

5

NUMERICAL LINEAR ALGEBRA

5.1 INTRODUCTION

In this chapter we will study methods for solving equations of the form

$$Ax = b \tag{5.1}$$

where $x, b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$ is an $n \times n$ matrix, and b and A are known.

Solving sets of linear equations is *the* most important problem in computational mathematics. Algorithms that solve non-linear problems generally use linear approximations which give rise to sets of linear equations. Non-linear algorithms generate sequences of these approximations which must be solved at each iteration. Hence it is important that robust, efficient algorithms be available to solve linear equations.

We will also consider briefly the following problems :

- *Matrix Equations.* Solve the matrix equation

$$AX = B \tag{5.2}$$

where $A \in \mathbb{R}^{n \times n}$, $X \in \mathbb{R}^{n \times m}$, and $B \in \mathbb{R}^{n \times m}$.

- *Matrix Inversion.* Solve the matrix equation

$$AX = I \tag{5.3}$$

This is a special case of (5.2) where $B = I$ and hence $X = A^{-1}$. This is the problem of inverting a matrix.

- *The Algebraic Eigenvalue Problem.* Solve

$$Ax = \lambda x \tag{5.4}$$

where $A \in \mathbb{R}^{n \times n}$, $x \in \mathbb{R}^n$, $\lambda \in \mathbb{C}$, and x and λ are unknowns.

Equation (5.4) is the Algebraic Eigenvalue problem where λ is an eigenvalue of A and x is an eigenvector of A .

If time permits, we will look at two other problems :

- Least Squares.

$$\min_x \|Ax - b\|^2 \quad (5.5)$$

- Linear Programming.

$$\min_x c^T x \quad \text{subject to} \quad Ax = b, \quad x \geq 0 \quad (5.6)$$

We will see that problems (5.1) (5.2) and (5.3) are computationally equivalent to the problem of multiplying two $n \times n$ matrices. We will see that (5.3) is a much more difficult problem that has been and is the subject of a vast literature.

5.2 MATHEMATICAL BASIS

This is a brief review of the basics of matrix algebra and analysis that we need to discuss and analyse methods for solving linear equations.

5.2.1 Matrix Algebra

We consider only square matrices in the chapter but many of the properties of square matrices hold for arbitrary rectangular matrix.

A square matrix A of order n is a square array of n^2 numbers which are real, unless stated otherwise. We denote a matrix A as follows

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = [a_{ij}]_{n \times n}$$

where a_{ij} is a general element of A . The subscripts i and j are called the row index and column index, respectively.

We may view a matrix A as a linear transformation $A : \mathbb{R}^n \rightarrow \mathbb{R}^n$ or as a vector $A \in \mathbb{R}^{n \times n}$.

We define the following algebraic operations on matrices

1. *Addition and subtraction* : $C = A \pm B$ where $c_{ij} = a_{ij} \pm b_{ij}$
2. *Multiplication by a scalar* : $C = \alpha A$ where $c_{ij} = \alpha a_{ij}$
3. *Multiplication of Two Matrices* : $C = AB$ where $c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$
4. *Transpose of a Matrix* : $C = A^T$ where $c_{ij} = a_{ji}$

Properties of Matrices and their operations

1. $A + B = B + A$
2. $(A + B)C = A + (B + C)$
3. $(A + B)^T = A^T + B^T$
4. $A(B + C) = AB + AC$
5. $A(BC) = (AB)C$
6. $(AB)^T = B^T A^T$
7. $AB \neq BA$ in general.

Special Types of Matrices

1. The Zero Matrix $0 = [0]_{n \times n}$.
2. The Identity Matrix I , where $I_{ij} = 1$, $i = j$ and $I_{ij} = 0$, $i \neq j$.
3. The Diagonal Matrix D where $d_{ij} = 0$, $i \neq j$.
4. The Symmetric Matrix $A^T = A$.
5. The Inverse Matrix A^{-1} where $AA^{-1} = A^{-1}A = I$.
6. The Upper Triangular Matrix A where $a_{ij} = 0$, $i > j$.
7. The Lower Triangular Matrix A where $a_{ij} = 0$, $i < j$.
8. The Unit Upper(Lower) Triangular Matrix A where A is upper(lower) triangular and $a_{ii} = 1$.
9. The Orthogonal Matrix A where $A^T = A^{-1}$ or $A^T A = I$.
10. The Positive-Definite Matrix A where $x^T A x > 0$ for all x .

An extremely important class of matrices are *Symmetric Positive Definite* because they arise in so many physical applications.

5.2.2 The Determinant of a Square Matrix: $\det(A)$

We define 'determinant' recursively :

$$\det(A_{1 \times 1}) = a_{11}, \quad \det(A_{2 \times 2}) = a_{11}a_{22} - a_{12}a_{21}$$

The **minor** M_{ij} of an element a_{ij} of A is the determinant of the matrix of order $n - 1$, obtained by deleting the i^{th} row and j^{th} column of A .

The **cofactor** A_{ij} of a_{ij} is $(-1)^{i+j} M_{ij}$.

We now define the determinant of A recursively as

$$\begin{aligned} \det(A) &= a_{i1}A_{i1} + a_{i2}A_{i2} + \cdots + a_{in}A_{in} \\ &= \sum_{j=1}^n a_{ij}A_{ij} \quad \text{for any given } i \\ &= \sum_{i=1}^n a_{ij}A_{ij} \quad \text{for any given } j \end{aligned}$$

This is called Laplace's Expansion.

Properties of the Determinant

1. If any row or column of A is zero then $\det A = 0$
2. If any two rows (columns) are interchanged then $\det(A') = -\det(A)$.
3. If any row (column) of A is multiplied by a scalar c then $\det(A_c) = c \det(A)$.
4. The value of a determinant is unaltered if a scalar multiple of one row (column) is added to another row (column).
5. If a row (column) is a scalar multiple of another row (column) then $\det(A) = 0$.
6. $\det(A^T) = \det(A)$
7. $\det(AB) = \det(A) \det(B)$
8. $\det(D) = d_{11}d_{22} \cdots d_{nn} = \prod_{i=1}^n d_i$
9. If $\det(A) \neq 0$ then A^{-1} exists and A is said to be **non-singular**.

This last property implies that A is non-singular if and only if the rows (columns) of A , regarded as vectors in \mathbb{R}^n , are linearly independent.

5.2.3 Linear Equations

Definition 5.1 (Rank of a Matrix). The **rank** of a matrix A is the maximum number of linearly independent columns of A . For square matrices, we also have the property $\text{rank}(A) = \text{rank}(A^T)$.

Theorem 5.1 (Linear Equations). *The following are equivalent for any square matrix A of order n .*

1. $Ax = b$ has a unique solution of all $b \in \mathbb{R}^n$.
2. $Ax = 0$ implies $x = 0$.
3. A^{-1} exists.
4. $\det(A) \neq 0$.
5. $\text{rank}(A) = n$.

Cramer's Method

The following is a method for solving $Ax = b$ which is theoretically important but impracticable for matrices of order 5 or more. Let $A_i = A$ with column i replaced by b .

Then the solution of $Ax = b$ is

$$x_i = \frac{\det(A_i)}{\det(A)} \quad i = 1, 2, \dots, n \quad (5.7)$$

If the evaluation of the determinants is carried out as shown above then the calculation of each x_i requires $O(n!)$ calculations. Thus the method is impracticable for even small values of n .

5.2.4 Matrix Analysis

In this section we consider the metric properties of vectors and matrices, i.e., their lengths or sizes and the distances between them. We will see, when we come to perturbation analysis, that we need to measure the distance between a matrix from the nearest singular matrix. Matrix norms are needed for this.

Vector Norms

We have already studied vector norms in Chapter 3 and so we merely re-state the norm axioms :

1. N1. $\|x\| \geq 0$ for all $x \in X$ $\|x\| = 0$ if and only if $x = 0$.
2. N2. $\|\alpha x\| = |\alpha| \|x\|$ for all $x \in X$ and all α .
3. N3. $\|x + y\| \leq \|x\| + \|y\|$

We have the following set of norms for $x \in \mathbb{R}^n$

$$\|x\|_p = \left[\sum_{i=1}^n |x_i|^p \right]^{\frac{1}{p}}, \quad 1 \leq p < \infty, \quad (5.8)$$

and

$$\|x\|_\infty = \max_i |x_i| \quad (5.9)$$

Theorem 5.2 (Equivalence of Norms). *Let $\|\cdot\|_u$ and $\|\cdot\|_v$ be any two norms. Then there exist constant $c_1, c_2 > 0$ for which*

$$c_1 \|x\|_u \leq \|x\|_v \leq c_2 \|x\|_u. \quad (5.10)$$

This theorem leads to the following :

$$\|x\|_2 \leq \|x\|_1 \leq \sqrt{n} \|x\|_2 \quad (5.11)$$

$$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n} \|x\|_\infty \quad (5.12)$$

$$\|x\|_\infty \leq \|x\|_1 \leq n \|x\|_\infty \quad (5.13)$$

Inner Product & Vector Angle

The *inner product* of two vectors x and y is defined as :

$$x \cdot y = x^T y = \sum_1^n x_i y_i. \quad (5.14)$$

The inner product is also called the *dot product*, for the obvious reason.

Using this definition we get, for the 2–norm $\|x\|_2$

$$\|x\|_2 = \left[\sum_{i=1}^n |x_i|^2 \right]^{\frac{1}{2}} = \left[\sum_{i=1}^n x_i x_i \right]^{\frac{1}{2}} = \sqrt{x^T x} = \sqrt{x \cdot x}. \quad (5.15)$$

The *angle* θ between two vectors x and y is defined as

$$\cos \theta = \frac{x \cdot y}{\sqrt{x \cdot x} \sqrt{y \cdot y}} = \frac{x \cdot y}{\|x\|_2 \|y\|_2} \quad (5.16)$$

The vectors $x/\|x\|_2 = x_N$ and $y/\|y\|_2 = y_N$ are called *normalized* because their lengths (norms) are 1.

A pair of vectors x, y is called *orthogonal* if $x \cdot y = 0$ and hence $\theta = \pi/2$. A set of vectors $X = \{x_1, x_2, \dots, x_n\}$ is orthogonal if $x_i \cdot x_j = 0$, $i \neq j$. A matrix A is orthogonal if its columns (rows) $\{a_1, a_2, \dots, a_n\}$, viewed as vectors, are orthogonal.

Matrix Norms

Matrix norms, to be useful, should have the following properties :

1. MN1. $\|A\|_m \geq 0$, for any non-zero A .
2. MN2. $\|\alpha A\|_m = |\alpha| \|A\|_m$, for any scalar α .
3. MN3. $\|A + B\|_m \leq \|A\|_m + \|B\|_m$.
4. MN4. $\|AB\|_m \leq \|A\|_m \|B\|_m$
5. MN5. $\|Ax\|_v \leq \|A\|_m \|x\|_v$

The fourth property is the triangle inequality for matrix multiplication, while the fifth is the matrix norm $\|\cdot\|_m$ induced by the vector norm $\|\cdot\|_v$

All these properties hold for the **standard definition of the induced matrix norm** :

$$\|A\|_m = \max_{\|x\|_v} \|Ax\|_v = \max_{x \neq 0} \frac{\|Ax\|_v}{\|x\|_v} \quad (5.17)$$

This definition gives a matrix norm $\|\cdot\|_m$ corresponding to each vector norm $\|\cdot\|_v$ because the right-hand side above is in terms of vector norms only.

Here are the matrix norms corresponding to the 1-, 2-, and ∞ -norms :

1. $\|x\|_1 = \sum_{i=1}^n |x_i|$ $\|A\|_1 = \max_j \sum_{i=1}^n |a_{ij}|$ Max Column Sum Norm.
2. $\|x\|_2 = [\sum_{i=1}^n |x_i|^2]^{\frac{1}{2}}$ $\|A\|_2 = [\rho(A^T A)]^{\frac{1}{2}}$ Spectral Norm.
3. $\|x\|_\infty = \max_i |x_i|$ $\|A\|_\infty = \max_i \sum_{j=1}^n |a_{ij}|$ Max Row Sum Norm.

where $\rho(A^T A)$ is the **spectral radius** of $A^T A$, i.e., the magnitude of the largest eigenvalue of $A^T A$.

Theorem 5.3 (Norm Bound). *Let A be an arbitrary square matrix. Then for any matrix norm $\|\cdot\|_m$*

$$\rho(A) \leq \|A\|_m. \quad (5.18)$$

Theorem 5.4 (Matrix Geometric Series). *Let A be a square matrix. If $\rho(A) < 1$ then $(I - A)^{-1}$ exists and can be expressed as*

$$(I - A)^{-1} = I + A + A^2 + \cdots + A^k + \cdots = \sum_{k=0}^{\infty} A^k \quad (5.19)$$

Note that we could have used $\|A\| < 1$ in Theorem 5.4 because $\|A\|_m < 1$ implies $\rho(A) \leq \|A\|_m < 1$ by Theorem (5.18).

5.3 MATRIX FACTORIZATIONS and APPLICATIONS

[This section is being re-written and will include examples of the application of these factorizations.]

Matrix factorizations (or decompositions) are central to the study of (computational) linear algebra. The purpose of factorization or decomposition is to break a matrix A into two or more matrices X, Y, Z in the hope that X, Y, Z reveal some property of A that is not apparent by looking at A , or that solving a problem involving A is made easier by using X, Y , and Z instead. This is a very vague description which we now make concrete by looking at the classic factorizations of linear algebra.

5.3.1 LU Factorization

Let $A \in \mathbb{R}^{n \times n}$. Then A can be factorized into $P, L, U \in \mathbb{R}^{n \times n}$ such that

$$PA = LU, \quad (5.20)$$

where L is a unit lower triangular matrix (ones on the diagonal) and U is upper triangular, and P is a permutation matrix.

5.3.2 Cholesky Factorization

Let $A \in \mathbb{R}^{n \times n}$ be a *symmetric positive definite* matrix, then A can be factorized into the product of three matrices

$$A = LDL^T, \quad (5.21)$$

where L is unit lower triangular and $D = \text{diag}(d_1, d_2, \dots, d_n)$ is a diagonal matrix with $d_i > 0$ for all i .

We may re-arrange this factorization as follows :

$$A = LDL^T = (L\sqrt{D})(\sqrt{D}L^T) = CC^T. \quad (5.22)$$

This is the Cholesky Factorization of A .

5.3.3 QR Factorization

Let $A \in \mathbb{R}^{m \times n}$ be a matrix of rank n . Then there is a decomposition of A into a product of two matrices

$$A = QR, \quad (5.23)$$

where Q is an $m \times m$ orthogonal matrix and R is an $m \times n$ upper triangular matrix with positive diagonal elements.

5.3.4 Singular Value Decomposition

Let $A \in \mathbb{R}^{m \times n}$ be a matrix of rank r . Then there is a decomposition of A into a product of three matrices

$$A = U\Sigma V^T, \quad (5.24)$$

where U is an $m \times m$ orthogonal matrix, V is an $n \times n$ orthogonal matrix and Σ is an $m \times n$ diagonal matrix, with elements The elements are usually ordered so that

$$\sigma_1 \geq \sigma_2 \geq \cdots \sigma_r \geq \sigma_{r+1} = \cdots = \sigma_p = 0, \quad p = \min\{m, n\} \quad (5.25)$$

and are called the *singular values* of A .

5.3.5 Spectral Decomposition

Let $A \in \mathbb{C}^{n \times n}$ be an $n \times n$ complex-valued matrix. A complex scalar λ such that

$$Ax = \lambda x, \quad x \neq 0, \quad (5.26)$$

is called an *eigenvalue* of A and x is an *eigenvector* of A corresponding to λ .

We can see that $Ax - \lambda x = 0 = (A - \lambda I)x$ has a non-zero solution if, and only if, $A - \lambda I$ is singular. That is, if λ satisfies the *characteristic equation* of A ,

$$\det(A - \lambda I) = 0. \quad (5.27)$$

The expression $\det(A - \lambda I)$ is in fact an n -degree polynomial in λ and so, by the fundamental theorem of algebra, the characteristic equation has n roots $\lambda_1, \lambda_2, \dots, \lambda_n$, not necessarily distinct. The set of eigenvalues $\lambda(A) = \{\lambda_1, \lambda_2, \dots, \lambda_n\}$, is called the *spectrum* of A .

Let $\Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$ be the diagonal matrix formed from the eigenvalues of A , and let $V = (v_1, v_2, \dots, v_n)$ be the corresponding eigenvectors. Hence we have $Av_i = \lambda_i v_i$, $i = 1, 2, \dots, n$. These n equations may be written

$$A(v_1, v_2, \dots, v_n) = (\lambda_1 v_1, \lambda_2 v_2, \dots, \lambda_n v_n) = (v_1, v_2, \dots, v_n) \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n), \text{ or, } AV = V\Lambda. \quad (5.28)$$

Now, if the eigenvectors $V = (v_1, v_2, \dots, v_n)$ are linearly independent, then V is invertible and we may write

$$A = V\Lambda V^{-1} \text{ or } V^{-1}AV = \Lambda. \quad (5.29)$$

This is called the spectral decomposition or *Jordan Canonical Form* of A . The transformation $\Lambda = V^{-1}AV$ is called a *similarity transformation* and A and Λ are called *similar matrices*.

The Jordan Canonical Form is used extensively in linear system theory and design (modern control theory).

Example 5.1 (Linear Time-Invariant State-Space Equations). The following set of equations is central to state-space control theory :

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned}$$

where A, B, C, D are, respectively, $n \times n$, $n \times p$, $q \times n$ and $q \times p$, real constant matrices and u, x, y are $p \times 1$, $n \times 1$, $q \times 1$, respectively. The block diagram for this system is shown in Figure 5.1. The first equation is a first-order differential equation in the **state** variable $x(t)$. The vector

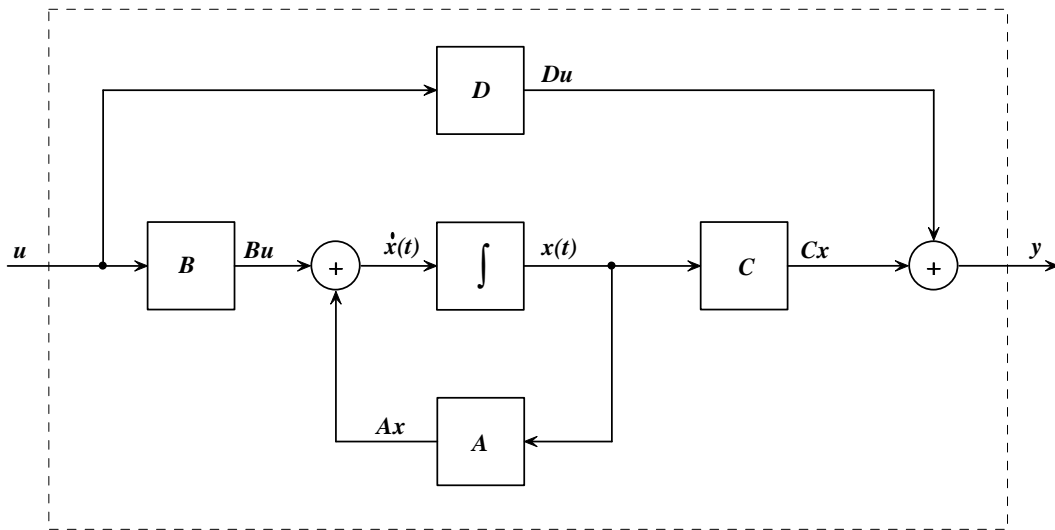


Figure 5.1 : State-Space Model

$u(t)$ is the **input** or forcing function vector. The second equation defines the **output** vector $y(t)$. It is purely algebraic and need not concern us here.

The solution of $\dot{x}(t) = Ax(t) + Bu(t)$ is

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\tau)}Bu(\tau) d\tau. \quad (5.30)$$

We will simplify matters and consider just the homogeneous equation ($u(t) = 0$)

$$\dot{x}(t) = Ax(t) \quad \text{whose solution is} \quad x(t) = e^{At}x(0). \quad (5.31)$$

5.4 BASIC LINEAR ALGEBRA ALGORITHMS (BLAAs)

Most Numerical Linear Algebra algorithms can be constructed from the following primitive or basic operations :

Table 5.1: Basic Linear Algebra Operations

Level	Work	Operation	Name
1	$O(n)$	$x \leftarrow \alpha x$	Vector scaling
1	$O(n)$	$y \leftarrow \alpha x + y$	Vector addition
1	$O(n)$	$d \leftarrow x^T y$	Dot Product
2	$O(n^2)$	$y \leftarrow \alpha Ax + \beta y$	Matrix-Vector multiplication.
3	$O(n^3)$	$C \leftarrow \alpha AB + \beta C$	Matrix-Matrix multiplication.

Level 1 Operations – Vector Scaling, Addition, & Dot Product

These operations are defined at the vector component level as follows :

$$x \leftarrow \alpha x : \quad x_i := \alpha \times x_i, \quad i = 1, 2, \dots, n. \quad (5.32)$$

$$y \leftarrow \alpha x + y : \quad y_i := \alpha \times x_i + y_i, \quad i = 1, 2, \dots, n. \quad (5.33)$$

$$d \leftarrow x^T y : \quad d := \sum_{i=1}^n x_i y_i. \quad (5.34)$$

These three operations obviously require n , $2n$, and $2n - 1$ flops respectively.

Level 2 Operations – Matrix-Vector Multiplication

Level 3 Operations – Matrix Matrix-Matrix Multiplication

The standard mathematical definition of matrix multiplication of $A_{m \times p}$ and $B_{p \times n}$ is the matrix $C_{m \times n}$ defined as follows :

$$c_{ij} = \sum_{k=1}^p a_{ik} \times b_{kj}, \quad i = 1, 2, \dots, m, \quad j = 1, 2, \dots, n. \quad (5.35)$$

The algorithm for this definition is

```
algorithm MatMult( $a, b, m, p, n, c$ )  
for  $i := 1$  to  $m$  do  
  for  $j := 1$  to  $n$  do  
     $c_{ij} := 0$   
    for  $k := 1$  to  $p$  do  
       $c_{ij} := c_{ij} + a_{ik} \times b_{kj}$   *  
    endfor  $k$   
  endfor  $j$   
endfor  $i$   
endalg MatMult
```

There are 2 flops(+, \times) for each execution of the (*) statement in the inner-most loop. Thus the number of flops is $2mnp$. If both matrices are $n \times n$ then the number of flops is $2n^3$.

— TO BE COMPLETED —

5.5 DIRECT METHODS FOR SOLVING $Ax = b$

Direct methods of solution in numerical problems seek to transform the original problem into one that is easier to solve than the original.

Direct methods are fundamentally different from iterative methods which start with an initial approximation to the solution and successively improve the approximation using (usually the data of the original problem).

The simplest example that illustrates the difference between direct and iterative methods is the following scalar problem:

Example 5.2 (Solve $ax = b$).

1. Direct Method : $ax = b \rightarrow x = b/a$
2. Iterative method : $x_{k+1} = x_k(1 + b - ax_k)$, $k = 0, 1, 2, \dots$

In the direct method the problem is transformed to $x = b/a$ whose solution requires 1 step (division). The iterative method starts with an initial guess x_0 and performs a potentially (and theoretically) infinite sequence of calculations.

Direct methods for solving $Ax = b$ apply elementary matrix operations to A and b which gives a transformed problem $T(Ax = b) = A'x' = b'$ which is easily solved for x' . The solution of the original problem x is then found as $x = T^{-1}(x')$. This last step is not always necessary.

5.5.1 Upper and Lower Triangular Systems

If $Ax = b$ has the following special form (upper triangular)

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}, \quad (5.36)$$

then these equations may be easily solved by **Back-substitution**, i.e., solve for x_i in the i th equation, given x_{i+1}, \dots, x_n . Substitute x_i, x_{i+1}, \dots, x_n in the $i - 1$ st equation and solve for x_{i-1} .

This gives the following algorithm

```

algorithm BackSubst ( $a, b, n, x$ )
 $x_n := b_n / a_{nn}$ 
for  $i := n - 1$  downto 1 do
   $sum := 0.0$ 
  for  $j := i + 1$  to  $n$  do
     $sum := sum + a_{ij} \times x_j$ 
  endfor  $j$ 
   $x_i := (b_i - sum) / a_{ii}$ 
endfor  $i$ 
endalg BackSubst

```

If $Ax = b$ is lower triangular then we solve for the x_i 's in reverse order. This is called **Forward Substitution**.

$$\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}, \quad (5.37)$$

The equivalent algorithm is

```

algorithm ForwSubst ( $a, b, n, x$ )
 $x_1 := b_1 / a_{11}$ 
for  $i := 2$  to  $n$  do
   $sum := 0.0$ 
  for  $j := 1$  to  $i - 1$  do
     $sum := sum + a_{ij} \times x_j$ 
  endfor  $j$ 
   $x_i := (b_i - sum) / a_{ii}$ 
endfor  $i$ 
endalg ForwSubst

```

Analysis of Algorithm *BackSubst*

We concentrate on the inner-most loop. This has two floating point operations (flops). Hence the total number of flops is

$$\begin{aligned} S(n) &= \sum_{k=1}^{n-1} \sum_{j=k+1}^n 2 = 2 \sum_{k=1}^{n-1} (n - k) = 2 \sum_{k=1}^{n-1} n - 2 \sum_{k=1}^{n-1} k \\ &= 2n(n-1) - n(n-1) = n(n-1) \approx n^2, \quad \text{for large } n. \end{aligned}$$

Hence Back Substitution is $O(n^2)$. Likewise, Forward Substitution is $O(n^2)$

Notice that the row indices in this matrix occur in the same order as $p = [4213]$.

If we post-multiply A by P_r we get

$$AP_r = \begin{bmatrix} 13 & 12 & 14 & 11 \\ 23 & 22 & 24 & 21 \\ 33 & 32 & 34 & 31 \\ 43 & 42 & 44 & 41 \end{bmatrix}, \text{ which is } A \text{ with its columns interchanged.}$$

Exercise 5.2. Repeat the operations above using P_c .

Properties of Permutation Matrices

Permutation matrices have many interesting properties, some of which we list here. The proofs of these properties are left as exercises.

1. $PP^T = P^T P = I$
2. $P^{-1} = P^T$
3. $\det(P) = \pm 1$

Definition 5.2 (Stochastic Matrix). A non-negative matrix S is a row (column) **stochastic matrix** if the sum of the elements in every row (column) is 1. That is

$$\sum_{j=1}^n s_{ij} = 1, \quad i = 1, 2, \dots, n, \quad \text{or} \quad \sum_{i=1}^n s_{ij} = 1, \quad j = 1, 2, \dots, n.$$

The matrix S is **doubly-stochastic** if all row and column sums are 1.

Theorem 5.5 (Birkoff-von Neumann Theorem). *Every doubly-stochastic matrix is a convex combination of permutation matrices.*

This means that if S is a doubly-stochastic matrix then there exist permutation matrices P_1, P_2, \dots, P_N , with $N < \infty$, and positive scalars a_1, a_2, \dots, a_N such that

$$\sum_{k=1}^N a_k = 1 \quad \text{and} \quad S = \sum_{k=1}^N a_k P_k \tag{5.41}$$

Proof : See page 330, Chvátal, *Linear Programming*, W.H. Freeman & Co., 1983.

Here is a MATLAB function that generates a doubly-stochastic matrix from a random convex combination of random permutation matrices.

```
function S = DStochMat(k,n)
    S = zeros(n,n);
    I = eye(n);
    t = rand(k,1); t = t./sum(t);
    for i = 1:k
        pv = randperm(n);
        P = I(pv,:);    random P matrix
        S = S + t(i)*P;
    end;
```

Exercise 5.3. Prove that $\det(P) = \pm 1$.

Exercise 5.4. How many $n \times n$ permutation matrices are there?

Exercise 5.5. If adding a multiple of row j to row i is accomplished by $C = MA$ what is the result of $C = AM$?

Exercise 5.6. What is the result of $C = M^T A$? What is the result of $C = AM^T$?

Exercise 5.7. Show that M^{-1} is the identity matrix with $-\alpha$ in position (i, j) .

Exercise 5.8. What is $\det(M)$?

5.6 GAUSSIAN ELIMINATION

Gaussian Elimination is a very old algorithm. It is said to have been used in China 1000 years ago. The algorithm, as we know it, was developed by Gauss (1777–1855). It is this algorithm that Gauss is referring to when he says ‘eliminate directly’ in the quotation at the beginning of this chapter.

Despite its age, it is still the best way of solving the general linear system of equations $Ax = b$. And despite its reputation there are still people who will trot out the Gauss-Jordan method, and, incredibly, Cramer’s method.

Example 5.3. Consider solving the following set of linear equations

$$\begin{aligned}x_1 + 2x_2 + 3x_3 &= 6 \\2x_1 - 3x_2 + 2x_3 &= 14 \\3x_1 + x_2 - x_3 &= -1\end{aligned}$$

Step 1. Eliminate x_1 from the 2nd and 3rd equations by subtracting 2 times the 1st equation from the 2nd, and 3 times the 1st equation from the 3rd:

$$\begin{aligned}x_1 + 2x_2 + 3x_3 &= 6 \\-7x_2 - 4x_3 &= 2 \\-5x_2 - 10x_3 &= -20\end{aligned}$$

Step 2. Eliminate x_2 from 3rd equation by subtracting $5/7$ times the 2nd equation from the 3rd:

$$\begin{aligned}x_1 + 2x_2 + 3x_3 &= 6 \\-7x_2 - 4x_3 &= 2 \\-\frac{50}{7}x_3 &= -\frac{150}{7}\end{aligned}$$

We can now solve for x_3

$$-\frac{50}{7}x_3 = -\frac{150}{7} \text{ gives } x_3 = 3.$$

We can now solve for x_2

$$-7x_2 - 4 \times 3 = 2 \text{ gives } x_2 = -2.$$

We can now solve for x_1

$$x_1 + 2 \times (-2) + 3 \times 3 = 6 \text{ gives } x_1 = 1.$$

In matrix-vector form these steps are

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & -3 & 2 \\ 3 & 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 14 \\ -1 \end{bmatrix} \xrightarrow{\text{Step 1}} \begin{bmatrix} 1 & 2 & 3 \\ 0 & -7 & -4 \\ 0 & -5 & -10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 2 \\ -20 \end{bmatrix} \xrightarrow{\text{Step 2}} \begin{bmatrix} 1 & 2 & 3 \\ 0 & -7 & -4 \\ 0 & 0 & -\frac{50}{7} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 2 \\ -\frac{150}{7} \end{bmatrix}$$

We can see that these two elimination steps have transformed the original problem $Ax = b$ to a new problem $Ux = b'$, where U is an upper-triangular matrix. This can now be solved using the *BackSubst* algorithm.

5.6.1 Derivation of the Gaussian Elimination Algorithm

The Gaussian Elimination Method uses a sequence of elementary matrix operations to transform the square system $Ax = b$ into an upper triangular system $Ux = b'$ which is then solved using back substitution.

The method starts at stage $k = 1$ with

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} & b_2^{(1)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} & b_n^{(1)} \end{bmatrix} = \left[A^{(1)} \mid b^{(1)} \right]$$

The coefficients $a_{21}^{(1)}, a_{31}^{(1)}, \dots, a_{n1}^{(1)}$ below the *pivot element* $a_{11}^{(1)}$ are eliminated (reduced to zero) by subtracting the following multiples of row 1 from rows 2, 3, ..., n :

$$\frac{a_{21}^{(1)}}{a_{11}^{(1)}}, \frac{a_{31}^{(1)}}{a_{11}^{(1)}}, \dots, \frac{a_{n1}^{(1)}}{a_{11}^{(1)}} = m_{21}, m_{31}, \dots, m_{n1}, \quad \text{where} \quad m_{k1} = \frac{a_{k1}^{(1)}}{a_{11}^{(1)}} \quad k = 2, 3, \dots, n.$$

This gives the transformed system at stage 2

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)} \\ \vdots & & & & \\ 0 & a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} & b_n^{(2)} \end{bmatrix} = \left[A^{(2)} \mid b^{(2)} \right].$$

At stage k we have

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1k}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2k}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)} \\ \vdots & & \ddots & \vdots & & & \\ 0 & \cdots & 0 & a_{kk}^{(k)} & \cdots & a_{kn}^{(k)} & b_k^{(k)} \\ \vdots & & \vdots & & & & \\ 0 & \cdots & 0 & a_{nk}^{(k)} & \cdots & a_{nn}^{(k)} & b_n^{(k)} \end{bmatrix} = \left[A^{(k)} \mid b^{(k)} \right] \quad (5.42)$$

The elements $a_{k+1,k}^{(k)}, a_{k+2,k}^{(k)}, \dots, a_{nk}^{(k)}$ are eliminated by subtracting the following multiples of row k from rows $k+1, k+2, \dots, n$:

$$\frac{a_{k+1,k}^{(k)}}{a_{kk}^{(k)}}, \frac{a_{k+2,k}^{(k)}}{a_{kk}^{(k)}}, \dots, \frac{a_{k+n,k}^{(k)}}{a_{kk}^{(k)}} = m_{k+1,k}, m_{k+2,k}, \dots, m_{n,k}$$

We have in general, assuming $a_{kk}^{(k)} \neq 0$

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \quad i = k+1, \dots, n \quad (5.43)$$

and

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik}a_{kj}^{(k)}, \quad i, j = k+1, \dots, n \quad (5.44)$$

$$b_i^{(k+1)} = b_i^{(k)} - m_{ik}b_k^{(k)}, \quad i = k+1, \dots, n. \quad (5.45)$$

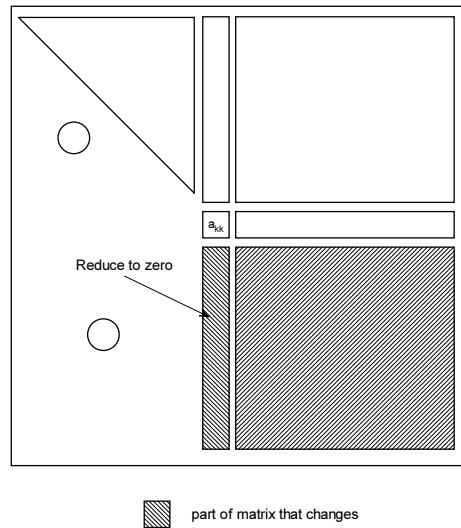


Figure 5.2 : Gaussian Elimination : changes at stage k

We summarize all of these calculations in the following algorithm

```

algorithm GaussElim (a, b, n)
  Assume that no  $a_{kk} = 0$ 
  for  $k := 1$  to  $n - 1$  do
    for  $i := k + 1$  to  $n$  do
       $m_{ik} := a_{ik} / a_{kk}$ 
      for  $j := k + 1$  to  $n$  do
         $a_{ij} := a_{ij} - m_{ik} \times a_{kj}$ 
      endfor  $j$ 
       $b_i := b_i - m_{ik} \times b_k$ 
    endfor  $i$ 
  endfor  $k$ 
endalg GaussElim
  
```

This algorithm transforms the original system $Ax = b$ to the following form :

$$Ux = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn}^{(n)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(n)} \end{bmatrix}$$

This system of equations can now be solved using Back Substitution.

Analysis of Algorithm *GaussElim*

We concentrate on the operations in the inner-most loop. We see that there are only 2 flops performed. Hence the number of steps performed is

$$\begin{aligned}
 S(n) &= \sum_{k=1}^{n-1} \sum_{i=k+1}^n \sum_{j=k+1}^n 2 = 2 \sum_{k=1}^{n-1} \sum_{i=k+1}^n (n-k) \\
 &= 2 \sum_{k=1}^{n-1} (n-k)^2 = 2[(n-1)^2 + (n-2)^2 + \cdots + 2^2 + 1^2] \\
 &= 2n(n-1)(2n-1)/6 \approx \frac{2n^3}{3}, \text{ for large } n.
 \end{aligned} \tag{5.46}$$

Hence Gaussian Elimination is an $O(n^3)$ process. Note that Matrix Multiplication requires 3 times as much work.

Exercise 5.9. The transformed system above uses the original x . Is this correct? Why?

Observations on *GaussElim*

1. Assumes $a_{kk}^{(k)} \neq 0$.
2. A and b are overwritten.
3. The 0's beneath the pivot element are not calculated. They are ignored.
4. An extra matrix is not needed to store the m_{ik} 's. They can be stored in place of the zeros.
5. The operations on b can be done separately, once we have stored the m_{ik} 's.
6. Because of 5 we may now solve for any b without going through the elimination calculations again.

5.6.2 LU Decomposition

We solved $Ax = b$ using Gaussian Elimination which required elementary row operations to be performed on both A and b . These operations are determined by the elements of A only. If we are required to solve the new equation $Ax = b'$ then *GaussElim* we would perform exactly the same operations because A is the same in both equations. Hence if we have stored the multipliers m_{ik} we need to perform only the final line of Algorithm *GaussElim*, i.e.,

$$b_i := b_i - m_{ik}b_k, \quad i = k+1, \dots, n, \quad k = 1, \dots, n-1. \tag{5.47}$$

If at each stage k of *GaussElim* we store m_{ik} in those cells of A that become zero then the A matrix after elimination would be as follows

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ m_{21} & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & a_{nn}^{(n)} \end{bmatrix} = \begin{bmatrix} \diagdown & U \\ L & \diagdown \end{bmatrix} \tag{5.48}$$

We define the upper and unit lower triangular parts as

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix} \quad \text{and} \quad L = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ m_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & 1 \end{bmatrix} \quad (5.49)$$

We now modify *GaussElim* to incorporate these ideas :

algorithm *GaussElimLU* (a, n)

Assume that no $a_{kk} = 0$

for $k := 1$ **to** $n - 1$ **do**

for $i := k + 1$ **to** n **do**

$a_{ik} := a_{ik} / a_{kk}$

for $j := k + 1$ **to** n **do**

$a_{ij} := a_{ij} - a_{ik} \times a_{kj}$

endfor j

endfor i

endfor k

endalg *GaussElimLU*

Notice that we do not store the diagonal elements of L because these are all 1s and so are known at any stage of the computations.

Theorem 5.6 (*LU Decomposition*). *If L and U are the upper and lower triangular matrices generated by Gaussian Elimination, assuming $a_{kk}^{(k)} \neq 0$ at each stage, then*

$$A = LU = \sum_{k=1}^n l_{ik} u_{kj}$$

where $u_{kj} = a_{kj}^{(k)} \quad k \leq j, \quad u_{kk} = a_{kk}^{(k)}$
 $l_{ik} = m_{ik} \quad k \leq i, \quad l_{kk} = 1,$

and this decomposition is unique.

Proof 5.1. *Algebraic manipulation (see Watkins, 1991, pages 51–53)*

We can now interpret Gaussian Elimination as a process which decomposes A into L and U and hence we have

$$Ax = LUx = L(Ux) = Ly = b.$$

This represents two triangular systems of equations

$$Ly = b \quad \text{and} \quad Ux = y$$

whose solutions are :

$$y = L^{-1}b, \quad Ux = L^{-1}b, \quad x = U^{-1}L^{-1}b.$$

The steps in solving $Ax = b$ using LU Decomposition are as follows:

algorithm	$SolveLU(a, b, n, x)$
1.	Calculate L and U using $GaussElimLU(A, n)$, where $A = LU$.
2.	Solve $Ly = b$ using $ForwSubst(L, b, n, y)$, where $y = L^{-1}b$.
3.	Solve $Ux = y$ using $BackSubst(U, y, n, x)$, where $x = U^{-1}y$.

The number of steps in this algorithm is

$$S(n) = O(n^3) + O(n^2) + O(n^2) = O(n^3)$$

Notice that we can easily solve $Ax = b$ for many different bs because step 1 ($A \rightarrow LU$) does not need to be performed for each b .

The LDU Decomposition of A

Gaussian Elimination also provides the decomposition

$$A = LDU', \quad (5.50)$$

where L and U' are unit lower and unit upper triangular and $D = [u_{ii}]$.

To see this decompose $A = LU$ and let $U' = D^{-1}U$. Since U is non-singular, $u_{ii} \neq 0$, $i = 1, 2, \dots, n$ and hence $D^{-1} = [1/u_{ii}]$ exists. It is easy to show that $D^{-1}U$ is a unit upper triangular matrix. Thus,

$$A = LU = LDD^{-1}U = LDU'.$$

If A is *symmetric* then

$$A = LDU' = LDL^T,$$

where L is unit lower triangular.

The Cholesky Decomposition of A

If A is symmetric and *positive definite* ($x^T Ax > 0$) then we can decompose A as follows :

$$A = LDL^T = L\sqrt{D}\sqrt{D}L^T = CC^T, \quad (5.51)$$

where $C = L\sqrt{D}$ and $\sqrt{D} = [\sqrt{d_{ii}}]_1^n$. This is possible because $x^T Ax > 0 \Rightarrow d_{ii} > 0$. This is often called the **Cholesky Factorization** of A .

5.6.3 Using the LU Decomposition

As we have seen, once the the matrices L and U have been computed, we can solve $Ax = b$ quite easily using $ForwSubst$ and $BackSubst$. The same is true for the other problems associated with equation-solving.

The Determinant of A

We have

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

Now $\det(L) = 1$ because L is unit lower triangular and $\det(U) = u_{11}u_{22} \dots u_{nn}$ because U is upper triangular. Hence

$$\det(A) = u_{11}u_{22} \dots u_{nn} = \prod_{i=1}^n u_{ii}. \quad (5.52)$$

Although this calculation is straight-forward it can give rise to under- or over-flow and careful programming is needed to avoid these exceptions.

Solving $AX = B$

If A is $n \times n$ and X and B are $n \times m$, then we first form LU using Gaussian Elimination. If we set $x^j = j$ th column of X and $b^j = j$ th column of B then we merely solve the sequence of equations

$$LUx^j = b^j \quad j = 1, 2, \dots, m. \quad (5.53)$$

The algorithm is as follows :

algorithm *SolveAXB* (A, X, B, m, n)

A is an $n \times n$ matrix, X and B are $n \times m$ matrices
 b^j and x^j are the j th columns of B and X , respectively

GaussElimLU(A, n) returns L and U where $A = LU$

for $j := 1$ **to** m **do**
 ForwSubst(L, b^j, n, y)
 BackSubst(U, y, n, x^j)
endfor j

endalg *SolveAXB*

GaussElimLU requires $2n^3/3$ steps and *ForwSubst* and *BackSubst* require $n^2/2$ steps each. Hence the number of steps for solving $AX = B$ is

$$S(n) = 2n^3/3 + m(n^2 + n^2) = 2n^3/3 + 2mn^2 = O(\max\{n^3, mn^2\}).$$

Matrix Inversion, A^{-1}

Solve $AX = B$, where $B = I$. Thus matrix inversion is equivalent to solving

$$Ax^j = e^j \quad j = 1, 2, \dots, n \quad (5.54)$$

where x^j is the j th column of $X = A^{-1}$ and e^j is the j th column of I . The algorithm for matrix inversion is a simple modification of *SolveAXB* :

algorithm *Invert* (A, X, n)

A is an $n \times n$ matrix, X is the inverse of A
 e^j and x^j are the j th columns of I and X , respectively

GaussElimLU(A, n) returns L and U where $A = LU$

for $j := 1$ **to** n **do**

ForwSubst(L, e^j, n, y)

BackSubst(U, y, n, x^j)

endfor j

endalg *Invert*

The number of steps for matrix inversion is

$$S(n) = 2n^3/3 + n(n^2 + n^2) = 8n^3/3 = O(n^3).$$

Thus we see that matrix inversion has the same complexity as matrix multiplication.

Exercise 5.10. The number of steps in *ForwSubst*(L, e^j, n, y^j) above can be reduced because e^j is zero except for a 1 in the j th position. Modify *ForwSubst*(L, e^j, n, y^j) to take advantage of this. Analyse the result and show that matrix inversion can be performed in $2n^3$ steps. This means that matrix inversion is no more difficult than matrix multiplication.

The Complexity of Matrix Operations

Matrix Multiplication, Equation Solving, Inversion and Determinant Calculation are all $O(n^3)$ processes and in this sense are equivalent. Indeed, if there exists a method for matrix multiplication that is $O(n^p)$, $p < 3$, then it can be shown that there are $O(n^p)$ algorithms for all the other matrix operations. (see Aho, Hopcroft, and Ullman, 1975).

Matrix multiplication is defined as

$$c_{ij} = \sum_{k=1}^n a_{ik} \times b_{kj}, \quad i, j = 1, 2, \dots, n.$$

The calculation of each c_{ij} requires $O(n)$ operations and so matrix multiplication, using this definition, requires $O(n^3)$ operations.

For 2×2 matrices we have

$$c_{ij} = \sum_{k=1}^2 a_{ik} \times b_{kj} = a_{i1} \times a_{1j} + a_{i2} \times b_{2j}, \quad i, j = 1, 2.$$

Each element c_{ij} requires 2 multiplications and 1 addition giving a total for all elements of 8 multiplications and 4 additions.

Matrix algebra was developed by Cayley, Sylvester, and others in the 1850's and until recently the only known method for matrix multiplication was the standard definition which has $O(n^3)$

complexity. In 1969 Strassen found a method for multiplying two 2×2 matrices using 7 multiplications and 18 additions instead of 8 multiplications and 4 additions for the standard algorithm. This was generalized to $n \times n$ matrices and resulted in an $O(n^{\log_2 7}) = O(n^{2.807})$ matrix multiplication algorithm.

Since then there has been remarkable progress and the current record is held by Coppersmith and Winograd (1987) with an $O(n^{2.376})$ algorithm.

The best lower bound on matrix multiplication is $O(n^{2+\epsilon})$ with $\epsilon > 0$. This is not very tight because just reading a matrix is $O(n^2)$.

5.6.4 Gaussian Elimination and LU Decomposition

In Section 4.2.4 we merely stated the connection between Gaussian Elimination and LU Decomposition in Theorem 4.4. We now show this connection using matrix notation rather than the component notation used in the theorem and the original description of the algorithm. This helps to clarify the connection and makes generalizations easier.

The first step of Gaussian Elimination of $Ax = b$ transforms

$$A^{(1)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{bmatrix} \quad \text{to} \quad A^{(2)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} \end{bmatrix}$$

This is the same as premultiplying $A^{(1)}$ by M_1 i.e. $A^{(2)} = M_1 A^{(1)}$, where

$$M_1 = \begin{bmatrix} 1 & & & & \\ -m_{21} & 1 & & & \\ -m_{31} & 0 & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ -m_{n1} & 0 & 0 & \cdots & 1 \end{bmatrix}, \quad \text{and} \quad m_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}, \quad i = 2, 3, \dots, n.$$

In general, at stage k we have $A^{(k+1)} = M_k A^{(k)}$ where

$$A^{(k)} = \begin{bmatrix} a_{11}^{(1)} & \cdots & a_{1k}^{(1)} & \cdots & a_{1n}^{(1)} \\ \vdots & \ddots & \vdots & & \vdots \\ 0 & & a_{kk}^{(k)} & \cdots & a_{kn}^{(k)} \\ \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & a_{nk}^{(k)} & \cdots & a_{nn}^{(n)} \end{bmatrix} \quad M_k = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -m_{k+1,k} & & \\ & & \vdots & \ddots & \\ & & -m_{n,k} & & 1 \end{bmatrix}$$

The whole process of elimination is really a sequence of matrix operations on the original matrix where

$$\begin{aligned} A^{(1)} &= A \\ A^{(2)} &= M_1 A^{(1)} \\ A^{(3)} &= M_2 A^{(2)} = M_2 M_1 A \\ &\vdots \\ A^{(n)} &= M_{n-1} A^{(n-1)} = M_{n-1} M_{n-2} \cdots M_2 M_1 A^{(1)} = MA \end{aligned}$$

Thus $A^{(n)} = MA = U$, and the pair of matrices (M, U) is called the *Elimination Form of the Decomposition*.

If we pre-multiply the equation $MA = U$ by M^{-1} we get

$$\begin{aligned} A &= M^{-1}U = (M_{n-1}M_{n-2} \cdots M_2M_1)^{-1}U \\ &= M_1^{-1}M_2^{-1} \cdots M_{n-1}^{-1}U, \\ &= L_1L_2 \cdots L_{n-1}U, \\ &= LU. \end{aligned}$$

Now the inverse of a lower-triangular matrix is lower-triangular and the product of two lower-triangular matrices is lower-triangular. Hence each L_k above is lower-triangular and the product $L_1L_2 \cdots L_{n-1} = L$ is lower-triangular. Thus we have obtained an LU decomposition of A . However, Gaussian Elimination (G.E) computes $M_{n-1}M_{n-2} \cdots M_2M_1 = M$ and not $L_1L_2 \cdots L_{n-1} = L$, and it is this matrix that we need.

How do we get the components of L ? The answer is surprisingly simple : they are calculated by the G.E. algorithm. To see this let us follow G.W Stewart (1996), using a 4×4 matrix A .

We start with

$$A^{(1)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & a_{14}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} & a_{34}^{(1)} \\ a_{41}^{(1)} & a_{42}^{(1)} & a_{43}^{(1)} & a_{44}^{(1)} \end{bmatrix} \quad \text{and} \quad M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -m_{21} & 1 & 0 & 0 \\ -m_{31} & 0 & 1 & 0 \\ -m_{41} & 0 & 0 & 1 \end{bmatrix}$$

Then stage 1 is :

$$A^{(2)} = M_1 A^{(1)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -m_{21} & 1 & 0 & 0 \\ -m_{31} & 0 & 1 & 0 \\ -m_{41} & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & a_{14}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} & a_{34}^{(1)} \\ a_{41}^{(1)} & a_{42}^{(1)} & a_{43}^{(1)} & a_{44}^{(1)} \end{bmatrix} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & a_{14}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & a_{34}^{(2)} \\ 0 & a_{42}^{(2)} & a_{43}^{(2)} & a_{44}^{(2)} \end{bmatrix}$$

Then stage 2 is :

$$A^{(3)} = M_2 A^{(2)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -m_{32} & 1 & 0 \\ 0 & -m_{42} & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & a_{14}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & a_{34}^{(2)} \\ 0 & a_{42}^{(2)} & a_{43}^{(2)} & a_{44}^{(2)} \end{bmatrix} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & a_{14}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)} \\ 0 & 0 & a_{43}^{(3)} & a_{44}^{(3)} \end{bmatrix}.$$

Finally, stage 3 is :

$$A^{(4)} = M_3 A^{(3)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -m_{43} & 1 \end{bmatrix} \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & a_{14}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)} \\ 0 & 0 & a_{43}^{(3)} & a_{44}^{(3)} \end{bmatrix} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & a_{14}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)} \\ 0 & 0 & 0 & a_{44}^{(4)} \end{bmatrix}.$$

Now let us see how L is obtained. First of all we note that M_k^{-1} is M_k with the off-diagonal signs reversed (see the exercise below), i.e., if

$$M_k = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -m_{k+1,k} & & \\ & & \vdots & \ddots & \\ & & -m_{nk} & & 1 \end{bmatrix} \quad \text{then} \quad M_k^{-1} = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & m_{k+1,k} & & \\ & & \vdots & \ddots & \\ & & m_{nk} & & 1 \end{bmatrix}$$

Applying this fact to the 4×4 case above we have

$$\begin{aligned} L &= L_1 L_2 L_3 = M_1^{-1} M_2^{-1} M_3^{-1} \\ &= M_1^{-1} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & m_{32} & 1 & 0 \\ 0 & m_{42} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & m_{43} & 1 \end{bmatrix} = M_1^{-1} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & m_{32} & 1 & 0 \\ 0 & m_{42} & m_{43} & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ m_{21} & 1 & 0 & 0 \\ m_{31} & 0 & 1 & 0 \\ m_{41} & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & m_{32} & 1 & 0 \\ 0 & m_{42} & m_{43} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ m_{21} & 1 & 0 & 0 \\ m_{31} & m_{32} & 1 & 0 \\ m_{41} & m_{42} & m_{43} & 1 \end{bmatrix} \end{aligned}$$

This shows that the G.E multipliers m_{ij} are the elements of the unit lower-triangular matrix L for $i > j$ and that $A = LU$. The derivation above applies to matrices of any order n and we have the result :

Gaussian Elimination applied to the matrix A gives an upper-triangular matrix U and a unit-lower-triangular matrix L where

$$A = LU,$$

and the elements of L are the multipliers m_{ij} calculated by Gaussian Elimination.

Exercise 5.11. Show how the multiplier matrix

$$M_1 = \begin{bmatrix} 1 & & & & \\ -m_{21} & 1 & & & \\ -m_{31} & 0 & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ -m_{n1} & 0 & 0 & \cdots & 1 \end{bmatrix}$$

is obtained from the elementary matrix operator M given in Sec. 5.2.1.

Exercise 5.12. Show that

$$M_{n-1}M_{n-2}\cdots M_2M_1 = M = \begin{bmatrix} 1 & & & & \\ -m_{21} & 1 & & & \\ -m_{31} & -m_{32} & 1 & & \\ \vdots & \vdots & & \ddots & \\ -m_{n1} & -m_{n2} & \cdots & -m_{n,n-1} & 1 \end{bmatrix}$$

Exercise 5.13. Show that $L_k = M_k^{-1}$ is M_k with the signs of the off-diagonals reversed.

Exercise 5.14. Show that

$$L_1L_2\cdots L_{n-1} = L = \begin{bmatrix} 1 & & & & \\ m_{21} & 1 & & & \\ m_{31} & m_{32} & 1 & & \\ \vdots & \vdots & & \ddots & \\ m_{n1} & m_{n2} & \cdots & m_{n,n-1} & 1 \end{bmatrix}$$

Summary

5.6.5 Pivoting and Scaling in Gaussian Elimination

In the preceding discussion of Gaussian Elimination we assumed that $a_{kk}^{(k)} \neq 0$ at each stage of the process. If $a_{kk}^{(k)} = 0$ then we can interchange rows of the matrix $A^{(k)}$ so that $a_{kk}^{(k)} \neq 0$. In fact we need only find a row $i > k$ for which $a_{kk}^{(k)} \neq 0$ and then interchange rows i and k . It can be easily shown that if A is non-singular then such a row exists. Hence, theoretically zero pivots cause no difficulty. The process of interchanging rows (or columns or both) in Gaussian Elimination is called *pivoting*.

However, there is a much more important reason for interchanging rows. Our intuition should warn us that even if $a_{kk}^{(k)} \neq 0$ then a small $a_{kk}^{(k)}$ could cause problems because of roundoff. This is indeed the case as we can see in the following example

Example 5.4 (Need for Pivoting 1). Consider the following set of equations :

$$\begin{aligned} .0001x_1 + 1.00x_2 &= 1.00 \\ 1.00x_1 + 1.00x_2 &= 2.00 \end{aligned}$$

The exact solution is

$$\begin{aligned} x_1 &= \frac{10000}{9999} = 1.00010 \\ x_2 &= \frac{9998}{9999} = 0.99990 \end{aligned}$$

If we perform Gaussian Elimination without interchanges, using 3-digit precision we get

$$\begin{aligned} A^{(2)} = 0.000100x_1 + 1.00x_2 &= 1.00 \\ -10,000x_2 &= -10,000. \end{aligned}$$

Hence $x_2 = 1.00$ and $x_1 = 0.0$. We see that x_2 is accurate (to 3 decimal digits) but x_1 is completely wrong.

If we interchange rows 1 and 2 we get

$$\begin{aligned} A^{(2)} = 1.00x_1 + 1.00x_2 &= 2.00 \\ 1.00x_2 &= 1.00. \end{aligned}$$

Hence $x_2 = 1.00$ and $x_1 = 1.00$. Both of these are accurate to 3 decimal digits.

Notice that the problem is due to roundoff and not any theoretical difficulty. If we had used exact arithmetic then interchanging would not have been necessary.

Example 5.5 (Need for Pivoting 2.). Here is a more general example that clearly shows the need for pivoting when using finite precision arithmetic. The problem is

$$\text{Solve } \begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \epsilon < \epsilon_m = 10^{-16}, \quad \text{using d.p. floating point arithmetic.}$$

Let us find the exact solution first so that we can see what is happening later. Subtracting the second row from the first gives the lower triangular system

$$\begin{bmatrix} \epsilon - 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}.$$

This gives $(\epsilon - 1)x_1 = -1$ or $x_1 = 1/(1 - \epsilon)$. From the second row we have $x_1 + x_2 = 2$. Hence $x_2 = 2 - 1/(1 - \epsilon)$. Thus the exact solution is

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1/(1 - \epsilon) \\ 2 - 1/(1 - \epsilon) \end{bmatrix}.$$

The correctly-rounded exact answer is

$$\text{fl} \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} 1/\text{fl}(1 - \epsilon) \\ 2 - 1/\text{fl}(1 - \epsilon) \end{bmatrix} = \begin{bmatrix} 1/1 \\ 2 - 1/1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \text{because } \epsilon < \epsilon_m.$$

Now we perform Gaussian Elimination without pivoting (row interchanges)

$$A = A^{(1)} = \begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix}, \quad M_1 = \begin{bmatrix} 1 & 0 \\ -1/\epsilon & 1 \end{bmatrix}, \quad A^{(2)} = M_1 A^{(1)} = \begin{bmatrix} \epsilon & 1 \\ 0 & \text{fl}(1 - 1/\epsilon) \end{bmatrix} = \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix}.$$

Thus we have

$$L = M_1^{-1} = \begin{bmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{bmatrix}, \quad \text{and} \quad \hat{U} = A^{(2)} = \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix}.$$

with the hat (^) over U indicating that it is not exact. The product $L\hat{U}$ should give A but

$$L\hat{U} = \begin{bmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{bmatrix} \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix} = \begin{bmatrix} \epsilon & 1 \\ 1 & 0 \end{bmatrix} = \hat{A} \neq A.$$

If we use the factors L and \hat{U} to solve the original problem we get

$$\begin{bmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{bmatrix} \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

Forward substitution gives

$$\begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -1/\epsilon & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{fl}(2 - 1/\epsilon) \end{bmatrix} = \begin{bmatrix} 1 \\ -1/\epsilon \end{bmatrix}.$$

Back substitution on

$$\begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -1/\epsilon \end{bmatrix} \quad \text{gives} \quad \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \begin{bmatrix} (1-1)/\epsilon \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

The second element x_2 is the correctly-rounded exact value but x_1 is has no digits correct and is completely wrong.

If we use pivoting then the steps are

$$A = A^{(1)} = \begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix}, \quad P_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad P_1 A^{(1)} = \begin{bmatrix} 1 & 1 \\ \epsilon & 1 \end{bmatrix}, \quad M_1 = \begin{bmatrix} 1 & 0 \\ -\epsilon & 1 \end{bmatrix},$$

$$A^{(2)} = M_1 P_1 A^{(1)} = \begin{bmatrix} 1 & 1 \\ 0 & \text{fl}(1 - \epsilon) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \hat{U} \quad \text{and} \quad L = M_1^{-1} = \begin{bmatrix} 1 & 0 \\ \epsilon & 1 \end{bmatrix}.$$

Checking that $L\hat{U} = PA$ gives

$$\begin{bmatrix} 1 & 0 \\ \epsilon & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \epsilon & \text{fl}(1 + \epsilon) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \epsilon & 1 \end{bmatrix} = PA.$$

Using these factors we get $LUx = Pb$, or

$$\begin{bmatrix} 1 & 0 \\ \epsilon & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

Forward substitution gives

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\epsilon & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ \text{fl}(1 - \epsilon) \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

Back substitution gives $x_2 = 1$ and then $x_1 = 1$, the correctly-rounded exact answer.

Notice that roundoff errors are still present at various (but different) stages of the solution, but they cause no harm because of the row interchanges.

There is an important observation about \hat{U} that can be generalized :

$$\hat{U} = \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix} \text{ without pivoting, and } \hat{U} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \text{ with pivoting.}$$

We can see that without pivoting the elements of $A \rightarrow \hat{U}$ have grown from $|\epsilon| < 10^{-16}$ in A to $|1/\epsilon| > 10^{16}$ in \hat{U} , whereas there is no growth in \hat{U} with pivoting.

In general, the effects of roundoff can be reduced if the growth of the elements of \hat{U} can be restricted. This is usually, but not always possible, as we will see later.

Partial Pivoting

If A is nonsingular then at each stage k of Gaussian Elimination we are guaranteed that some $a_{ik}^{(k)} \neq 0, i \geq k$.

Partial pivoting finds the row p_k for which $|a_{p_k k}^{(k)}|$ is a maximum and interchanges rows p_k and k . Hence we must add the following statements in the outer loop of the elimination algorithm.

1. Find p_k that maximizes $|a_{p_k k}^{(k)}|, k \leq p_k \leq n$
2. Interchange rows k and p_k

The using the pivot obtained in this way ensures that the multipliers m_{ik} have $|m_{ik}| \leq 1$ and this ensures that the elements of $A^{(k)}$ do not grow with each stage. This in general reduces the effects of roundoff error.

If we are calculating the determinant we must remember that each interchange $p_k \neq k$ causes a change of sign.

We may view the interchange at each stage as a permutation matrix operation.

Thus

$$A^{(k+1)} = M^{(k)} P^{(k)} A^{(k)} = M^{(k)} P^{(k)} M^{(k-1)} P^{(k-1)} \dots M_1 P^{(1)} A^{(1)}$$

If complete rows are interchanged, i.e., multipliers and A -matrix elements, the array calculated by Gaussian Elimination with partial pivoting (GEPP) will contain the LU decomposition of $P_{n-1} \cdots P_2 P_1 A$. Hence we have

$$P_{n-1} \cdots P_2 P_1 A = PA = LU.$$

That is, the sequence of permutation matrices may be viewed as a single permutation matrix operation performed at the start of the elimination. We will show this later, if time permits. This gives the following theorem

Theorem 5.7 (LUP-Decomposition). *If A is an arbitrary square matrix there exists a permutation matrix P , a unit lower triangular matrix L and an upper triangular matrix U such that*

$$LU = PA.$$

However L and U are not always uniquely determined by P and A .

Proof 5.2. (see Forsythe and Moler, Section 16).

We now modify *GaussElimLU* to incorporate these ideas :

algorithm <i>GaussElimLUP</i> (a, n, p)
<p style="margin: 0;">Gaussian Elimination with partial pivoting</p> <pre style="margin: 0;"> for $k := 1$ to $n - 1$ do $p_k := \text{MaxPiv}(a, k, n)$ for $j := 1$ to n do $\text{temp} := a_{kj}$ $a_{kj} := a_{p_k j}$ $a_{p_k j} := \text{temp}$ endfor j for $i := k + 1$ to n do $a_{ik} := a_{ik} / a_{kk}$ for $j := k + 1$ to n do $a_{ij} := a_{ij} - a_{ik} \times a_{kj}$ endfor j endfor i endfor k endalg <i>GaussElimLUP</i> </pre>

The function *MaxPiv*(a, k, n) returns p_k for which $|a_{ik}|$ is a maximum, $k \leq i \leq n$, and the **for** j -loop after this interchanges complete rows k and p_k .

Once we have L , U , and P we can solve $Ax = b$ as follows :

$$Ax = b \rightarrow PAx = Pb \rightarrow LU = Pb$$

The steps in solving $Ax = b$ using *LUP* Decomposition are as follows:

algorithm *SolveLEQP*(a, b, n, x)

1. Calculate P , L and U using *GaussElimLUP*(A, n, P), where $PA = LU$.
2. Calculate $b' = Pb$.
3. Solve $Ly = b'$ using *ForwSubst*(L, b', n, y), where $y = L^{-1}b'$.
4. Solve $Ux = y$ using *BackSubst*(U, y, n, x), where $x = U^{-1}y$.

It should be noted that the permutation matrix P above is represented by the n -vector p which records in p_k the row that was interchanged with row k at stage k . The calculation $b' = Pb$ in step 2 above interchanges the elements of b in the same way that the rows of A were interchanged in step 1.

Complete Pivoting

This is a natural extension of partial pivoting whereby we find p_k and q_k such that $|a_{p_k q_k}| = \max_{k \leq i, j \leq n} |a_{ij}|$

This means that we interchange rows p_k and k and columns q_k and k . The row interchange does not have any effect (theoretically) on the solution but the interchanging the columns interchanges the variable names (labels) i.e. $x_{q_k} \Leftrightarrow x_k$. These interchanges of columns must be recorded so that the correct variable is associated with the corresponding solution value at the end of the algorithm.

Complete pivoting is an $O(n^2)$ process at each stage k . Thus it adds $O(n^3)$ steps to Gaussian Elimination which is a substantial increase although G.E. is still $O(n^3)$.

Complete pivoting is rarely used because it has been found in practice that partial pivoting appears to be entirely adequate.

Conclusion

1. Complete pivoting is safe (proved).
2. Partial pivoting is safe with high probability (experimental).
3. Complete pivoting is not worth it.

Column and Row Scaling

We use the example in Forsythe & Moler, page 40, to demonstrate the effects of scaling.

Consider the previous numerical example (5.55)

$$.0001x_1 + 1.00x_2 = 1.00$$

$$1.00x_1 + 1.00x_2 = 2.00$$

and assume the first equation has been multiplied by 10^5 we get

$$10.0x_1 + 100,000x_2 = 100,000$$

$$1.00x_1 + 1.00x_2 = 2.00$$

Because $10.0 > 1.00$ partial pivoting does no interchange, and we get

$$\begin{aligned}10.0x_1 + 100,000x_2 &= 100,000 \\ -10,000x_2 &= -10,000\end{aligned}$$

$$x_2 = 1.00 \quad x_1 = 0.0 \quad (\text{Bad})$$

If we scale the first equation down by 10^5 and use partial pivoting we get a satisfactory result.

In general scaling is used to eliminate the effect of roundoff by making sure that of all the elements of A are relatively small. If all the numbers are relatively small then the 'grain structure' of the floating point numbers is finer and hence better resolution (accuracy) is to be expected.

In general, therefore, we should try to use numbers x such that $|x| \leq 1$. This will tend to reduce 'loss of significance' errors due to operations on numbers of widely differing magnitudes.

In a similar manner we may scale the columns of the A matrix remembering that this also scales the unknowns x_i and so when the scaled solutions x'_i are found these must be inversely scaled to get the 'correct' values.

When we are scaling a matrix we are interested in getting the magnitudes of the elements close to each other. We can do this by simply altering the exponents which can be more easily performed by shifting than by multiplication or division.

Very little is known about the precise effects of scaling and it is usual to consider only row scaling. If we consider the row scaling performed on the above example we can see that its main effect was to alter the choice of the pivot element.

We conclude that scaling by integer powers of the base b only affects the choice of pivots.

5.7 PERTURBATION ANALYSIS OF LINEAR EQUATIONS

We wish to study the effect on the solution vectors x of changes in A and b , the data of the problem. We will examine perturbations in A and b separately.

5.7.1 Perturbation of b

Let the right-hand-side b be perturbed by δb . Then we want to find the solution of

$$A(x + \delta x) = b + \delta b.$$

We have $A\delta x = \delta b$. Hence

$$\delta x = A^{-1}\delta b \quad \text{and} \quad \|\delta x\| \leq \|A^{-1}\| \|\delta b\|, \quad \text{which is a sharp bound.}$$

Since $b = Ax$ we have $\|b\| \leq \|A\| \|x\|$. Hence

$$\|\delta x\| \|b\| \leq \|A\| \|A^{-1}\| \|x\| \|\delta b\|$$

Defining the **condition number** of A as

$$\boxed{\text{cond}(A) = \|A\| \|A^{-1}\|} \quad (5.55)$$

we get

$$\boxed{\frac{\|\delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|\delta b\|}{\|b\|}} \quad (5.56)$$

Here, we may interpret $\text{cond}(A)$ as the amount a relative error in b is magnified in the solution vector x .

The condition number depends on the norm used and is bounded below by 1. This is shown by noting that $\|I\| = 1$ for any norm and

$$1 = \|I\| = \|AA^{-1}\| \leq \|A\| \|A^{-1}\| = \text{cond}(A).$$

The condition number is also bounded below by

$$1 \leq \frac{|\lambda_{\max}|}{|\lambda_{\min}|} = \text{cond}^*(A) \leq \text{cond}(A)$$

for any norm.

A matrix A which has a large condition number is said to be ill-conditioned.

Example 5.6. Ill-Conditioned Matrix

$$A = \begin{bmatrix} 1 & .99 \\ .99 & .98 \end{bmatrix} \quad \lambda_1 = 1.98 \quad \lambda_2 = -0.00005 \quad \text{cond}^*(A) = 39,600$$

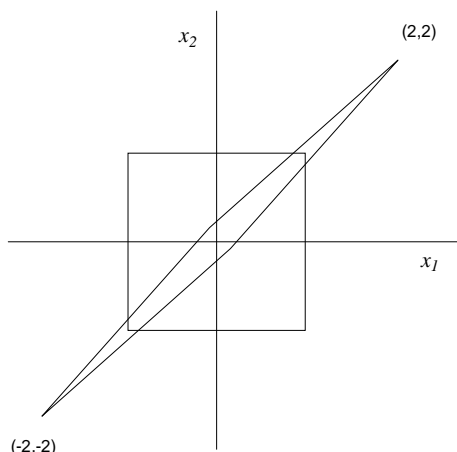


Figure 5.3 : Flattening of a Square by an Ill-Conditioned Matrix

Hence this matrix is very ill-conditioned.

Consider the system

$$\begin{aligned}x_1 + 0.99x_2 &= 1.99 \\0.99x_1 + 0.98x_2 &= 1.97\end{aligned}$$

The true solution is $x_1 = 1$ and $x_2 = 1$ but $x_1 = 3.0000$ and $x_2 = -1.0203$ gives

$$\begin{aligned}x_1 + 0.99x_2 &= 1.989903 \\0.99x_1 + 0.98x_2 &= 1.970106\end{aligned}$$

If the square (ball) $\|x\|_\infty = 1$ is transformed by A , i.e., $\{Ax : \|x\|_\infty = 1\}$ then we can see the enormous distortion that A causes by considering how the corners of the square that are transformed by A .

We can see in Figure 5.2 that A has ‘flattened’ the square $\{x : \|x\|_\infty = 1\}$.

Popular Misconception

It is very often said that a very small determinant is a clear indication of an ill-conditioned matrix. This is certainly not so. Nor does the order n cause difficulty. These ideas can be refuted by the following obvious and simple counter example

Let $A_{n \times n} = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$, with $\lambda_i = \frac{1}{10}$. Then $\det(A) = (\frac{1}{10})^n$ which is small for even moderate size n . Gaussian Elimination can solve this problem easily without even pivoting. The solution is $x_i = b_i / \lambda = 10b_i$ and

$$\text{cond}(A) = \frac{\max |\lambda|}{\min |\lambda|} = \frac{\lambda}{\lambda} = 1,$$

the lowest possible condition number.

5.7.2 Perturbation of A

If A is perturbed by δA then we have

$$\begin{aligned}(A + \delta A)(x + \delta x) &= b \\ Ax + A\delta x + \delta Ax + \delta A\delta x &= b \\ A\delta x &= -\delta A(x + \delta x) \\ \delta x &= -A^{-1}\delta A(x + \delta x)\end{aligned}$$

Taking norms and using the triangle inequality we have

$$\begin{aligned}\|\delta x\| &= \|A^{-1}\delta A(x + \delta x)\| \\ &\leq \|A^{-1}\| \|\delta A\| (\|x\| + \|\delta x\|) \\ &= \|A^{-1}\| \|\delta A\| \|x\| + \|A^{-1}\| \|\delta A\| \|\delta x\| \\ (1 - \|A^{-1}\| \|\delta A\|) \|\delta x\| &\leq \|A^{-1}\| \|\delta A\| \|x\|\end{aligned}$$

Re-arranging the last inequality gives

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\|A^{-1}\| \|\delta A\|}{(1 - \|A^{-1}\| \|\delta A\|)} = \frac{\|A\| \|A^{-1}\|}{(1 - \|A^{-1}\| \|\delta A\|)} \frac{\|\delta A\|}{\|A\|} \quad (5.57)$$

If $\|A^{-1}\| \|\delta A\| \ll 1$ we have

$$\boxed{\frac{\|\delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|\delta A\|}{\|A\|}} \quad (5.58)$$

Theorem 5.8 (Distance to a Singular Matrix). *If A is non-singular, and*

$$\frac{\|\delta A\|}{\|A\|} < \frac{1}{\text{cond}(A)} \quad (5.59)$$

then $A + \delta A$ is also non-singular.

This theorem tells us the condition number measures the distance from A to the nearest singular matrix. If $\text{cond}(A)$ is large then A is close to a singular matrix, and *vice versa*.

5.7.3 Errors and Residuals

In many applications of linear equations we want to solve $Ax = b$ so that the difference between the left- and right-hand sides is small, i.e., so that $\|r\| = \|b - Ax\|$ is small. The term r is called the **residual**. We now examine the relationship between r and errors in x . Let \hat{x} be the

computed solution to $Ax = b$. Then

$$\begin{aligned}
 \delta x &= x - \hat{x} \\
 r &= b - A\hat{x} \\
 A\delta x &= Ax - A\hat{x} = b - A\hat{x} = r \\
 \delta x &= A^{-1}r \\
 \|\delta x\| &\leq \|A^{-1}\| \|r\| \\
 \frac{\|\delta x\|}{\|x\|} &\leq \|A^{-1}\| \frac{\|r\|}{\|x\|} \\
 &= \|A\| \|A^{-1}\| \frac{\|r\|}{\|A\| \|x\|} \\
 &\leq \|A\| \|A^{-1}\| \frac{\|r\|}{\|Ax\|} \\
 &= \|A\| \|A^{-1}\| \frac{\|r\|}{\|b\|}
 \end{aligned}$$

Thus

$$\boxed{\frac{\|\delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|r\|}{\|b\|}} \quad (5.60)$$

If A is ill-conditioned then small $\|r\|$ does not imply small $\|\delta x\|/\|x\|$.

LUP Decomposition tends to give solutions with small residuals but can give large errors in \hat{x} if A is ill-conditioned.

Example 5.7 (Residuals and Errors — Which solution is better?). Consider the following simple-looking system of two equations from (Van Loan, 2000, page 234) :

$$\begin{bmatrix} 0.780 & 0.563 \\ 0.913 & 0.659 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.217 \\ 0.254 \end{bmatrix}$$

Suppose we have obtained two solutions, calculated by different methods :

$$\tilde{x} = \begin{bmatrix} 0.341 \\ -0.087 \end{bmatrix}, \quad \hat{x} = \begin{bmatrix} 0.999 \\ -1.00 \end{bmatrix}$$

Which is the better solution? Let us compute the residuals, because in general we will not know the exact solution,

$$b - A\tilde{x} = \begin{bmatrix} 0.000001 \\ 0 \end{bmatrix}, \quad b - A\hat{x} = \begin{bmatrix} 0.000780 \\ 0.000913 \end{bmatrix}$$

Calculating the exact solution we get $x = [1 \ -1]^T$, and we get the forward errors

$$x - \tilde{x} = \begin{bmatrix} 0.659 \\ -0.913 \end{bmatrix}, \quad x - \hat{x} = \begin{bmatrix} 0.001 \\ 0 \end{bmatrix}$$

Thus we see that \tilde{x} gives a small residual and a large forward error while \hat{x} gives a large residual and small forward error. The user must choose between these solutions, based on the problem being solved.

Example 5.8 (A Russian Matrix). This innocent-looking symmetric matrix comes from the book by Faddeev & Faddeeva, 1960.

$$A = \begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix}$$

The $LU = PA$ decomposition is

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.8571429 & 1 & 0 & 0 \\ 0.7142857 & 0.25 & 1 & 0 \\ 0.7142857 & 0.25 & -0.2 & 1 \end{bmatrix},$$

$$U = \begin{bmatrix} 7 & 10 & 8 & 7 \\ 0 & -0.5714286 & 3.1428571 & 3 \\ 0 & 0 & 2.5 & 4.25 \\ 0 & 0 & 0 & 0.1 \end{bmatrix},$$

and $p^T = [2 \ 3 \ 4 \ 1]$.

Now let us solve the system $Ax = b = [23 \ 32 \ 33 \ 31]^T$. The exact answer is $x = [1 \ 1 \ 1 \ 1]^T$. Let \hat{x} denote the calculated answer. The steps are

$$PAx = Pb = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 23 \\ 32 \\ 33 \\ 31 \end{bmatrix} = \begin{bmatrix} 32 \\ 33 \\ 31 \\ 23 \end{bmatrix}$$

Substitute LU for PA and we get $LUx = Pb$, i.e.,

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.8571429 & 1 & 0 & 0 \\ 0.7142857 & 0.25 & 1 & 0 \\ 0.7142857 & 0.25 & -0.2 & 1 \end{bmatrix} \begin{bmatrix} 7 & 10 & 8 & 7 \\ 0 & -0.5714286 & 3.1428571 & 3 \\ 0 & 0 & 2.5 & 4.25 \\ 0 & 0 & 0 & 0.1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 32 \\ 33 \\ 31 \\ 23 \end{bmatrix}$$

Solving $Ly = Pb = b'$ by Forward Substitution gives

$$y = \begin{bmatrix} 32 \\ 5.5714286 \\ 6.75 \\ 0.1 \end{bmatrix}$$

Finally we solve $Ux = y$ by Back Substitution to get :

$$\hat{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix},$$

and so $\hat{x} = x$, a perfect answer.

Let us calculate $\text{cond}(A) = \|A\|_1 \|A^{-1}\|_1$.

$$\|A\|_1 = \left\| \begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix} \right\|_1 = 30.3, \text{ and } \|A^{-1}\|_1 = \left\| \begin{bmatrix} 68 & -1 & -17 & 10 \\ -41 & 25 & 10 & -6 \\ -17 & 10 & 5 & -3 \\ 10 & -6 & -3 & 2 \end{bmatrix} \right\|_1 = 98.5.$$

Hence,

$$\text{cond}(A) = \|A\|_1 \|A^{-1}\|_1 = 30.3 \times 98.5 = 2984.1$$

Using the theorem on page 5.27 which says that if $\|\delta A\| \geq \|A\|/\text{cond}(A)$ then $A + \delta A$ is singular, we see that if A is perturbed by δA such that

$$\|\delta A\| = 30.3/2984.1 = 0.0102,$$

then $A + \delta A$ will be singular.

Let

$$A(\epsilon) = \begin{bmatrix} 5 + \epsilon & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix}, \quad \text{i.e.,} \quad \delta A = \begin{bmatrix} \epsilon & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

It can be shown that (using *Maxima*, *Maple*, *MuPad*)

$$\det(A) = 1.0 \quad \text{and that} \quad \det(A(\epsilon)) = 1 + 68\epsilon.$$

If we choose $\epsilon = -1/68$ then $\det(A(\epsilon)) = 0$ and $A(\epsilon)$ will be singular.

Let us choose various values of ϵ and see how the system $A(\epsilon)x = b$ responds.

With $\epsilon = -0.02$ we get

$$A(\epsilon) = \begin{bmatrix} 4.98 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix} \quad \text{and} \quad A^{-1}(\epsilon) = \begin{bmatrix} -188.9 & 113.9 & 47.2 & -27.8 \\ 113.9 & -68.4 & -28.7 & 16.8 \\ 47.2 & -28.7 & -11.1 & 6.4 \\ -27.8 & 16.8 & 6.4 & -3.6 \end{bmatrix}.$$

We have $\|A(\epsilon)\|_1 = 33$, $\|A^{-1}(\epsilon)\|_1 = 378$ and $\text{cond}(A(\epsilon)) = 12467$. This is an over-estimate because $\text{cond}^* A(\epsilon) = 8279$.

The solution of $A(\epsilon)x = b$ is

$$\hat{x} = \begin{bmatrix} -2.8 \\ 3.3 \\ 1.9 \\ 0.4 \end{bmatrix} \quad \text{and} \quad \frac{\|x - \hat{x}\|}{\|x\|} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} -2.8 \\ 3.3 \\ 1.9 \\ 0.4 \end{bmatrix} = \begin{bmatrix} 3.8 \\ 2.3 \\ 0.9 \\ 0.6 \end{bmatrix}$$

Thus we see that the relative errors in the components range from 380% to 60%.

With $\epsilon = -1/68$ we get

$$A(\epsilon) = \begin{bmatrix} 4.9852941 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix} \quad \text{and} \quad A^{-1}(\epsilon) = \begin{bmatrix} 29 & -17 & -7 & 4 \\ -17 & 10 & 4 & -3 \\ -7 & 41 & 2 & -1 \\ 4 & -3 & -1 & 1 \end{bmatrix} \times 10^{14}.$$

We have $\|A(\epsilon)\|_1 = 30$, $\|A^{-1}(\epsilon)\|_1 = 378 \times 10^{14}$ and $\text{cond}^*(A(\epsilon)) = 4 \times 10^{15}$.

The solution of $A(\epsilon)x = b$ is

$$\hat{x} = 10^{12} \times \begin{bmatrix} 42.8 \\ -25.8 \\ -10.7 \\ 6.3 \end{bmatrix}$$

It is instructive to look at the factors calculated for this matrix. With $\epsilon = -1/68$ we get the factors

$$L(\epsilon) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.8571429 & 1 & 0 & 0 \\ 0.7142857 & 0.25 & 1 & 0 \\ 0.7121849 & 0.2132353 & -0.1470588 & 1 \end{bmatrix}, \quad U(\epsilon) = \begin{bmatrix} 7 & 10 & 8 & 7 \\ 0 & -0.5714286 & 3.1428571 & 3 \\ 0 & 0 & 2.5 & 4.25 \\ 0 & 0 & 0 & 2.331 \times 10^{-15} \end{bmatrix}$$

If we compare these with the factors for the original matrix we see where the trouble lies :

$$L(0) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.8571429 & 1 & 0 & 0 \\ 0.7142857 & 0.25 & 1 & 0 \\ 0.7142857 & 0.25 & -0.2 & 1 \end{bmatrix}, \quad U(0) = \begin{bmatrix} 7 & 10 & 8 & 7 \\ 0 & -0.5714286 & 3.1428571 & 3 \\ 0 & 0 & 2.5 & 4.25 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}.$$

The matrices $U(0)$ and $U(\epsilon)$ are the same except for the element $u_{44}(\epsilon) = 2.331 \times 10^{-15}$, which is numerically 0 compared to the other elements of the matrix, and so $U(\epsilon)$ is numerically singular to machine double precision. The first element of \hat{x} that is calculated is

$$\hat{x}_4 = y_4 / u_{44} = 0.0147 / (2.331 \times 10^{-15}) = 6.3 \times 10^{12},$$

and this is used in subsequent calculations, causing the other components of \hat{x} to acquire huge values.

This is a dramatic demonstration of how a small relative change in the matrix A ($\epsilon \approx 10^{-2}$) results in an enormous change in the solution ($\delta x(\epsilon) \approx 10^{14}$). The calculations with the original matrix gave the exact answer yet the calculations on the slightly-perturbed matrix gives a meaningless result.

WARNING! The determinant of $A(\epsilon)$ is

$$\det A(\epsilon) = \det U(\epsilon) = -0.233100011655 \times 10^{-13}$$

The fact that the determinant of $A(\epsilon)$ has a 'small' value does not, by itself, indicate near-singularity. In general the value of $\det(A)$ tells us nothing about the closeness of A to singularity.

5.7.4 Iterative Refinement of Direct Solutions

This method, also called Residual Correction, assumes that we have solved $Ax = b$ by some direct method — in particular $LU = PA$ Decomposition — and obtained an approximate solution $\hat{x}^{(1)}$.

The error in the solution is

$$\delta x^{(1)} = x - \hat{x}^{(1)} \quad \text{and the residual is} \quad r^{(1)} = b - A\hat{x}^{(1)}$$

Multiplying the first equation by A we get

$$A\delta x^{(1)} = Ax - A\hat{x}^{(1)} = b - A\hat{x}^{(1)} = r^{(1)}$$

If we solve $A\delta x^{(1)} = r^{(1)}$ for $\delta x^{(1)}$ and get $\delta\hat{x}^{(1)}$ then we should expect that

$$\hat{x}^{(2)} = \hat{x}^{(1)} + \delta\hat{x}^{(1)}$$

to be a better approximation to x . Indeed, if all the calculations above were performed in exact arithmetic, then $x^{(2)}$ would be the exact solution to $Ax = b$.

We can repeat this process and compute a new residual

$$r^{(2)} = b - A\hat{x}^{(2)} \quad \text{and solve} \quad A\delta x^{(2)} = r^{(2)}$$

This gives the following algorithm :

algorithm ItRefine($A, b, L, U, P, x_0, \text{maxits}$)

for $k := 1$ **to** maxits **do**

$r^{(k)} := b - A\hat{x}^{(k)}$ using $2p$ - digit precision

Solve $LU\delta x^{(k)} = Pr^{(k)}$ using p - digit precision

$\hat{x}^{(k+1)} := \hat{x}^{(k)} + \delta\hat{x}^{(k)}$ using $2p$ - digit precision

endfor

endalg Power

We need the original A matrix to calculate $r^{(k)}$ but we can solve for each $\delta x^{(k)}$ using *ForwSubst* and *BackSubst* in $O(n^2)$ operations because we have PLU from the decomposition. Thus each iteration of this algorithm costs $O(n^2)$ ops.

The residual r^k needs to be computed with higher precision because catastrophic cancellation is likely when subtracting the two close vectors b and $A\hat{x}^k$. In addition, we note that the error and the residual are related by

$$\frac{\|\delta x\|}{\|x\|} \leq \|A\| \|A^{-1}\| \frac{\|r\|}{\|b\|} = \text{cond}(A) \frac{\|r\|}{\|b\|}$$

If A is ill-conditioned then $\text{cond}(A)$ will be large and any error in the calculation of r^k will be magnified. Because of this it is absolutely essential to calculate r^k using double the working precision. The need for two precisions is the main disadvantage of this method.

Forsythe and Moler, (1967, page 50), have shown that after the first iteration of Iterative Refinement we have

$$\delta x^{(1)} \approx \text{cond}(A) \|x^{(1)}\| b^{-p}, \quad (5.61)$$

where b is the base of the number system used and p is the number of digits of precision. If $\text{cond}(A) \approx b^k$ then

$$\delta x^{(1)} \approx \|x^{(1)}\| b^{k-p}$$

If $\text{cond}(A)$ is so large that $k \geq p$ then the error is greater than the exact solution and no solution has any meaning. In the Russian matrix example, $\text{cond}A(\epsilon) = 10^{15}$ and so $k = 15$. Even using IEEE double precision with $p \approx 15$ we get $\delta x \approx \|x\| \beta^0 = \|x\|$, i.e., a relative error of 100%, and so no digits in the solution are correct. In this case Iterative Refinement is no use and we must resort to higher precision decomposition.

We can estimate $\text{cond}(A)$ by re-arranging equation (5.61)

$$\text{cond}(A) \simeq b^p \frac{\|\delta \hat{x}^{(1)}\|}{\|\hat{x}^{(1)}\|}. \quad (5.62)$$

This implies that if Iterative Refinement does not give a small relative error immediately then the matrix is ill-conditioned.

SUMMARY OF PERTURBATION ANALYSIS

$$\text{cond}(A) = \|A\| \|A^{-1}\|$$

$$\frac{\|\delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|\delta b\|}{\|b\|}$$

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\|A^{-1}\| \|\delta A\|}{(1 - \|A^{-1}\| \|\delta A\|)} = \frac{\|A\| \|A^{-1}\|}{(1 - \|A^{-1}\| \|\delta A\|)} \frac{\|\delta A\|}{\|A\|}$$

$$\frac{\|\delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|\delta A\|}{\|A\|}, \quad \|A^{-1}\| \|\delta A\| \ll 1$$

$$\frac{\|\delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|r\|}{\|b\|}$$

After the first iteration of Iterative Refinement we have

$$\text{cond}(A) \simeq b^p \frac{\|\delta \hat{x}^{(1)}\|}{\|\hat{x}^{(1)}\|}$$

Theorem 5.9 (Distance to a Singular Matrix). *If A is non-singular, and*

$$\frac{\|\delta A\|}{\|A\|} < \frac{1}{\text{cond}(A)}, \quad \text{then } A + \delta A \text{ is also non-singular}$$