in \mathbb{R}^{n \times n}$ has an LU factorization and that L and U are known. Give an algorithm which can compute the (i, j) entry of A^{-1} in approximately $(n-j)^2 + (n-i)^2$ flops.

P3.4.4 Suppose $\hat{X}$ is the computed inverse obtained via (3.4.12). Give an upper bound for $\| A\hat{X} - I \|_F$.

P3.4.5 Extend Algorithm 3.4.3 so that it can produce the factorization (3.4.11). How many flops are required?

Notes and References for §3.4

Papers concerned with element growth and pivoting include:

C.W. Cryer (1968). "Pivot Size in Gaussian Elimination," *Numer. Math. 12,* 335–345.

J.K. Reid (1971). "A Note on the Stability of Gaussian Elimination," *J.Inst. Math. Applic. 8,* 374–375.

P.A. Businger (1971). "Monitoring the Numerical Stability of Gaussian Elimination," *Numer. Math. 16,* 360–361.

A.M. Cohen (1974). "A Note on Pivot Size in Gaussian Elimination," *Lin. Alg. Applic. 8,* 361–68.

A.M. Erisman and J.K. Reid (1974). "Monitoring the Stability of the Triangular Factorization of a Sparse Matrix," *Numer. Math. 22,* 183–186.

J. Day and B. Peterson (1988). "Growth in Gaussian Elimination," *Amer. Math. Monthly 95,* 489–513.

N.J. Higham and D.J. Higham (1989). "Large Growth Factors in Gaussian Elimination with Pivoting," *SIAM J. Matrix Anal. Applic. 10,* 155–164.

L.N. Trefethen and R.S. Schreiber (1990). "Average-Case Stability of Gaussian Elimination," *SIAM J. Matrix Anal. Applic. 11,* 335–360.

N. Gould (1991). "On Growth in Gaussian Elimination with Complete Pivoting," *SIAM J. Matrix Anal. Applic. 12*, 354–361.

A. Edelman (1992). "The Complete Pivoting Conjecture for Gaussian Elimination is False," *Mathematica J. 2*, 58–61.

S.J. Wright (1993). "A Collection of Problems for Which Gaussian Elimination with Partial Pivoting is Unstable," *SIAM J. Sci. Stat. Comput. 14*, 231–238.

L.V. Foster (1994). "Gaussian Elimination with Partial Pivoting Can Fail in Practice," *SIAM J. Matrix Anal. Applic. 15*, 1354–1362.

A. Edelman and W. Mascarenhas (1995). "On the Complete Pivoting Conjecture for a Hadamard Matrix of Order 12," *Lin. Multilin. Alg. 38*, 181–185.

J.M. Pena (1996). "Pivoting Strategies Leading to Small Bounds of the Errors for Certain Linear Systems," *IMA J. Numer. Anal. 16*, 141–153.

J.L. Barlow and H. Zha (1998). "Growth in Gaussian Elimination, Orthogonal Matrices, and the 2-Norm," *SIAM J. Matrix Anal. Applic. 19*, 807–815.

P. Favati, M. Leoncini, and A. Martinez (2000). "On the Robustness of Gaussian Elimination with Partial Pivoting," *BIT 40*, 62–73.

As we mentioned, the size of L^{-1} is relevant to the growth factor. Thus, it is important to have an understanding of triangular matrix condition, see:

D. Viswanath and L.N. Trefethen (1998). "Condition Numbers of Random Triangular Matrices," *SIAM J. Matrix Anal. Applic. 19*, 564–581.

The connection between small pivots and near singularity is reviewed in:

T.F. Chan (1985). "On the Existence and Computation of LU Factorizations with Small Pivots," *Math. Comput. 42*, 535–548.

A pivot strategy that we did not discuss is *pairwise pivoting*. In this approach, 2-by-2 Gauss transformations are used to zero the lower triangular portion of A. The technique is appealing in certain multiprocessor environments because only adjacent rows are combined in each step, see:

D. Sorensen (1985). "Analysis of Pairwise Pivoting in Gaussian Elimination," *IEEE Trans. Comput. C-34*, 274–278.

A related type of pivoting called *tournament pivoting* that is of interest in distributed memory computing is outlined in §3.6.3. For a discussion of rook pivoting and its properties, see:

L.V. Foster (1997). "The Growth Factor and Efficiency of Gaussian Elimination with Rook Pivoting," *J. Comput. Appl. Math., 86*, 177–194.

G. Poole and L. Neal (2000). "The Rook's Pivoting Strategy," *J. Comput. Appl. Math. 123*, 353–369.

X-W Chang (2002) "Some Features of Gaussian Elimination with Rook Pivoting," *BIT 42*, 66–83.

3.5 Improving and Estimating Accuracy

Suppose we apply Gaussian elimination with partial pivoting to the n-by-n system $Ax = b$ and that IEEE double precision arithmetic is used. Equation (3.4.9) essentially says that if the growth factor is modest then the computed solution $\hat{x}$ satisfies

$$(A + E)\hat{x} = b, \qquad \| E \|_\infty \approx \mathbf{u} \| A \|_\infty. \tag{3.5.1}$$

In this section we explore the practical ramifications of this result. We begin by stressing the distinction that should be made between residual size and accuracy. This is followed by a discussion of scaling, iterative improvement, and condition estimation. See Higham (ASNA) for a more detailed treatment of these topics.

We make two notational remarks at the outset. The infinity norm is used throughout since it is very handy in roundoff error analysis and in practical error estimation. Second, whenever we refer to "Gaussian elimination" in this section we really mean Gaussian elimination with some stabilizing pivot strategy such as partial pivoting.

3.5.1 Residual Size versus Accuracy

The *residual* of a computed solution $\hat{x}$ to the linear system $Ax = b$ is the vector $b - A\hat{x}$. A small residual means that $A\hat{x}$ effectively "predicts" the right hand side b. From Equation 3.5.1 we have $\| b - A\hat{x} \|_\infty \approx \mathbf{u} \| A \|_\infty \| \hat{x} \|_\infty$ and so we obtain

Heuristic I. *Gaussian elimination produces a solution $\hat{x}$ with a relatively small residual.*

Small residuals do not imply high accuracy. Combining Theorem 2.6.2 and (3.5.1), we see that

$$\frac{\| \hat{x} - x \|_\infty}{\| x \|_\infty} \approx \mathbf{u}\kappa_\infty(A) . \tag{3.5.2}$$

This justifies a second guiding principle.

Heuristic II. *If the unit roundoff and condition satisfy $\mathbf{u} \approx 10^{-d}$ and $\kappa_\infty(A) \approx 10^q$, then Gaussian elimination produces a solution $\hat{x}$ that has about $d - q$ correct decimal digits.*

If $\mathbf{u}\kappa_\infty(A)$ is large, then we say that A is ill-conditioned with respect to the machine precision.

As an illustration of the Heuristics I and II, consider the system

$$\begin{bmatrix} .986 & .579 \\ .409 & .237 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} .235 \\ .107 \end{bmatrix}$$

in which $\kappa_\infty(A) \approx 700$ and $x = [2, -3]^T$. Here is what we find for various machine precisions:

$\mathbf{u}$	$\hat{x}_1$	$\hat{x}_2$	$\frac{\| \hat{x} - x \|_\infty}{\| x \|_\infty}$	$\frac{\| b - A\hat{x} \|_\infty}{\| A \|_\infty \| \hat{x} \|_\infty}$
10^{-3}	2.11	−3.17	$5 \cdot 10^{-2}$	$2.0 \cdot 10^{-3}$
10^{-4}	1.986	−2.975	$8 \cdot 10^{-3}$	$1.5 \cdot 10^{-4}$
10^{-5}	2.0019	−3.0032	$1 \cdot 10^{-3}$	$2.1 \cdot 10^{-6}$
10^{-6}	2.00025	−3.00094	$3 \cdot 10^{-4}$	$4.2 \cdot 10^{-7}$

Whether or not to be content with the computed solution $\hat{x}$ depends on the requirements of the underlying source problem. In many applications accuracy is not important but small residuals are. In such a situation, the $\hat{x}$ produced by Gaussian elimination is probably adequate. On the other hand, if the number of correct digits in $\hat{x}$ is an issue, then the situation is more complicated and the discussion in the remainder of this section is relevant.

3.5.2 Scaling

Let β be the machine base (typically $\beta = 2$) and define the diagonal matrices $D_1 = \text{diag}(\beta^{r_1}, \ldots, \beta^{r_n})$ and $D_2 = \text{diag}(\beta^{c_1}, \ldots, \beta^{c_n})$. The solution to the n-by-n linear system $Ax = b$ can be found by solving the *scaled system* $(D_1^{-1} A D_2)y = D_1^{-1} b$ using

Gaussian elimination and then setting $x = D_2 y$. The scalings of A, b, and y require only $O(n^2)$ flops and may be accomplished without roundoff. Note that D_1 scales equations and D_2 scales unknowns.

It follows from Heuristic II that if $\hat{x}$ and $\hat{y}$ are the computed versions of x and y, then

$$\frac{\| D_2^{-1}(\hat{x} - x) \|_\infty}{\| D_2^{-1} x \|_\infty} = \frac{\| \hat{y} - y \|_\infty}{\| y \|_\infty} \approx \mathbf{u}\kappa_\infty(D_1^{-1} A D_2). \tag{3.5.3}$$

Thus, if $\kappa_\infty(D_1^{-1}AD_2)$ can be made considerably smaller than $\kappa_\infty(A)$, then we might expect a correspondingly more accurate $\hat{x}$, provided errors are measured in the "D_2" norm defined by $\| z \|_{D_2} = \| D_2^{-1} z \|_\infty$. This is the objective of scaling. Note that it encompasses two issues: the condition of the scaled problem and the appropriateness of appraising error in the D_2-norm.

An interesting but very difficult mathematical problem concerns the exact minimization of $\kappa_p(D_1^{-1}AD_2)$ for general diagonal D_i and various p. Such results as there are in this direction are not very practical. This is hardly discouraging, however, when we recall that (3.5.3) is a heuristic result, it makes little sense to minimize exactly a heuristic bound. What we seek is a fast, approximate method for improving the quality of the computed solution $\hat{x}$.

One technique of this variety is *simple row scaling*. In this scheme D_2 is the identity and D_1 is chosen so that each row in $D_1^{-1}A$ has approximately the same ∞-norm. Row scaling reduces the likelihood of adding a very small number to a very large number during elimination—an event that can greatly diminish accuracy.

Slightly more complicated than simple row scaling is *row-column equilibration*. Here, the object is to choose D_1 and D_2 so that the ∞-norm of each row and column of $D_1^{-1}AD_2$ belongs to the interval $[1/\beta, 1]$ where β is the base of the floating point system. For work along these lines, see McKeeman (1962).

It cannot be stressed too much that simple row scaling and row-column equilibration do not "solve" the scaling problem. Indeed, either technique can render a worse $\hat{x}$ than if no scaling whatever is used. The ramifications of this point are thoroughly discussed in Forsythe and Moler (SLE, Chap. 11). The basic recommendation is that the scaling of equations and unknowns must proceed on a problem-by-problem basis. General scaling strategies are unreliable. It is best to scale (if at all) on the basis of what the source problem proclaims about the significance of each a_{ij}. Measurement units and data error may have to be considered.

3.5.3 Iterative Improvement

Suppose $Ax = b$ has been solved via the partial pivoting factorization $PA = LU$ and that we wish to improve the accuracy of the computed solution $\hat{x}$. If we execute

$$\begin{aligned} &r = b - A\hat{x} \\ &\text{Solve } Ly = Pr. \\ &\text{Solve } Uz = y. \\ &x_{\text{new}} = \hat{x} + z \end{aligned} \tag{3.5.4}$$

then in exact arithmetic $Ax_{\text{new}} = A\hat{x} + Az = (b - r) + r = b$. Unfortunately, the naive floating point execution of these formulae renders an x_{new} that is no more accurate

than $\hat{x}$. This is to be expected since $\hat{r} = \mathrm{fl}(b - A\hat{x})$ has few, if any, correct significant digits. (Recall Heuristic I.) Consequently, $\hat{z} = \mathrm{fl}(A^{-1}r) \approx A^{-1} \cdot \text{noise} \approx \text{noise}$ is a very poor correction *from the standpoint of improving the accuracy of* $\hat{x}$. However, Skeel (1980) has an error analysis that indicates when (3.5.4) gives an improved x_{new} *from the standpoint of backward error.* In particular, if the quantity

$$\tau = \left(\| \, |A| \, |A^{-1}| \, \|_\infty \right) \left(\max_i (|A||x|)_i \, / \min_i (|A||x|)_i \right)$$

is not too big, then (3.5.4) produces an x_{new} such that $(A + E)x_{new} = b$ for very small E. Of course, if Gaussian elimination with partial pivoting is used, then the computed $\hat{x}$ already solves a nearby system. However, this may not be the case for certain pivot strategies used to preserve sparsity. In this situation, the *fixed precision iterative improvement* step (3.5.4) can be worthwhile and cheap. See Arioli, Demmel, and Duff (1988).

In general, for (3.5.4) to produce a more accurate x, it is necessary to compute the residual $b - A\hat{x}$ with extended precision floating point arithmetic. Typically, this means that if t-digit arithmetic is used to compute $PA = LU$, x, y, and z, then $2t$-digit arithmetic is used to form $b - A\hat{x}$. The process can be iterated. In particular, once we have computed $PA = LU$ and initialize $x = 0$, we repeat the following:

$$\begin{aligned} &r = b - Ax \text{ (higher precision)} \\ &\text{Solve } Ly = Pr \text{ for } y \text{ and } Uz = y \text{ for } z. \\ &x = x + z \end{aligned} \tag{3.5.5}$$

We refer to this process as *mixed-precision iterative improvement.* The original A must be used in the high-precision computation of r. The basic result concerning the performance of (3.5.5) is summarized in the following heuristic:

Heuristic III. *If the machine precision* $\mathbf{u}$ *and condition satisfy* $\mathbf{u} = 10^{-d}$ *and* $\kappa_\infty(A) \approx 10^q$, *then after* k *executions of (3.5.5),* x *has approximately* $\min\{d, k(d-q)\}$ *correct digits if the residual computation is performed with precision* $\mathbf{u}^2$.

Roughly speaking, if $\mathbf{u}\,\kappa_\infty(A) \leq 1$, then iterative improvement can ultimately produce a solution that is correct to full (single) precision. Note that the process is relatively cheap. Each improvement costs $O(n^2)$, to be compared with the original $O(n^3)$ investment in the factorization $PA = LU$. Of course, no improvement may result if A is badly conditioned with respect to the machine precision.

3.5.4 Condition Estimation

Suppose that we have solved $Ax = b$ via $PA = LU$ and that we now wish to ascertain the number of correct digits in the computed solution $\hat{x}$. It follows from Heuristic II that in order to do this we need an estimate of the condition $\kappa_\infty(A) = \| A \|_\infty \| A^{-1} \|_\infty$. Computing $\| A \|_\infty$ poses no problem as we merely use the $O(n^2)$ formula (2.3.10). The challenge is with respect to the factor $\| A^{-1} \|_\infty$. Conceivably, we could estimate this quantity by $\| \hat{X} \|_\infty$, where $\hat{X} = [\, \hat{x}_1 \,|\, \cdots \,|\, \hat{x}_n \,]$ and $\hat{x}_i$ is the computed solution to $Ax_i = e_i$. (See §3.4.9.) The trouble with this approach is its expense: $\hat{\kappa}_\infty = \| A \|_\infty \| \hat{X} \|_\infty$ costs about three times as much as $\hat{x}$.

The central problem of *condition estimation* is how to estimate reliably the condition number in $O(n^2)$ flops assuming the availability of $PA = LU$ or one of the factorizations that are presented in subsequent chapters. An approach described in Forsythe and Moler (SLE, p. 51) is based on iterative improvement and the heuristic

$$\mathbf{u}\kappa_\infty(A) \approx \| z \|_\infty / \| x \|_\infty$$

where z is the first correction of x in (3.5.5).

Cline, Moler, Stewart, and Wilkinson (1979) propose an approach to the condition estimation problem thatis based on the implication

$$Ay = d \quad \Longrightarrow \quad \| A^{-1} \|_\infty \geq \| y \|_\infty / \| d \|_\infty.$$

The idea behind their estimator is to choose d so that the solution y is large in norm and then set

$$\hat{\kappa}_\infty = \| A \|_\infty \| y \|_\infty / \| d \|_\infty.$$

The success of this method hinges on how close the ratio $\| y \|_\infty / \| d \|_\infty$ is to its maximum value $\| A^{-1} \|_\infty$.

Consider the case when $A = T$ is upper triangular. The relation between d and y is completely specified by the following column version of back substitution:

$$\begin{array}{l} p(1{:}n) = 0 \\ \textbf{for } k = n\text{:} -1\text{:}1 \\ \quad \text{Choose } d(k). \\ \quad y(k) = (d(k) - p(k))/T(k,k) \\ \quad p(1{:}k-1) = p(1{:}k-1) + y(k)T(1{:}k-1,k) \\ \textbf{end} \end{array} \tag{3.5.6}$$

Normally, we use this algorithm to solve a *given* triangular system $Ty = d$. However, in the condition estimation setting we are free to pick the right-hand side d subject to the "constraint" that y is large relative to d.

One way to encourage growth in y is to choose $d(k)$ from the set $\{-1, +1\}$ so as to maximize $y(k)$. If $p(k) \geq 0$, then set $d(k) = -1$. If $p(k) < 0$, then set $d(k) = +1$. In other words, (3.5.6) is invoked with $d(k) = -\text{sign}(p(k))$. Overall, the vector d has the form $d(1{:}n) = [\pm 1, \ldots, \pm 1]^T$. Since this is a unit vector, we obtain the estimate $\hat{\kappa}_\infty = \| T \|_\infty \| y \|_\infty$.

A more reliable estimator results if $d(k) \in \{-1, +1\}$ is chosen so as to encourage growth both in $y(k)$ and the running sum update $p(1{:}k-1,k) + T(1{:}k-1,k)y(k)$. In particular, at step k we compute

$$\begin{aligned} y(k)^+ &= (1 - p(k))/T(k,k), \\ s(k)^+ &= |y(k)^+| + \| p(1{:}k-1) + T(1{:}k-1,k)y(k)^+ \|_1, \\ y(k)^- &= (-1 - p(k))/T(k,k), \\ s(k)^- &= |y(k)^-| + \| p(1{:}k-1) + T(1{:}k-1,k)y(k)^- \|_1, \end{aligned}$$

and set

$$y(k) = \begin{cases} y(k)^+ & \text{if } s(k)^+ \geq s(k)^-, \\ y(k)^- & \text{if } s(k)^+ < s(k)^-. \end{cases}$$

This gives the following procedure.

Algorithm 3.5.1 (Condition Estimator) Let $T \in \mathbb{R}^{n\times n}$ be a nonsingular upper triangular matrix. This algorithm computes unit ∞-norm y and a scalar κ so $\| Ty \|_\infty \approx 1/\| T^{-1} \|_\infty$ and $\kappa \approx \kappa_\infty(T)$

$p(1{:}n) = 0$
for $k = n{:}-1{:}1$
  $y(k)^+ = (1 - p(k))/T(k,k)$
  $y(k)^- = (-1 - p(k))/T(k,k)$
  $p(k)^+ = p(1{:}k-1) + T(1{:}k-1,k)y(k)^+$
  $p(k)^- = p(1{:}k-1) + T(1{:}k-1,k)y(k)^-$
  if $|y(k)^+| + \| p(k)^+ \|_1 \geq |y(k)^-| + \| p(k)^- \|_1$
    $y(k) = y(k)^+$
    $p(1{:}k-1) = p(k)^+$
  else
    $y(k) = y(k)^-$
    $p(1{:}k-1) = p(k)^-$
  end
end
$\kappa = \| y \|_\infty \| T \|_\infty$
$y = y/\| y \|_\infty$

The algorithm involves several times the work of ordinary back substitution.

We are now in a position to describe a procedure for estimating the condition of a square nonsingular matrix A whose $PA = LU$ factorization is available:

Step 1. Apply the lower triangular version of Algorithm 3.5.1 to U^T and obtain a large-norm solution to $U^T y = d$.

Step 2. Solve the triangular systems $L^T r = y$, $Lw = Pr$, and $Uz = w$.

Step 3. Set $\hat{\kappa}_\infty = \| A \|_\infty \| z \|_\infty / \| r \|_\infty$.

Note that $\| z \|_\infty \leq \| A^{-1} \|_\infty \| r \|_\infty$. The method is based on several heuristics. First, if A is ill-conditioned and $PA = LU$, then it is usually the case that U is correspondingly ill-conditioned. The lower triangle L tends to be fairly well-conditioned. Thus, it is more profitable to apply the condition estimator to U than to L. The vector r, because it solves $A^T P^T r = d$, tends to be rich in the direction of the left singular vector associated with $\sigma_{\min}(A)$. Right-hand sides with this property render large solutions to the problem $Az = r$.

In practice, it is found that the condition estimation technique that we have outlined produces adequate order-of-magnitude estimates of the true condition number.

Problems

P3.5.1 Show by example that there may be more than one way to equilibrate a matrix.

P3.5.2 Suppose $P(A+E) = \hat{L}\hat{U}$, where P is a permutation, $\hat{L}$ is lower triangular with $|\hat{\ell}_{ij}| \leq 1$, and $\hat{U}$ is upper triangular. Show that $\hat{\kappa}_\infty(A) \geq \| A \|_\infty / (\| E \|_\infty + \mu)$ where $\mu = \min |\hat{u}_{ii}|$. Conclude that if a small pivot is encountered when Gaussian elimination with pivoting is applied to A, then A is ill-conditioned. The converse is not true. (Hint: Let A be the matrix B_n defined in (2.6.9)).

P3.5.3 (Kahan (1966)) The system $Ax = b$ where

$$A = \begin{bmatrix} 2 & -1 & 1 \\ -1 & 10^{-10} & 10^{-10} \\ 1 & 10^{-10} & 10^{-10} \end{bmatrix}, \qquad b = \begin{bmatrix} 2(1+10^{-10}) \\ -10^{-10} \\ 10^{-10} \end{bmatrix}$$

has solution $x = [10^{-10} \;\; -1 \;\; 1]^T$. (a) Show that if $(A+E)y = b$ and $|E| \leq 10^{-8}|A|$, then $|x-y| \leq 10^{-7}|x|$. That is, small relative changes in A's entries do not induce large changes in x even though $\kappa_\infty(A) = 10^{10}$. (b) Define $D = \text{diag}(10^{-5}, 10^5, 10^5)$. Show that $\kappa_\infty(DAD) \leq 5$. (c) Explain what is going on using Theorem 2.6.3.

P3.5.4 Consider the matrix:

$$T = \begin{bmatrix} 1 & 0 & M & -M \\ 0 & 1 & -M & M \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad M \in \mathbf{R}.$$

What estimate of $\kappa_\infty(T)$ is produced when (3.5.6) is applied with $d(k) = -\text{sign}(p(k))$? What estimate does Algorithm 3.5.1 produce? What is the true $\kappa_\infty(T)$?

P3.5.5 What does Algorithm 3.5.1 produce when applied to the matrix B_n given in (2.6.9)?

Notes and References for §3.5

The following papers are concerned with the scaling of $Ax = b$ problems:

F.L. Bauer (1963). "Optimally Scaled Matrices," *Numer. Math. 5,* 73–87.
P.A. Businger (1968). "Matrices Which Can Be Optimally Scaled," *Numer. Math. 12,* 346–48.
A. van der Sluis (1969). "Condition Numbers and Equilibration Matrices," *Numer. Math. 14,* 14–23.
A. van der Sluis (1970). "Condition, Equilibration, and Pivoting in Linear Algebraic Systems," *Numer. Math. 15,* 74–86.
C. McCarthy and G. Strang (1973). "Optimal Conditioning of Matrices," *SIAM J. Numer. Anal. 10,* 370–388.
T. Fenner and G. Loizou (1974). "Some New Bounds on the Condition Numbers of Optimally Scaled Matrices," *J. ACM 21,* 514–524.
G.H. Golub and J.M. Varah (1974). "On a Characterization of the Best L_2-Scaling of a Matrix," *SIAM J. Numer. Anal. 11,* 472–479.
R. Skeel (1979). "Scaling for Numerical Stability in Gaussian Elimination," *J. ACM 26,* 494–526.
R. Skeel (1981). "Effect of Equilibration on Residual Size for Partial Pivoting," *SIAM J. Numer. Anal. 18,* 449–55.
V. Balakrishnan and S. Boyd (1995). "Existence and Uniqueness of Optimal Matrix Scalings," *SIAM J. Matrix Anal. Applic. 16,* 29–39.

Part of the difficulty in scaling concerns the selection of a norm in which to measure errors. An interesting discussion of this frequently overlooked point appears in:

W. Kahan (1966). "Numerical Linear Algebra," *Canadian Math. Bull. 9,* 757–801.

For a rigorous analysis of iterative improvement and related matters, see:

C.B. Moler (1967). "Iterative Refinement in Floating Point," *J. ACM 14,* 316-371.
M. Jankowski and M. Wozniakowski (1977). "Iterative Refinement Implies Numerical Stability," *BIT 17,* 303–311.
R.D. Skeel (1980). "Iterative Refinement Implies Numerical Stability for Gaussian Elimination," *Math. Comput. 35,* 817–832.
N.J. Higham (1997). "Iterative Refinement for Linear Systems and LAPACK," *IMA J. Numer. Anal. 17,* 495–509.

A. Dax (2003). "A Modified Iterative Refinement Scheme," *SIAM J. Sci. Comput. 25*, 1199–1213.
J. Demmel, Y. Hida, W. Kahan, X.S. Li, S. Mukherjee, and E.J. Riedy (2006). "Error Bounds from Extra-Precise Iterative Refinement," *ACM Trans. Math. Softw. 32*, 325–351.

The condition estimator that we described is given in:

A.K. Cline, C.B. Moler, G.W. Stewart, and J.H. Wilkinson (1979). "An Estimate for the Condition Number of a Matrix," *SIAM J. Numer. Anal.* 16, 368-75.

Other references concerned with the condition estimation problem include:

C.G. Broyden (1973). "Some Condition Number Bounds for the Gaussian Elimination Process," *J. Inst. Math. Applic. 12*, 273–286.
F. Lemeire (1973). "Bounds for Condition Numbers of Triangular Value of a Matrix," *Lin. Alg. Applic. 11*, 1–2.
D.P. O'Leary (1980). "Estimating Matrix Condition Numbers," *SIAM J. Sci. Stat. Comput. 1*, 205–209.
A.K. Cline, A.R. Conn, and C. Van Loan (1982). "Generalizing the LINPACK Condition Estimator," in *Numerical Analysis* , J.P. Hennart (ed.), Lecture Notes in Mathematics No. 909, Springer-Verlag, New York.
A.K. Cline and R.K. Rew (1983). "A Set of Counter examples to Three Condition Number Estimators," *SIAM J. Sci. Stat. Comput. 4*, 602–611.
W. Hager (1984). "Condition Estimates," *SIAM J. Sci. Stat. Comput. 5*, 311–316.
N.J. Higham (1987). "A Survey of Condition Number Estimation for Triangular Matrices," *SIAM Review 29*, 575–596.
N.J. Higham (1988). "FORTRAN Codes for Estimating the One-Norm of a Real or Complex Matrix with Applications to Condition Estimation (Algorithm 674)," *ACM Trans. Math. Softw. 14*, 381–396.
C.H. Bischof (1990). "Incremental Condition Estimation," *SIAM J. Matrix Anal. Applic. 11*, 312–322.
G. Auchmuty (1991). "A Posteriori Error Estimates for Linear Equations," *Numer. Math. 61*, 1–6.
N.J. Higham (1993). "Optimization by Direct Search in Matrix Computations," *SIAM J. Matrix Anal. Applic. 14*, 317–333.
D.J. Higham (1995). "Condition Numbers and Their Condition Numbers," *Lin. Alg. Applic. 214*, 193–213.
G.W. Stewart (1997). "The Triangular Matrices of Gaussian Elimination and Related Decompositions," *IMA J. Numer. Anal. 17*, 7–16.

3.6 Parallel LU

In §3.2.11 we show how to organize a block version of Gaussian elimination (without pivoting) so that the overwhelming majority of flops occur in the context of matrix multiplication. It is possible to incorporate partial pivoting and maintain the same level-3 fraction. After stepping through the derivation we proceed to show how the process can be effectively parallelized using the block-cyclic distribution ideas that were presented in §1.6.

3.6.1 Block LU with Pivoting

Throughout this section assume $A \in \mathbb{R}^{n\times n}$ and for clarity that $n = rN$:

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1N} \\ \vdots & \ddots & \vdots \\ A_{N1} & \cdots & A_{NN} \end{bmatrix} \qquad A_{i,j} \in \mathbb{R}^{r\times r}. \tag{3.6.1}$$

We revisit Algorithm 3.2.4 (nonrecursive block LU) and show how to incorporate partial pivoting.

The first step starts by applying scalar Gaussian elimination with partial pivoting to the first block column. Using an obvious rectangular matrix version of Algorithm 3.4.1 we obtain the following factorization:

$$P_1 \begin{bmatrix} A_{11} \\ A_{21} \\ \vdots \\ A_{N1} \end{bmatrix} = \begin{bmatrix} L_{11} \\ L_{21} \\ \vdots \\ L_{N1} \end{bmatrix} U_{11}. \tag{3.6.2}$$

In this equation, $P_1 \in \mathbb{R}^{n\times n}$ is a permutation, $L_{11} \in \mathbb{R}^{r\times r}$ is unit lower triangular, and $U_{11} \in \mathbb{R}^{r\times r}$ is upper triangular.

The next task is to compute the first block row of U. To do this we set

$$P_1 A = \begin{bmatrix} \tilde{A}_{11} & \cdots & \tilde{A}_{1N} \\ \vdots & \ddots & \vdots \\ \tilde{A}_{N1} & \cdots & \tilde{A}_{NN} \end{bmatrix}, \qquad \tilde{A}_{i,j} \in \mathbb{R}^{r\times r}, \tag{3.6.3}$$

and solve the lower triangular multiple-right-hand-side problem

$$L_{11} \left[\begin{array}{c|c|c} U_{12} & \cdots & U_{1N} \end{array} \right] = \left[\begin{array}{c|c|c} \tilde{A}_{12} & \cdots & \tilde{A}_{1N} \end{array} \right] \tag{3.6.4}$$

for $U_{12}, \ldots, U_{1N} \in \mathbb{R}^{r\times r}$. At this stage it is easy to show that we have the partial factorization

$$P_1 A = \left[\begin{array}{c|ccc} L_{11} & 0 & \cdots & 0 \\ \hline L_{21} & I_r & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ L_{N1} & 0 & \cdots & I_r \end{array} \right] \left[\begin{array}{c|c} I_r & 0 \\ \hline 0 & A^{(\text{new})} \end{array} \right] \left[\begin{array}{c|ccc} U_{11} & U_{12} & \cdots & U_{1N} \\ \hline 0 & I_r & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & I_r \end{array} \right]$$

where

$$A^{(\text{new})} = \begin{bmatrix} \tilde{A}_{22} & \cdots & \tilde{A}_{2N} \\ \vdots & \ddots & \vdots \\ \tilde{A}_{N2} & \cdots & \tilde{A}_{NN} \end{bmatrix} - \begin{bmatrix} L_{21} \\ \vdots \\ L_{N1} \end{bmatrix} \left[\begin{array}{c|c|c} U_{12} & \cdots & U_{1N} \end{array} \right]. \tag{3.6.5}$$

Note that the computation of $A^{(\text{new})}$ is a level-3 operation as it involves one matrix multiplication per A-block.

The remaining task is to compute the pivoted LU factorization of $A^{(\text{new})}$. Indeed, if

$$P^{(\text{new})} A^{(\text{new})} = L^{(\text{new})} U^{(\text{new})}$$

and

$$P^{(\text{new})} \begin{bmatrix} L_{21} \\ \vdots \\ L_{N1} \end{bmatrix} = \begin{bmatrix} \tilde{L}_{21} \\ \vdots \\ \tilde{L}_{N1} \end{bmatrix},$$

then

$$PA = \left[\begin{array}{c|ccc} L_{11} & 0 & \cdots & 0 \\ \hline \tilde{L}_{21} & & & \\ \vdots & & L^{(\text{new})} & \\ \tilde{L}_{N1} & & & \end{array}\right] \left[\begin{array}{c|ccc} U_{11} & U_{12} & \cdots & U_{1N} \\ \hline 0 & & & \\ \vdots & & U^{(\text{new})} & \\ 0 & & & \end{array}\right]$$

is the pivoted block LU factorization of A with

$$P = \begin{bmatrix} I_r & 0 \\ 0 & P^{(\text{new})} \end{bmatrix} P_1.$$

In general, the processing of each block column in A is a four-part calculation:

Part A. Apply rectangular Gaussian Elimination with partial pivoting to a block column of A. This produces a permutation, a block column of L, and a diagonal block of U. See (3.6.2).

Part B. Apply the Part A permutation to the "rest of A." See (3.6.3).

Part C. Complete the computation of U's next block row by solving a lower triangular multiple right-hand-side problem. See (3.6.4).

Part D. Using the freshly computed L-blocks and U-blocks, update the "rest of A." See (3.6.5).

The precise formulation of the method with overwriting is similar to Algorithm 3.2.4 and is left as an exercise.

3.6.2 Parallelizing the Pivoted Block LU Algorithm

Recall the discussion of the block-cyclic distribution in §1.6.2 where the parallel computation of the matrix multiplication update $C = C + AB$ was outlined. To provide insight into how the pivoted block LU algorithm can be parallelized, we examine a representative step in a small example that also makes use of the block-cyclic distribution.

Assume that $N = 8$ in (3.6.1) and that we have a p_{row}-by-p_{col} processor network with $p_{\text{row}} = 2$ and $p_{\text{col}} = 2$. At the start, the blocks of $A = (A_{ij})$ are cyclically distributed as shown in Figure 3.6.1. Assume that we have carried out two steps of block LU and that the computed L_{ij} and U_{ij} have overwritten the corresponding A-blocks. Figure 3.6.2 displays the situation at the start of the third step. Blocks that are to participate in the Part A factorization

$$P_3 \begin{bmatrix} A_{33} \\ \vdots \\ A_{83} \end{bmatrix} = \begin{bmatrix} L_{33} \\ \vdots \\ L_{83} \end{bmatrix} U_{33}$$

are highlighted. Typically, p_{row} processors are involved and since the blocks are each r-by-r, there are r steps as shown in (3.6.6).

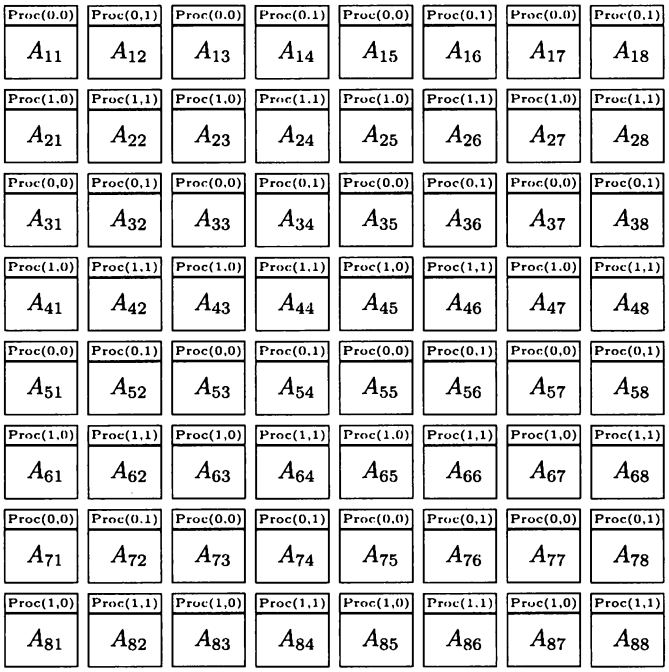


Figure 3.6.1.

Part A:

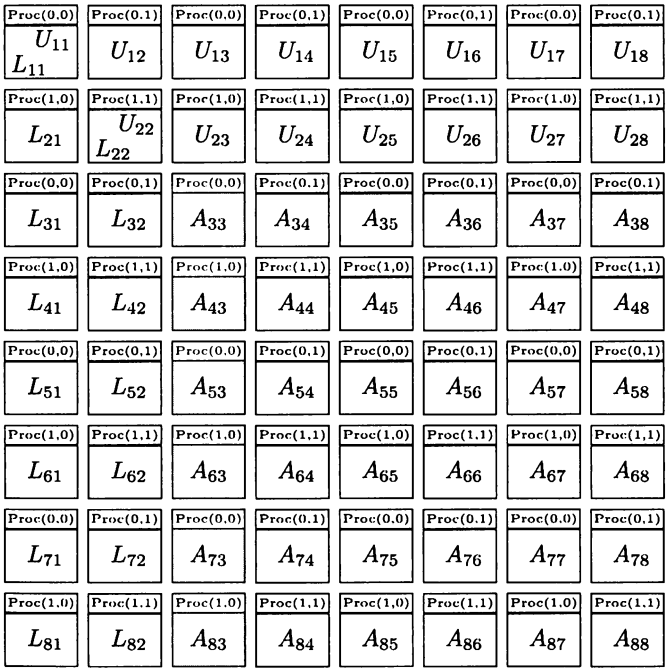


Figure 3.6.2.

for $j = 1{:}r$

Columns $A_{kk}(:,j),\ldots,A_{N,k}(:,j)$ are assembled in the processor housing A_{kk}, the "pivot processor"

The pivot processor determines the required row interchange and the Gauss transform vector

The swapping of the two A-rows may require the involvement of two processors in the network

The appropriate part of the Gauss vector together with $A_{kk}(j,j{:}r)$ is sent by the pivot processor to the processors that house $A_{k+1,k},\ldots,A_{N,k}$ (3.6.6)

The processors that house $A_{kk},\ldots,A_{N,k}$ carry out their share of the update, a local computation

end

Upon completion, the parallel execution of Parts B and C follow. In the Part B computation, those blocks that may be involved in the row swapping have been highlighted. See Figure 3.6.3. This overhead generally engages the entire processor network, although communication is local to each processor column.

Part B:

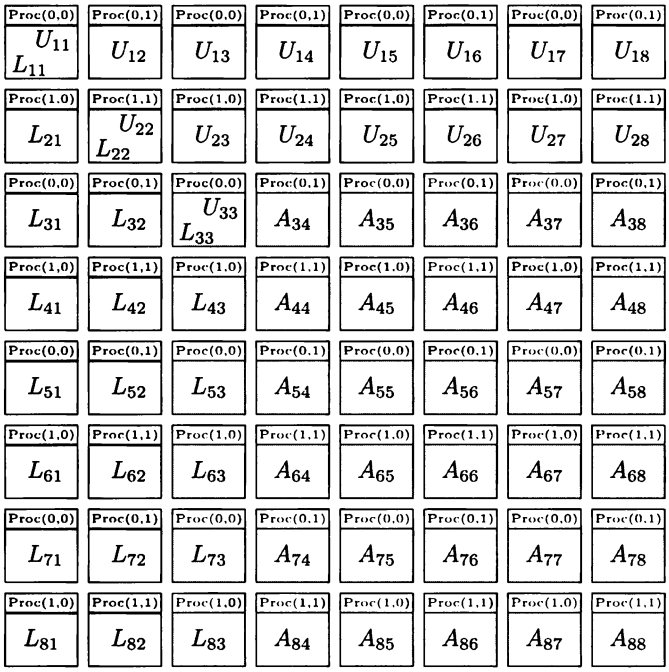


Figure 3.6.3.

Note that Part C involves just a single processor row while the "big" level-three update that follows typically involves the entire processor network. See Figures 3.6.4 and 3.6.5.

Part C:

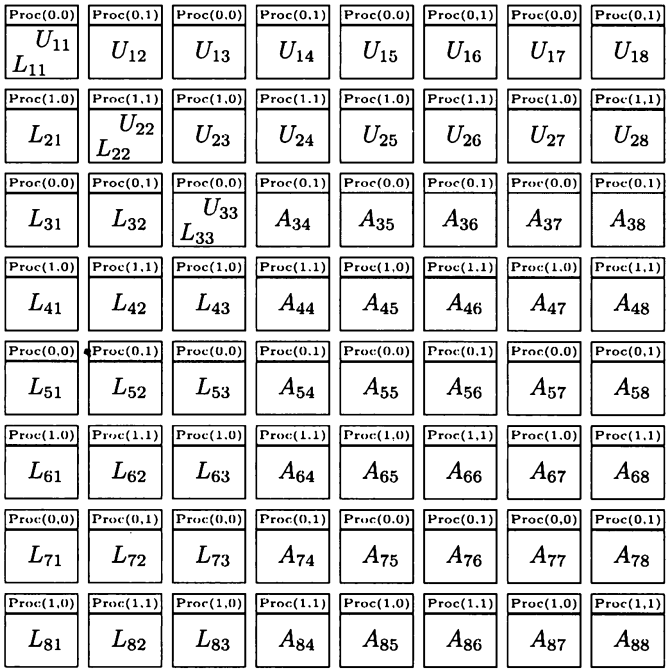


Figure 3.6.4.

Part D:

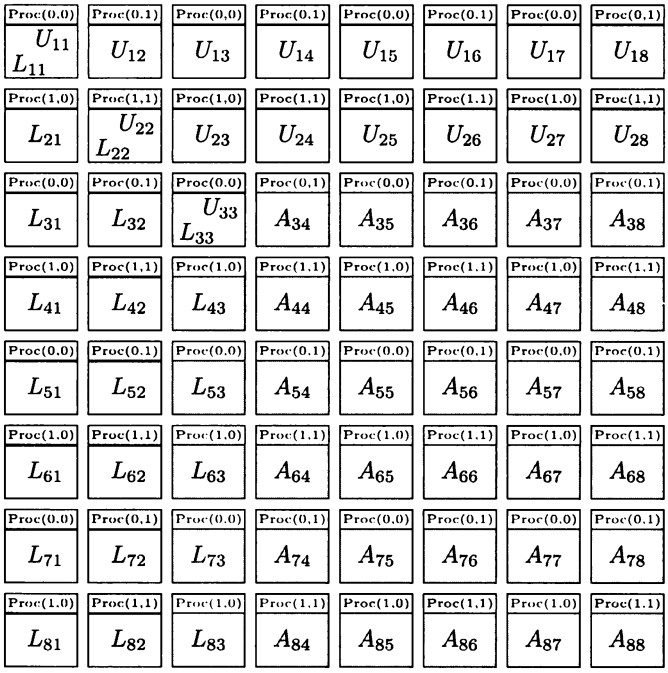


Figure 3.6.5.

The communication overhead associated with Part D is masked by the matrix multiplications that are performed on each processor.

This completes the $k = 3$ step of parallel block LU with partial pivoting. The process can obviously be repeated on the trailing 5-by-5 block matrix. The virtues of the block-cyclic distribution are revealed through the schematics. In particular, the dominating level-3 step (Part D) is load balanced for all but the last few values of k. Subsets of the processor grid are used for the "smaller," level-2 portions of the computation.

We shall not attempt to predict the fraction of time that is devoted to these computations or the propagation of the interchange permutations. Enlightenment in this direction requires benchmarking.

3.6.3 Tournament Pivoting

The decomposition via partial pivoting in Step A requires a lot of communication. An alternative that addresses this issue involves a strategy called *tournament pivoting*. Here is the main idea. Suppose we want to compute $PW = LU$ where the blocks of

$$W = \begin{bmatrix} W_1 \\ W_2 \\ W_3 \\ W_4 \end{bmatrix} \in \mathbb{R}^{n \times r}$$

are distributed around some network of processors. Assume that each W_i has many more rows than columns. The goal is to choose r rows from W that can serve as pivot rows. If we compute the "local" factorizations

$$P_1 W_1 = L_1 U_1, \qquad P_2 W_2 = L_2 U_2, \qquad P_3 W_3 = L_3 U_3, \qquad P_4 W_4 = L_4 U_4,$$

via Gaussian elimination with partial pivoting, then the top r rows of the matrices $P_1 W_1$, $P_2 W_2$, $P_3 W_3$, are $P_4 W_4$ are pivot row candidates. Call these square matrices W_1', W_2', W_3', and W_4' and note that we have reduced the number of possible pivot rows from n to $4r$.

Next we compute the factorizations

$$P_{12} W_{12}' = P_{12} \begin{bmatrix} W_1' \\ W_2' \end{bmatrix} = L_{12} U_{12},$$

$$P_{34} W_{34}' = P_{34} \begin{bmatrix} W_3' \\ W_4' \end{bmatrix} = L_{34} U_{34},$$

and recognize that the top r rows of $P_{12} W_{12}'$ and the top r rows of $P_{34} W_{34}'$ are even better pivot row candidates. Assemble these $2r$ rows into a matrix W_{1234} and compute

$$P_{1234} W_{1234} = L_{1234} U_{1234}.$$

The top r rows of $P_{1234} W_{1234}$ are then the chosen pivot rows for the LU reduction of W.

Of course, there are communication overheads associated with each round of the "tournament," but the volume of interprocessor data transfers is much reduced. See Demmel, Grigori, and Xiang (2010).

Problems

P3.6.1 In §3.6.1 we outlined a single step of block LU with partial pivoting. Specify a complete version of the algorithm.

P3.6.2 Regarding parallel block LU with partial pivoting, why is it better to "collect" all the permutations in Part A before applying them across the remaining block columns? In other words, why not propagate the Part A permutations as they are produced instead of having Part B, a separate permutation application step?

P3.6.3 Review the discussion about parallel shared memory computing in §1.6.5 and §1.6.6. Develop a shared memory version of Algorithm 3.2.1. Designate one processor for computation of the multipliers and a load-balanced scheme for the rank-1 update in which all the processors participate. A **barrier** is necessary because the rank-1 update cannot proceed until the multipliers are available. What if partial pivoting is incorporated?

Notes and References for §3.6

See the **scaLAPACK** manual for a discussion of parallel Gaussian elimination as well as:

J. Ortega (1988). *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, New York.

K. Gallivan, W. Jalby, U. Meier, and A.H. Sameh (1988). "Impact of Hierarchical Memory Systems on Linear Algebra Algorithm Design," *Int. J. Supercomput. Applic. 2,* 12–48.

J. Dongarra, I. Duff, D. Sorensen, and H. van der Vorst (1990). *Solving Linear Systems on Vector and Shared Memory Computers,* SIAM Publications, Philadelphia, PA.

Y. Robert (1990). *The Impact of Vector and Parallel Architectures on the Gaussian Elimination Algorithm*, Halsted Press, New York.

J. Choi, J.J. Dongarra, L.S. Osttrouchov, A.P. Petitet, D.W. Walker, and R.C. Whaley (1996). "Design and Implementation of the ScaLAPACK LU, QR, and Cholesky Factorization Routines," *Scientific Programming, 5,* 173–184.

X.S. Li (2005). "An Overview of SuperLU: Algorithms, Implementation, and User Interface," *ACM Trans. Math. Softw. 31,* 302–325.

S. Tomov, J. Dongarra, and M. Baboulin (2010). "Towards Dense Linear Algebra for Hybrid GPU Accelerated Manycore Systems," *Parallel Comput. 36*, 232–240.

The tournament pivoting strategy is a central feature of the optimized LU implementation discussed in:

J. Demmel, L. Grigori, and H. Xiang (2011). "CALU: A Communication Optimal LU Factorization Algorithm," *SIAM J. Matrix Anal. Applic. 32*, 1317-1350.

E. Solomonik and J. Demmel (2011). "Communication-Optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms," *Euro-Par 2011 Parallel Processing Lecture Notes in Computer Science, 2011, Volume 6853/2011,* 90–109.