Chapter 5
Linear Algebraic Systems

5.1 Introduction

The problem of finding the solution of n linear equations in m unknowns is found in all areas of engineering, mathematics and science. Recall that matrix multiplication was defined with systems in mind; thus, the equations

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1m}x_m &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2m}x_m &= b_2 \\ \vdots \qquad\qquad & \qquad\qquad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nm}x_m &= b_n \end{aligned} \tag{5.1}$$

may be written in concise matrix form $\boldsymbol{Ax} = \boldsymbol{b}$. Recall that $\boldsymbol{x}$ is a column vector containing the m unknown values. A solution to (5.1) is a set of values $x_1, x_2, \ldots, x_m$ which substituted on the left hand side will give the values b_1, b_2,..., b_n on the right hand side.

If (5.1) has one or more solutions we have a *consistent* system. On the other hand (5.1) may not have a solution in which case we say that the system is *inconsistent*. For example, the system

$$\begin{aligned} x + 2y &= 3 \\ 2x + 4y &= 7 \end{aligned}$$

is inconsistent. (Why?) Although there are more equations than unknowns in

$$\begin{aligned} x - y &= 2 \\ 2x - 2y &= 4 \\ -x + y &= -2 \end{aligned}$$

we have a consistent system with the solution: $x = 2 + y$ for any arbitrary value of y.

In the simple case of two unknowns, each equation represents a line in the xy-plane leading to a convenient geometric interpretation of the system. It is an easy task to itemize the possibilities: all lines may intersect at a common point – providing a unique solution, two of the lines may be parallel with different y-intercepts – no solution, or all of the lines are parallel with identical y-intercepts – infinitely many solutions. The second and third cases are represented in the examples. Equations with three unknowns may be interpreted in terms of planes in three dimensional space.

The numerous possibilities associated with the problem of n linear equations in m unknowns are treated in most linear algebra texts. We shall consider the standard situation of n linear equations in n unknowns. In this case the coefficient matrix $\boldsymbol{A}$ will be $n \times n$.

5.2 Gaussian Elimination

Frequently, students learn to solve linear systems by using *elimination.* The idea is simple. Solve the first equation in (5.1) for x_1, with $n = m$,

$$x_1 = -\frac{a_{12}}{a_{11}}x_2 - \frac{a_{13}}{a_{11}}x_3 - \ldots - \frac{a_{1n}}{a_{11}}x_n + \frac{b_1}{a_{11}}.$$

Next, substitute for x_1 in the remaining equations, 2 to n, effectively eliminating x_1 from these equations. The process works as long as a_{11} is different from zero. After eliminating x_1, the scheme continues by solving the updated second equation in (5.1) for x_2 and then substituting for x_2 in equations 3 to n, and so forth. Eventually, we arrive at the last equation which contains only the unknown x_n. Without complications along the way, the system of equations will be reduced to the upper triangular form

$$\begin{aligned} 1x_1 + \overline{a}_{12}x_2 + \overline{a}_{13}x_3 + \ldots + \overline{a}_{1n}x_n &= \beta_1 \\ 1x_2 + \overline{a}_{23}x_3 + \ldots + \overline{a}_{2n}x_n &= \beta_2 \\ 1x_3 + \ldots + \overline{a}_{3n}x_n &= \beta_3 \\ \vdots \quad &= \quad \vdots \\ 1x_n &= \beta_n \end{aligned} \tag{5.2}$$

The $\overline{a}_{ij}$'s and β_i's are combinations of the original a_{ij}'s and b_i's. With x_n known from the last equation, we work our way back by finding, in order, $x_{n-1}, x_{n-2}, \ldots, x_2$ and eventually x_1. This process is called *back substitution*. In matrix form, (5.2) is $\boldsymbol{Ux = \beta}$ where $\boldsymbol{U}$ is an upper triangular matrix with ones on the main diagonal. Consider a 3×3 example.

$$\begin{aligned} -1x_1 + 2x_2 - 3x_3 &= 3 \\ 2x_1 - 1x_2 + 5x_3 &= 1 \\ 3x_1 - 3x_2 + 10x_3 &= 2 \end{aligned} \tag{5.3}$$

Solving the first of equations (5.3) gives $x_1 = 2x_2 - 3x_3 - 3$. Substituting into the second and third equations of (5.3) we find the system

$$\begin{aligned} 1x_1 - 2x_2 + 3x_3 &= -3 \\ 3x_2 - 1x_3 &= 7 \\ 3x_2 + 1x_3 &= 11 \end{aligned} \tag{5.4}$$

The second equation produces $x_2 = \frac{1}{3}x_3 + \frac{7}{3}$. Substituting into the third equation in (5.4) gives

$$\begin{aligned} 1x_1 - 2x_2 + 3x_3 &= -3 \\ 1x_2 - \tfrac{1}{3}x_3 &= \tfrac{7}{3} \\ 2x_3 &= 4 \end{aligned} \tag{5.5}$$

Finally, dividing the last equation in (5.5) by 2 produces the desired form (5.2), $\boldsymbol{Ux = \beta}$. The back substitution phase is very simple showing $x_3 = 2$, $x_2 = 3$, and $x_1 = -3$.

The *Gaussian elimination* algorithm is a matrix method to implement the elimination procedure. It should be clear that the symbols used for the unknowns, the x_i's, are not critical.

The important information is contained in the square matrix, $\boldsymbol{A}$, and the column vector, $\boldsymbol{b}$. The actual solution to the system depends on the known values: the a_{ij}'s and the b_i's. As we shall see, the German mathematician Carl Friedrich Gauss (1777-1855) made many contributions to numerical methods.

Instead of working with equations, the Gaussian algorithm manipulates the rows of an *augmented* matrix constructed from $\boldsymbol{A}$ and $\boldsymbol{b}$, by appending the column $\boldsymbol{b}$ to the right side of $\boldsymbol{A}$ giving $[\,\boldsymbol{A}\,|\,\boldsymbol{b}\,]$, a matrix with n rows and $n+1$ columns. Using *elementary row operations* the augmented matrix $[\,\boldsymbol{A}\,|\,\boldsymbol{b}\,]$ is transformed to $[\,\boldsymbol{U}\,|\,\boldsymbol{\beta}\,]$ where $\boldsymbol{U}$ is an upper triangular matrix and $\boldsymbol{\beta}$ is a column matrix or vector, see (5.2). In most versions of Gaussian elimination the main diagonal of $\boldsymbol{U}$ may contain values other than ones. Elementary row operations are limited to

1. Interchange any two rows, $\mathrm{R}_i \leftrightarrow \mathrm{R}_j$.
2. Multiply any row by a constant $\mathrm{R}_i \leftarrow c\mathrm{R}_i$.
3. Add to any row a multiple of another row, $\mathrm{R}_j \leftarrow \mathrm{R}_j + c\mathrm{R}_i$.

It should be understood that the row operations are computed entry by entry within the row.

The Gaussian algorithm will transform the original problem, $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, to the equivalent system $\boldsymbol{U}\boldsymbol{x} = \boldsymbol{\beta}$. Back substitution begins with row n of $[\,\boldsymbol{U}\,|\,\boldsymbol{\beta}\,]$, which will contain the coefficients of the last equation in $\boldsymbol{U}\boldsymbol{x} = \boldsymbol{\beta}$, that is $u_{nn}x_n = \beta_n$.

Implementing the algorithm is more complicated that it initially appears. The sequence of row operations transforming $\boldsymbol{A}$ to $\boldsymbol{U}$ will require logical tests to prevent division by zero along the way plus considerable algebra. To understand the issues, consider a 3×4 augmented matrix.

$$[\,\boldsymbol{A}\,|\,\boldsymbol{b}\,] = \begin{bmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{bmatrix} \tag{5.6}$$

Assuming that a_{11} is not zero, the first step of the elimination scheme is designed to eliminate x_1 from the second and third equations. In other words, the entries a_{21} and a_{31} should be replaced by zeros. This may be accomplished by the row operations

$$\mathrm{R}_2 \leftarrow \mathrm{R}_2 - \frac{a_{21}}{a_{11}}\mathrm{R}_1 \text{ and } \mathrm{R}_3 \leftarrow \mathrm{R}_3 - \frac{a_{31}}{a_{11}}\mathrm{R}_1. \tag{5.7}$$

The ratios $\frac{a_{21}}{a_{11}} = m_{21}$ and $\frac{a_{31}}{a_{11}} = m_{31}$ are usually called the *multipliers*.

After the two row operations, we find

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ 0 & \widehat{a}_{22} & \widehat{a}_{23} & \widehat{b}_2 \\ 0 & \widehat{a}_{32} & \widehat{a}_{33} & \widehat{b}_3 \end{bmatrix}. \tag{5.8}$$

The next step will replace the position occupied by $\widehat{a}_{32}$ with zero. Assuming that $\widehat{a}_{22} \neq 0$, the row operation $\mathrm{R}_3 \leftarrow \mathrm{R}_3 - \frac{\widehat{a}_{32}}{\widehat{a}_{22}}\mathrm{R}_2$ will accomplish the task. The multiplier is $m_{32} = \frac{\widehat{a}_{32}}{\widehat{a}_{22}}$.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ 0 & \widehat{a}_{22} & \widehat{a}_{23} & \widehat{b}_2 \\ 0 & 0 & \overline{a}_{33} & \overline{b}_3 \end{bmatrix} = [\,\boldsymbol{U}\,|\,\boldsymbol{\beta}\,]. \tag{5.9}$$

Entries with identical row and column indices are usually called *pivot* elements. To arrive at (5.9) we have assumed that both pivot elements, a_{11} and $\widehat{a}_{22}$, were different from zero. If a pivot element happens to be zero the Gaussian algorithm calls for a row interchange with a row below the pivot row according to a prescribed criterion.

Consider the augmented matrix for the system given in (5.3). As we shall see, pivoting is not necessary for this example.

$$\begin{bmatrix} -1 & 2 & -3 & 3 \\ 2 & -1 & 5 & 1 \\ 3 & -3 & 10 & 2 \end{bmatrix}$$

Since $a_{11} = -1$ is not zero, the row operations in (5.7) become

$$\mathrm{R}_2 \leftarrow \mathrm{R}_2 - \frac{2}{(-1)}\mathrm{R}_1 \text{ and } \mathrm{R}_3 \leftarrow \mathrm{R}_3 - \frac{3}{(-1)}\mathrm{R}_1. \tag{5.10}$$

$$\begin{bmatrix} -1 & 2 & -3 & 3 \\ 0 & 3 & -1 & 7 \\ 0 & 3 & 1 & 11 \end{bmatrix}$$

With $\widehat{a}_{22} = 3$, a final row operation $\mathrm{R}_3 \leftarrow \mathrm{R}_3 - \frac{3}{3}\mathrm{R}_2$ will produce the $[\, \boldsymbol{U} \,|\, \boldsymbol{\beta} \,]$ form.

$$\begin{bmatrix} -1 & 2 & -3 & 3 \\ 0 & 3 & -1 & 7 \\ 0 & 0 & 2 & 4 \end{bmatrix}$$

You should note the similarities between Gaussian elimination with equations and the use of row operations on the augmented matrix.

Pivoting Criteria

If a pivot element happens to be zero we must interchange rows of the augmented matrix. Implementing the Gaussian algorithm via hand computations, allows for a simple approach. Prior to any row operations, inspect the entries in the first column, the a_{i1}'s. Find the first non zero entry, say $i = k$ and perform the row interchange $\mathrm{R}_1 \leftrightarrow \mathrm{R}_k$. Keep in mind that k may be 1 so that an interchange is not necessary. We shall refer to this scheme as *basic* pivoting. Following the row interchange the new a_{11} will be non zero and can be used in the row operations $\mathrm{R}_i \leftarrow \mathrm{R}_i - \frac{a_{i1}}{a_{11}}\mathrm{R}_1$, $i = 2, 3, \ldots, n$.

The process may be repeated by inspecting the new entries in the second column – below the first row – $\widehat{a}_{i2}$ for $i = 2, 3, \ldots, n$. Find the first non zero entry, say $i = j$, and make the row interchange $\mathrm{R}_2 \leftrightarrow \mathrm{R}_j$.

Basic pivoting is adequate for hand computation; however, Gaussian elimination is usually implemented on computers with finite precision. To improve numerical accuracy various pivoting criteria have been developed. One of the most common schemes is the so-called *partial* pivoting algorithm.

Partial pivoting begins by finding the largest element in absolute value in the first column. In other words if $\max |a_{i1}|$ occurs when $i = k$, rows 1 and k are interchanged, $R_1 \leftrightarrow R_k$. In the case of ties, select the smallest row index. Following the Gaussian row operations $R_i \leftarrow R_i - \frac{a_{i1}}{a_{11}} R_1$, $i = 2, 3, \ldots, n$, the process is repeated using column two. Identify the row, starting with row 2, in which the absolute value of $\widehat{a}_{i2}$, $i = 2, 3, \ldots, n$, is the largest. If the maximum occurs in row j, $R_2 \leftrightarrow R_j$. Continue until the procedure terminates.

MATLAB includes a command to find the value and index of the largest entry in a vector. The command **max** will perform the task. Consider the vector $[\,1, 3, -5, 2, 0\,]$ and the MATLAB examples

```
>> [value,index]=max([1,3,-5,2,0])
value =
    3
index =
    2
>> [value,index]=max(abs([1,3,-5,2,0]))
value =
    5
index =
    3
```

To illustrate the use of partial pivoting, consider the linear system represented by the augmented matrix

$$[\,\boldsymbol{A} \mid \boldsymbol{b}\,] = \begin{bmatrix} 2 & 6 & 10 & 0 \\ 1 & 3 & 3 & 2 \\ 4 & 14 & 28 & -8 \end{bmatrix}. \tag{5.11}$$

The largest entry in the first column is the value | 4 | in row 3. Thus $R_1 \leftrightarrow R_3$:

$$\boldsymbol{M}_1 = \begin{bmatrix} 4 & 14 & 28 & -8 \\ 1 & 3 & 3 & 2 \\ 2 & 6 & 10 & 0 \end{bmatrix}.$$

Using the three row operations $\begin{cases} R_1 \leftarrow R_1 - \frac{0}{4} R_1 \\ R_2 \leftarrow R_2 - \frac{1}{4} R_1 \\ R_3 \leftarrow R_3 - \frac{2}{4} R_1 \end{cases}$ gives

$$\boldsymbol{M}_2 = \begin{bmatrix} 4 & 14 & 28 & -8 \\ 0 & -0.5 & -4 & 4 \\ 0 & -1 & -4 & 4 \end{bmatrix}.$$

The first row operation may seem unnecessary; however, we shall soon see why it is included. Matrix algebra and MATLAB may be used to perform the row operations. The first row of $\boldsymbol{M}_1$ is selected by the matrix product $[\,1\,0\,0\,]\,\boldsymbol{M}_1$. In turn, this product may be multiplied by a column vector of multipliers $[\,\frac{0}{4}\ \frac{1}{4}\ \frac{2}{4}\,]^T$ to produce a 3×4 matrix

$$\begin{bmatrix} \frac{0}{4} \\ \frac{1}{4} \\ \frac{2}{4} \end{bmatrix} [\,1 \quad 0 \quad 0\,]\, \boldsymbol{M}_1 = \begin{bmatrix} \frac{0}{4} \\ \frac{1}{4} \\ \frac{2}{4} \end{bmatrix} [\,4 \quad 14 \quad 28 \quad -8\,] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{4}\cdot 4 & \frac{1}{4}\cdot 14 & \frac{1}{4}\cdot 28 & \frac{1}{4}\cdot -8 \\ \frac{2}{4}\cdot 4 & \frac{2}{4}\cdot 14 & \frac{2}{4}\cdot 28 & \frac{2}{4}\cdot -8 \end{bmatrix}. \qquad (5.12)$$

Since row one is unchanged, a multiplier of zero, $\frac{0}{4}$, is used. The MATLAB commands to subtract (5.12) from $\boldsymbol{M}_1$ are as follows:

```
>> format short
>> m1=[4 14 28 -8;1 3 3 2;2 6 10 0];
>> m2=m1-[0;1/4;2/4]*[1,0,0]*m1

m2 =
   4.0000   14.0000   28.0000   -8.0000
        0   -0.5000   -4.0000    4.0000
        0   -1.0000   -4.0000    4.0000
```

Our next step is to locate the largest entry in absolute value using the entries in the second column below row 1. Row 3 is dominant resulting in the interchange $\mathrm{R}_2 \leftrightarrow \mathrm{R}_3$,

$$\begin{bmatrix} 4 & 14 & 28 & -8 \\ 0 & -1 & -4 & 4 \\ 0 & -0.5 & -4 & 4 \end{bmatrix}.$$

One last row operation, $\mathrm{R}_3 \leftarrow \mathrm{R}_3 - \frac{(-0.5)}{(-1)}\mathrm{R}_2$, will give

$$\begin{bmatrix} 4 & 14 & 28 & -8 \\ 0 & -1 & -4 & 4 \\ 0 & 0 & -2 & 2 \end{bmatrix} = [\,\boldsymbol{U} \,|\, \boldsymbol{\beta}\,]. \qquad (5.13)$$

You should be able to produce the results given in (5.13) using MATLAB commands similar to those used to compute **m2**.

At this point the augmented matrix (5.11) has been transformed to the upper triangular form $[\,\boldsymbol{U} \,|\, \boldsymbol{\beta}\,]$ shown in equation (5.13). The back substitution process now takes over to compute the answer. In upper triangular form, the actual equations are

$$\begin{aligned} 4x_1 + 14x_2 + 28x_3 &= -8 \\ -1x_2 - 4x_3 &= 4 \\ -2x_3 &= 2 \end{aligned}$$

It is a simple task to show that the solutions are $x_3 = -1,\ x_2 = 0$ and $x_1 = 5$. Although not evident in this simple textbook example, keep in mind that the purpose of partial pivoting is to improve the accuracy of the Gaussian elimination algorithm.

This brief section has only considered some of the basic aspects of Gaussian elimination. Many other details are needed before we have the complete picture. Further explanation plus information about other pivoting schemes may be found in many textbooks on numerical analysis.

5.3 Matrix Factorization

MATLAB uses factoring methods, related to Gaussian elimination, to solve linear systems. The MATLAB command to solve a linear system, $\boldsymbol{Ax} = \boldsymbol{b}$, is represented by a special symbol, the backslash, \ . Using the example from the last section we find

```
>> a=[2,6,10;1,3,3;4,14,28];b=[0;2;-8];
>> x=a\b
x =
    5
    0
   -1
```

To better understand factoring techniques, consider the problem of factoring a known matrix $\boldsymbol{A}$ into a product of a lower triangular matrix and an upper triangular matrix, $\boldsymbol{A} = \boldsymbol{LU}$, the so-called *LU-factorization* of $\boldsymbol{A}$. To reach this goal we will need to solve a matrix equation. Consider a 3×3 example,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

where there are a total of 12 unknowns in $\boldsymbol{L}$ and $\boldsymbol{U}$. Since $\boldsymbol{A}$ has nine entries, there are only nine equations. To eliminate the difficulty, Doolittle suggested that the diagonal entries of $\boldsymbol{L}$ be set equal to ones reducing the number of unknowns to nine. In other words

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}.$$

In order to compute the l_{ij}'s and u_{ij}'s the symbolic product of $\boldsymbol{L}$ and $\boldsymbol{U}$ is needed. MATLAB produces the following

```
[          U11,          U12,          U13]
[       L21*U11,      L21*U12+U22,      L21*U13+U23]
[       L31*U11,  L31*U12+L32*U22,  L31*U13+L32*U23+U33]
```

The structure of the product suggests a systematic way to compute the unknowns in $\boldsymbol{L}$ and $\boldsymbol{U}$. Keep in mind that the a_{ij}'s are known making the first step obvious, $u_{1j} = a_{1j}$, $j = 1, 2, 3$. The following sequence of steps insures that appropriate values of l_{ij} and u_{ij} are known at the proper time in the row-column procedure beginning with the first row of $\boldsymbol{U}$.

First row of ***U***: $u_{1j} = a_{1j}$ for $j = 1,\ 2,\ 3$

First column of ***L***: $l_{i1} = a_{i1}/u_{11}$ for $i = 2,\ 3$

Second row of ***U***: $u_{2j} = a_{2j} - l_{21}u_{1j}$ for $j = 2,\ 3$

Second column of ***L***: $l_{32} = (a_{32} - l_{31}u_{12})/u_{22}$

Third row of ***U***: $u_{33} = a_{33} - l_{31}u_{13} - l_{32}u_{23}$

This simple LU factorization procedure does not include a pivoting criteria. For example, if a_{11}equals zero the procedure will fail as we attempt to compute the first column of ***L***. The following M-file will compute the Doolittle factorization without pivoting. Note how ***L*** and ***U*** are initialized plus the extensive use of the colon operator. For example, a term such as **U(**k,k:n**)** with $k = 2$, **U**(2,2:n), refers to all entries in row 2 and columns 2, 3, 4,…, n. In other words, a single for loop operates on a given row and multiple columns without the need for a second for loop.

M-file doolit.m

```
function [L,U] = doolit(A)
%DOOLIT Doolittle factorization w/o pivoting
[m,n]=size(A);
L=eye(n);
U=zeros(n,n);
for k = 1:n
        U(k,k:n) = A(k,k:n) - L(k,1:k-1)*U(1:k-1,k:n);
        L(k+1:n,k) = (A(k+1:n,k) - L(k+1:n,1:k-1)*U(1:k-1,k))/U(k,k);
end
```

For example, the 3×3 matrix $\boldsymbol{A} = \begin{bmatrix} -2 & 7 & 1 \\ -4 & 16 & 8 \\ 6 & -11 & 26 \end{bmatrix}$ may be factored as follows: (5.14)

```
>> a=[-2,7,1;-4,16,8;6,-11,26];
>> [L,U]=doolit(a)
L =
    1    0    0
    2    1    0
   -3    5    1
U =
   -2    7    1
    0    2    6
    0    0   -1
```

If ***A*** can be factored without pivoting, the lower triangular matrix ***L*** will contain the multipliers used in Gaussian elimination. A 3×3 example is

$$\boldsymbol{L} = \begin{bmatrix} 1 & 0 & 0 \\ m_{21} & 1 & 0 \\ m_{31} & m_{32} & 1 \end{bmatrix}. \tag{5.15}$$

Furthermore, the upper triangular matrix $\boldsymbol{U}$ will be given by the product

$$\boldsymbol{U} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -m_{32} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -m_{21} & 1 & 0 \\ -m_{31} & 0 & 1 \end{bmatrix} \boldsymbol{A}. \tag{5.16}$$

MATLAB contains a command, **lu**, that will factor a matrix using a pivoting strategy. Pivoting will permute the rows so that $\boldsymbol{LU} = \boldsymbol{PA}$ where $\boldsymbol{P}$ is a *permutation* matrix containing a single one in each row and column with zeros elsewhere. The following example shows that the interchange of rows two and three (one of the elementary row operations) may be accomplished by multiplying by a permutation matrix.

```
a =
   -2    7    1
   -4   16    8
    6  -11   26
>> p=[1,0,0;0,0,1;0,1,0]
p =
    1    0    0
    0    0    1
    0    1    0
>> p*a
ans =
   -2    7    1
    6  -11   26
   -4   16    8
```

Using MATLAB to factor the matrix given in (5.14) yields

```
>> [la,ua,pa]=lu(a)
la =
   1.0000         0         0
  -0.6667    1.0000         0
  -0.3333    0.3846    1.0000
ua =
   6.0000  -11.0000   26.0000
         0    8.6667   25.3333
         0         0   -0.0769
pa =
    0    0    1
    0    1    0
    1    0    0
```

The permutation matrix reveals that the order of the rows has been inverted. This may be confirmed by the following products.

```
>> la*ua
ans =
   6  -11   26
  -4   16    8
  -2    7    1

>> pa*a
ans =
   6  -11   26
  -4   16    8
  -2    7    1
```

Factoring the coefficient matrix of a linear system, $\boldsymbol{Ax} = \boldsymbol{b}$, leads to a different sequence of operations to obtain the solution. With the permutation matrix an outcome of the factoring process, the original system may be pre-multiplied by $\boldsymbol{P}$ to give $\boldsymbol{PAx} = \boldsymbol{Pb}$. With $\boldsymbol{A}$ factored as $\boldsymbol{LU} = \boldsymbol{PA}$, the original system may be rewritten as

$$\boldsymbol{LUx} = \boldsymbol{Pb}. \tag{5.17}$$

The matrix product $\boldsymbol{Ux}$ is actually a column vector. We define $\boldsymbol{y} = \boldsymbol{Ux}$ so that (5.17) becomes

$$\boldsymbol{Ly} = \boldsymbol{Pb}. \tag{5.18}$$

The system in (5.18) contains the lower triangular matrix $\boldsymbol{L}$ and may easily be solved by *forward* substitution. In other words, begin with the first equation to compute y_1 followed by y_2, $y_3, \ldots, y_{n-1}$and finally y_n. With $\boldsymbol{y}$ known we may return to

$$\boldsymbol{Ux} = \boldsymbol{y} \tag{5.19}$$

and compute the values for **x** using backward substitution.

In summary, the factor−forward−backward procedure is

1. Factor $\boldsymbol{A}$ using pivoting gives $\boldsymbol{L}$, $\boldsymbol{U}$ and $\boldsymbol{P}$ so that $\boldsymbol{LU} = \boldsymbol{PA}$
2. Solve $\boldsymbol{Ly} = \boldsymbol{Pb}$ for $\boldsymbol{y}$ using forward substitution
3. Solve $\boldsymbol{Ux} = \boldsymbol{y}$ for $\boldsymbol{x}$ using backward substitution

As an example, consider the system, $\boldsymbol{Ax} = \boldsymbol{b}$, given at the beginning of this section

$$\begin{bmatrix} 2 & 6 & 10 \\ 1 & 3 & 3 \\ 4 & 14 & 28 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} 0 \\ 2 \\ -8 \end{bmatrix}.$$

The first step in the procedure is factoring. An attempt to factor the coefficient matrix $\boldsymbol{A}$ without pivoting using the M-file doolit.m fails.

>> [La,Ua]=doolit(a)
Warning: Divide by zero.

The warning message indicates that pivoting is needed. MATLAB's **lu** may be used to compute the factors and permutation matrix. The results are as follows:

$$\boldsymbol{L} = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.25 & 0.5 & 1 \end{bmatrix}, \qquad \boldsymbol{U} = \begin{bmatrix} 4 & 14 & 28 \\ 0 & -1 & -4 \\ 0 & 0 & -2 \end{bmatrix}, \qquad \boldsymbol{P} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

The second step calls for the solution of $\boldsymbol{Ly} = \boldsymbol{Pb}$ which is given by

$$\begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.25 & 0.5 & 1 \end{bmatrix} \boldsymbol{y} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ -8 \end{bmatrix} = \begin{bmatrix} -8 \\ 0 \\ 2 \end{bmatrix}.$$

It is a simple task to show that the solution for $\boldsymbol{y}$ is $[\,-8 \quad 4 \quad 2\,]^T$. The third and final step makes use of backward substitution to solve $\boldsymbol{Ux = y}$,

$$\begin{bmatrix} 4 & 14 & 28 \\ 0 & -1 & -4 \\ 0 & 0 & -2 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} -8 \\ 4 \\ 2 \end{bmatrix},$$

which gives $\boldsymbol{x} = [\,5 \quad 0 \quad -1\,]^T$.

Although the procedure outlined above may seem unnecessarily complicated, the actual number of arithmetic operations is exactly the same as the number of operations used in Gaussian elimination. For many linear systems the MATLAB backslash operator, \, follows the three step procedure described above. The exact procedure used by MATLAB depends on the form of $\boldsymbol{A}$. For example, if, by chance, $\boldsymbol{A}$ is lower or upper triangular there is no need for factoring and the system may be solved by simple forward or backward substitution although pivoting may be necessary.

The factor−forward−backward procedure is particularly economical if the system $\boldsymbol{Ax} = \boldsymbol{b}$ must be solved repeatedly for different $\boldsymbol{b}$'s. One example is the numerical computation of the inverse of a matrix. Consider the special linear system $\boldsymbol{Av}_1 = \boldsymbol{e}_1 = [\,1\,0\,0\cdots\,0\,0\,]^T$. Symbolically, the solution vector $\boldsymbol{v}_1$ is given by the matrix product $\boldsymbol{v}_1 = \boldsymbol{A^{-1}e}_1$. With the special choice for $\boldsymbol{e}_1$ the vector $\boldsymbol{v}_1$ is identical to the first column of $\boldsymbol{A^{-1}}$. The process may be repeated with the linear system $\boldsymbol{Av}_2 = \boldsymbol{e}_2 = [\,0\,1\,0\cdots\,0\,0\,]^T$ to compute the second column of $\boldsymbol{A^{-1}}$. If $\boldsymbol{A}$ is an $n \times n$ matrix, the process will be repeated n times to compute the n $\boldsymbol{v}_i$'s and thus the inverse matrix. Those readers familiar with the complicated mathematical efforts to compute an inverse will appreciate the direct numerical approach.

MATLAB uses the factoring procedure to compute the determinant of a matrix with the **det** command. The equation $\boldsymbol{LU = PA}$ leading to $\boldsymbol{A} = \boldsymbol{P}^{-1}\boldsymbol{LU}$ is the key. For example, using properties of determinants we have

$$\det(\,\boldsymbol{A}\,) = \det(\,\boldsymbol{P}^{-1}\boldsymbol{LU}) = \det(\boldsymbol{P}^{-1}\,) \cdot \det(\,\boldsymbol{L}\,) \cdot \det(\,\boldsymbol{U}\,).$$

The three determinants on the right hand side are easy to compute. With ones on the main diagonal, we may show that det($\boldsymbol{L}$) $= 1$. Further, det($\boldsymbol{U}$) is simply the product of all entries on the main diagonal of $\boldsymbol{U}$, $u_{11}u_{22}\cdots u_{nn}$. Finally, since $\boldsymbol{P}^{-1} = \boldsymbol{P}^{T}$ is obtained by interchanging rows of the identity matrix, det($\boldsymbol{P}^{-1}$) equals 1 or -1 depending on the number of interchanges.

5.4 Tridiagonal Systems

Tridiagonal matrices occur in various mathematical applications, such as the numerical solution of partial differential equations and in the construction of cubic splines discussed in Chapter 6. The special structure of tridiagonal matrices leads to a simple factoring scheme that results in an efficient solution method for tridiagonal systems. In this section we study how to factor a tridiagonal matrix. A matrix $\boldsymbol{T}$ is said to be tridiagonal if it has the form

$$\boldsymbol{T} = \begin{bmatrix} d_1 & c_1 & 0 & 0 & 0 \\ a_2 & d_2 & c_2 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & a_{n-1} & d_{n-1} & c_{n-1} \\ 0 & 0 & 0 & a_n & dn \end{bmatrix}. \tag{5.20}$$

Non-zero entries may only be on the main diagonal (the d_i's) and the 'diagonals' above and below (the c_i's and the a_i's). Note that the entries have a single subscript corresponding to the row. To save space the matrix $\boldsymbol{T}$ could be stored in three vectors.

Factoring a tridiagonal matrix without pivoting gives an interesting result. Consider the MATLAB example

```
a =
    1    4    0    0    0
   -2    3    6    0    0
    0    1    1    1    0
    0    0    5    3    8
    0    0    0    1    4
>> [L,U]=doolit(a)
L =
    1.0000         0         0         0         0
   -2.0000    1.0000         0         0         0
         0    0.0909    1.0000         0         0
         0         0   11.0000    1.0000         0
         0         0         0   -0.1250    1.0000
U =
    1.0000    4.0000         0         0         0
         0   11.0000    6.0000         0         0
         0         0    0.4545    1.0000         0
         0         0         0   -8.0000    8.0000
         0         0         0         0    5.0000
```

Based on this example, both $\boldsymbol{L}$ and $\boldsymbol{U}$ are also special tridiagonal matrices and, moreover, the non-zero upper diagonal of $\boldsymbol{U}$ is identical to the upper diagonal of $\boldsymbol{A}$.

If we use Gaussian elimination without pivoting on the linear system $\boldsymbol{Tx} = \boldsymbol{b}$ many of the multipliers will be zero. In fact it can be shown that the Doolittle factorization of $\boldsymbol{T}$ will have the form

$$\boldsymbol{T} = \boldsymbol{LU} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ \alpha_2 & 1 & 0 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \alpha_{n-1} & 1 & 0 \\ 0 & 0 & 0 & \alpha_n & 1 \end{bmatrix} \begin{bmatrix} \delta_1 & c_1 & 0 & 0 & 0 \\ 0 & \delta_2 & c_2 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & \delta_{n-1} & c_{n-1} \\ 0 & 0 & 0 & 0 & \delta_n \end{bmatrix}$$

For example, with $n = 5$, the symbolic result for the $\boldsymbol{LU}$ product is

$$\begin{bmatrix} \delta_1 & c_1 & 0 & 0 & 0 \\ \alpha_2\delta_1 & \alpha_2 c_1 + \delta_2 & c_2 & 0 & 0 \\ 0 & \alpha_3\delta_2 & \alpha_3 c_2 + \delta_3 & c_3 & 0 \\ 0 & 0 & \alpha_4\delta_3 & \alpha_4 c_3 + \delta_4 & c_4 \\ 0 & 0 & 0 & \alpha_5\delta_4 & \alpha_5 c_4 + \delta_5 \end{bmatrix}$$

Using the corresponding elements from the tridiagonal matrix $\boldsymbol{T}$, (5.20), we see that

$$\begin{aligned} d_1 &= \delta_1 \\ a_i &= \alpha_i \delta_{i-1} \qquad \text{for } i = 2,\ 3,\ 4, 5 \\ d_i &= \alpha_i c_{i-1} + \delta_i \quad \text{for } i = 2,\ 3,\ 4, 5 \end{aligned} \tag{5.21}$$

Equation (5.21) may be solved for the unknown values, the α_i's and the δ_i's. In general terms

$$\begin{aligned} \delta_1 &= d_1 \\ \alpha_i &= a_i/\delta_{i-1} \qquad \text{for } i = 2,\ \ldots,\ n \\ \delta_i &= d_i - \alpha_i c_{i-1} \quad \text{for } i = 2,\ \ldots,\ n \end{aligned} \tag{5.22}$$

This simple result makes the factoring of a tridiagonal matrix very fast. Only $2n - 2$ divisions and multiplications are needed to compute the unknowns. Consider the following example.

$$\begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} 2 \\ 5 \\ 2 \end{bmatrix}$$

We begin with factoring. With $\delta_1 = d_1 = 2$, equations (5.22) will yield

$$\begin{aligned} \alpha_2 &= a_2/\delta_1 \ = 1/2, \\ \delta_2 &= d_2 - \alpha_2 c_1 = 2 - (1/2)1 = 3/2, \\ \alpha_3 &= a_3/\delta_2 \ = 1/(3/2) = 2/3, \\ \delta_3 &= d_3 - \alpha_3 c_2 = 2 - (2/3)1 = 4/3. \end{aligned}$$

Thus $$\boldsymbol{T} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix} = \boldsymbol{LU} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 0 & 2/3 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 0 \\ 0 & 3/2 & 1 \\ 0 & 0 & 4/3 \end{bmatrix}$$

The second step is forward substitution, $\boldsymbol{Ly} = [\ 2,\ 5,\ 2\ ]^T$.

Solve for $\boldsymbol{y}$: $$\begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 0 & 2/3 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 2 \end{bmatrix}.$$

$$\begin{aligned} 1y_1 &= 2 & y_1 &= 2 \\ (1/2)y_1 + 1y_2 &= 5 & y_2 &= 4 \\ (2/3)y_2 + 1y_3 &= 2 & y_3 &= -2/3 \end{aligned}$$

The third, and final, step is backward substitution, $\boldsymbol{Ux} = \boldsymbol{y} = [\ 2,\ 4,\ 2/3\ ]^T$.

Solve for $\boldsymbol{x}$: $$\begin{bmatrix} 2 & 1 & 0 \\ 0 & 3/2 & 1 \\ 0 & 0 & 4/3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ -2/3 \end{bmatrix}.$$

$$\begin{aligned} (4/3)x_3 &= -2/3 & x_3 &= -1/2 \\ (3/2)x_2 + 1x_3 &= 4 & x_2 &= 3 \\ 2x_1 + 1x_2 &= 2 & x_1 &= -1/2 \end{aligned}$$

Symmetry in the solution, $\boldsymbol{x} = [\ -1/2\ \ 3\ \ -1/2\ ]^T$, is a result of the symmetry in the example problem. MATLAB confirms our efforts with

```
>> a = [2 1 0;1 2 1;0 1 2]; b = [2 5 2]';
>> x=a\b
x =
 -5.0000e-001
  3.0000e+000
 -5.0000e-001
```

5.5 Iterative Methods

Gaussian elimination and the related factoring procedures are often called direct methods for the solution of linear systems. In other words, after a finite number of arithmetic operations we arrive at the solution to the problem. In theory, there is no limit to the size of the problem; however, available computer memory will place a practical restriction on the actual size.

Iterative methods are frequently applied to very large systems where the coefficient matrix, ***A***, contains a high percentage of zero entries, a so-called *sparse* matrix. Beginning with an initial estimate of the solution, usually a column of zeros, an iterative scheme generates a sequence of vectors to approximate the solution. If all goes as intended, the sequence will

converge to the true solution. As a practical issue, the sequence must be terminated at some point. (Recall the termination procedures for root finding methods.)

The basic idea behind an iterative method begins with conversion of the original problem, $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, to an algebraically equivalent form with the unknown $\boldsymbol{x}$ on both sides of the equation, say $\boldsymbol{x} = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{d}$. Based on this equivalent form, we may define an iteration scheme as follows:

$$\widehat{\boldsymbol{x}}_{i+1} = \boldsymbol{C}\,\widehat{\boldsymbol{x}}_i + \boldsymbol{d} \text{ for } i = 0,\ 1,\ 2,\ \ldots \tag{5.23}$$

where the vector $\widehat{\boldsymbol{x}}_i$ is an approximation to the actual solution vector $\boldsymbol{x}$. All iteration schemes require an initial vector, $\widehat{\boldsymbol{x}}_0$, specified by the user. Lacking specific knowledge about the solution, $\widehat{\boldsymbol{x}}_0$ is usually set equal to the null matrix.

To illustrate how an iteration scheme may be derived, consider the three equations

$$\begin{aligned} -7x + 2y + 0z &= 5 \\ 0x + 16y + 8z &= 7 \\ 6x - 11y + 26z &= -3 \end{aligned} \tag{5.24}$$

Solving the first equation for x, the second for y and the third for z gives

$$\begin{aligned} x &= -\frac{1}{7}(5 + 0x - 2y - 0z) \\ y &= \frac{1}{16}(7 + 0x + 0y - 8z) \\ z &= \frac{1}{26}(-3 - 6x + 11y + 0z) \end{aligned}$$

These equations lead to the *Jacobi* iteration method

$$\begin{aligned} \widehat{x}_{i+1} &= -\frac{1}{7}(5 + 0\widehat{x}_i - 2\widehat{y}_i - 0\widehat{z}_i) \\ \widehat{y}_{i+1} &= \frac{1}{16}(7 + 0\widehat{x}_i + 0\widehat{y}_i - 8\widehat{z}_i) \\ \widehat{z}_{i+1} &= \frac{1}{26}(-3 - 6\widehat{x}_i + 11\widehat{y}_i + 0\widehat{z}_i) \end{aligned} \tag{5.25}$$

The example equations were chosen with care. Note that none of the terms on the main diagonal of the coefficient matrix are zero. If a zero appears on the main diagonal the equations should be rearranged to eliminate this difficulty. In general terms (5.25) could be written as

$$\begin{aligned} \widehat{x}_{i+1} &= \frac{1}{a_{11}}(b_1 - 0\widehat{x}_i - a_{12}2\widehat{y}_i - a_{13}\widehat{z}_i) \\ \widehat{y}_{i+1} &= \frac{1}{a_{22}}(b_2 - a_{21}\widehat{x}_i - 0\widehat{y}_i - a_{23}\widehat{z}_i) \\ \widehat{z}_{i+1} &= \frac{1}{a_{33}}(b_3 - a_{31}\widehat{x}_i - a_{32}\widehat{y}_i - 0\widehat{z}_i) \end{aligned} \tag{5.26}$$

Observe that the terms in parentheses on the right hand side of (5.26) may be written as

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} - \begin{bmatrix} 0 & a_{12} & a_{13} \\ a_{21} & 0 & a_{23} \\ a_{31} & a_{32} & 0 \end{bmatrix} \begin{bmatrix} \widehat{x}_i \\ \widehat{y}_i \\ \widehat{z}_i \end{bmatrix}.$$

Using matrices, Jacobi's method begins by writing $\boldsymbol{A}$ as the sum of two terms: first the diagonal elements of $\boldsymbol{A}$ denoted $\boldsymbol{D}_{\mathrm{A}}$, and, second, the off-diagonal elements denoted $\boldsymbol{M} = \boldsymbol{A} - \boldsymbol{D}_{\mathrm{A}}$. In other words $\boldsymbol{Ax} = \boldsymbol{b}$ becomes $(\boldsymbol{D}_{\mathrm{A}} + \boldsymbol{M})\boldsymbol{x} = \boldsymbol{b}$ or $\boldsymbol{D}_{\mathrm{A}}\boldsymbol{x} + \boldsymbol{Mx} = \boldsymbol{b}$. Moving $\boldsymbol{Mx}$ to the right hand side of the equation produces $\boldsymbol{D}_{\mathrm{A}}\boldsymbol{x} = \boldsymbol{b} - \boldsymbol{Mx}$. The right hand side mirrors the matrix expression just above. Finally, multiplying by the inverse of $\boldsymbol{D}_{\mathrm{A}}$ gives $\boldsymbol{x} = \boldsymbol{D}_{\mathrm{A}}^{-1}(\boldsymbol{b} - \boldsymbol{Mx})$ which leads to the Jacobi method

$$\widehat{\boldsymbol{x}}_{i+1} = \boldsymbol{D}_{\mathrm{A}}^{-1}(\boldsymbol{b} - \boldsymbol{M}\,\widehat{\boldsymbol{x}}_i) \text{ for } i = 0,\ 1,\ 2,\ \ldots, \text{ given } \widehat{\boldsymbol{x}}_0. \tag{5.27}$$

Assuming that none of the diagonal elements of $\boldsymbol{A}$ are zero, $a_{ii} \neq 0$, the inverse of $\boldsymbol{D}_{\mathrm{A}}$ is very easy to compute. $\boldsymbol{D}_{\mathrm{A}}^{-1}$ is a diagonal matrix with elements on the main diagonal given by $1/a_{ii}$. You should be able to recognize the relationship between (5.26) and (5.27).

MATLAB contains linear algebra commands that make the computations of (5.27) an easy programming task. The command **diag** will extract the main diagonal of a matrix as a column vector and also construct a diagonal matrix given a vector. For example

```
a =
   -7    2    0
    0   16    8
    6  -11   26
>> v=diag(a)
v =
   -7
   16
   26
>> D=diag(v)
D =
   -7    0    0
    0   16    0
    0    0   26
```

Actually, the two commands may be nested as follows: **>> D=diag(diag(a))**.

The following M-file will perform the computations of the Jacobi iteration scheme and display the results. In addition to the data vectors, a vector of initial values along with the number of iterations must be specified. The M-file jacobi.m implements the matrix computations described in (5.27) and displays a row of solution values at each iteration.

M-file jacobi.m

```
function xj = jacobi(a,b,x0,k)
%JACOBI Jacobi iteration method
% b and x0 must be column vectors
```

```
% k is the number of iterations
b = b(:);xi = x0(:);
D = diag(diag(a));
M = a-D;
Dinv = diag(1./diag(a));
for i = 1:k
  xi = Dinv*(b-M*xi);
disp([i,xi'])
end
xj = xi;
```

Using the example problem of this section, (5.24), and the initial vector $[\,0\,0\,0\,]^T$ we find

```
>> xj =jacobi(a,b,[0 0 0],10);
   1.0000   -0.7143    0.4375   -0.1154
   2.0000   -0.5893    0.4952    0.2345
   3.0000   -0.5728    0.3202    0.2301
   4.0000   -0.6228    0.3224    0.1523
   5.0000   -0.6222    0.3614    0.1648
   6.0000   -0.6110    0.3551    0.1811
   7.0000   -0.6128    0.3470    0.1759
   8.0000   -0.6152    0.3496    0.1728
   9.0000   -0.6144    0.3511    0.1745
  10.0000   -0.6140    0.3503    0.1749
```

It does appear that the values are converging slowly to the solution given by

```
>> x=a\b
x =
  -0.6142
   0.3502
   0.1745
```

The example Jacobi equations, (5.25), suggest another approach, usually called the *Gauss-Seidel* method. As soon as an unknown has been updated, the updated value should be used in subsequent computations. In other words,

$$\begin{aligned}
\widehat{x}_{i+1} &= -\frac{1}{7}(5 + 0\widehat{x}_i - 2\widehat{y}_i - 0\widehat{z}_i) \\
\widehat{y}_{i+1} &= \frac{1}{16}(7 + 0\widehat{x}_{i+1} + 0\widehat{y}_i - 8\widehat{z}_i) \\
\widehat{z}_{i+1} &= \frac{1}{26}(-3 - 6\widehat{x}_{i+1} + 11\widehat{y}_{i+1} + 0\widehat{z}_i)
\end{aligned} \tag{5.28}$$

Intuitively, we expect the Gauss-Seidel method to produce better results.

For use in the Gauss-Seidel method, MATLAB contains two linear algebra commands, **tril** and **triu**, which will extract the lower and upper triangular part of a matrix, respectively. For example,

```
a =
    -7     2     0
     0    16     8
     6   -11    26
>> La=tril(a)
La =
    -7     0     0
     0    16     0
     6   -11    26

>> Ua=triu(a)
Ua =
    -7     2     0
     0    16     8
     0     0    26
```

Note that the diagonal elements appear in both **tril(a)** and **triu(a)**.

The Gauss-Seidel method begins by expressing $\boldsymbol{A}$ as the sum of two matrices, first the lower triangular part of $\boldsymbol{A}$ denoted $\boldsymbol{L}_{\mathrm{A}}$, and, second, the upper triangular part of $\boldsymbol{A}$ with the diagonal elements set to zero denoted $\boldsymbol{M}$. For our example,

```
>> M=Ua-diag(diag(a))
M =
     0     2     0
     0     0     8
     0     0     0
```

Thus, $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ becomes $(\boldsymbol{L}_{\mathrm{A}} + \boldsymbol{M})\boldsymbol{x} = \boldsymbol{b}$ or $\boldsymbol{L}_{\mathrm{A}}\boldsymbol{x} + \boldsymbol{M}\boldsymbol{x} = \boldsymbol{b}$. Moving $\boldsymbol{M}\boldsymbol{x}$ to the right hand side of the equation gives $\boldsymbol{L}_{\mathrm{A}}\boldsymbol{x} = \boldsymbol{b} - \boldsymbol{M}\boldsymbol{x}$. This particular strategy will keep the appropriate updated terms together. See equation (5.28). We may now multiply by the inverse of $\boldsymbol{L}_{\mathrm{A}}$, $\boldsymbol{x} = \boldsymbol{L}_{\mathrm{A}}^{-1}(\boldsymbol{b} - \boldsymbol{M}\boldsymbol{x})$, which leads to the Gauss-Seidel method

$$\widehat{\boldsymbol{x}}_{i+1} = \boldsymbol{L}_{\mathrm{A}}^{-1}(\boldsymbol{b} - \boldsymbol{M}\,\widehat{\boldsymbol{x}}_i) \text{ for } i = 0,\ 1,\ 2,\ \ldots, \text{ given } \widehat{\boldsymbol{x}}_0. \tag{5.29}$$

It is a simple task to modify the M-file jacobi.m to perform the Gauss-Seidel computations. Using the example problem, (5.24), the Gauss-Seidel results are as follows:

```
>> xg = gseidel(a,b,[0 0 0]',10);
   1.0000   -0.7143    0.4375    0.2345
   2.0000   -0.5893    0.3202    0.1561
   3.0000   -0.6228    0.3595    0.1804
   4.0000   -0.6116    0.3473    0.1727
```

5.0000	**-0.6151**	**0.3512**	**0.1751**
6.0000	**-0.6140**	**0.3499**	**0.1743**
7.0000	**-0.6143**	**0.3503**	**0.1746**
8.0000	**-0.6142**	**0.3502**	**0.1745**
9.0000	**-0.6142**	**0.3502**	**0.1745**
10.0000	**-0.6142**	**0.3502**	**0.1745**

Comparing the two sets of output, Jacobi and Gauss-Seidel, some improvement is noted in the second method.

If the coefficient matrix is small there is nothing to gain by using an iterative method; however, if the system is large and also sparse, iteration may be an appropriate choice. There are a number of theorems which provide information on the convergence on iterative methods. Most advanced textbooks on numerical analysis contain the theorems along with proofs.

5.6 Error in Linear Systems

Gaussian elimination, a factoring algorithm or an iteration scheme all lead to an approximate solution influenced by the finite word length of a computer. In many cases the approximate solution is more than satisfactory; however, some information about the error is often desirable.

Let $\boldsymbol{x}$ be the exact solution to $\boldsymbol{Ax} = \boldsymbol{b}$. The approximate solution, computed using the methods of this chapter, is denoted $\boldsymbol{x}_a$. To assess the solution of most mathematical problems it is standard practice to substitute back into the original problem. In other words we hope that the product $\boldsymbol{Ax}_a$ is very close to $\boldsymbol{b}$. To measure the closeness of these two vectors, a *residual* vector may be computed, $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{Ax}_a$.

If $\boldsymbol{A}$ is $n \times n$, the residual vector $\boldsymbol{r}$ will contain n values. In an effort to analyze the residual vector the concepts of a *vector norm* and also a *matrix norm* are needed. A norm is a scalar value, somewhat similar to absolute value, that characterizes either a vector or a matrix. There are various definitions for both norms; however, the following choices are satisfactory for our purposes.

$$\text{Vector norm of } \boldsymbol{r} = \| \boldsymbol{r} \| = \max_{1 \le i \le n} | r_i |. \tag{5.30}$$

$$\text{Matrix norm of } \boldsymbol{A} = \| \boldsymbol{A} \| = \max_{1 \le i \le n} \sum_{j=1}^{n} | a_{ij} |. \tag{5.31}$$

The matrix norm in (5.31) is often called the *row-sum* norm in that $\sum_{j=1}^{n} | a_{ij} |$ sums across row i.

For example, with row sums of 10, 28, and 43, using absolute values, the matrix norm of $\begin{bmatrix} -2 & 7 & 1 \\ -4 & 16 & 8 \\ 6 & -11 & 26 \end{bmatrix}$ is 43. The vector norm of $[\, 1, -3, 2 \,]$ is simply 3. >> **help norm** will provide information on various norms that MATLAB can compute. More advanced texts in numerical analysis and linear algebra will provide further details on vector and matrix norms.

It seems very plausible that if the norm of the residual vector is small we should expect that the norm $\|\boldsymbol{x} - \boldsymbol{x}_a\|$ is also small. For many linear systems this is indeed true; however, depending on the coefficient matrix $\boldsymbol{A}$, there are situations where the prediction fails.

A very important theorem provides an upper bound for the *relative error* defined by $\|\boldsymbol{x} - \boldsymbol{x}_a\| / \|\boldsymbol{x}\|$. Assuming that $\|\boldsymbol{x}\|$ and $\|\boldsymbol{b}\|$are both different from zero we have the following inequality for relative error

$$\frac{\|\boldsymbol{x} - \boldsymbol{x}_a\|}{\|\boldsymbol{x}\|} \leq \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|}. \tag{5.32}$$

The product $\|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|$ in (5.31) is called the *condition number* of the matrix $\boldsymbol{A}$, cond($\boldsymbol{A}$) for short. It is possible to show that cond($\boldsymbol{A}$) ≥ 1. Rewriting (5.32) we have

$$\frac{\|\boldsymbol{x} - \boldsymbol{x}_a\|}{\|\boldsymbol{x}\|} \leq \text{cond}(\boldsymbol{A})\,\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|}. \tag{5.33}$$

This inequality (5.33) has interesting implications. If cond($\boldsymbol{A}$) is small, a small residual norm, $\|\boldsymbol{r}\|$, shows that the relative error will be bounded by a small value. We say that $\boldsymbol{A}$ is *well-conditioned* implying confidence in the numerical results. On the other hand, if cond($\boldsymbol{A}$) is very large, a small residual norm does not imply that the relative error will be small. We also say that $\boldsymbol{A}$ is *ill-conditioned* precluding any quick decisions on the accuracy of our results. In numerical computations with matrices, MATLAB will print a warning message about ill-conditioned matrices and provide an approximate value for the reciprocal of the condition number if such information is warranted. Recall Wilkinson's matrix from Chapter 1.

```
>> inv(w)
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 3.505136e-017.
```

RCOND, the reciprocal of the condition number, is very small indicating that cond($\boldsymbol{W}$) will be very large. In other words $\boldsymbol{W}$ is an ill-conditioned matrix.

5.7 Problems

5-1. Solve the following systems using Gaussian elimination without pivoting. Use exact arithmetic in the computations. Check your answers with MATLAB

a. $\begin{bmatrix} 3 & 2 & 1 \\ -1 & 1 & 0 \\ 5 & 2 & 4 \end{bmatrix}\boldsymbol{x} = \begin{bmatrix} 1 \\ -1 \\ -3 \end{bmatrix}$

b. $\begin{bmatrix} 5 & 6 & 2 \\ 1 & 2 & 1 \\ -3 & 2 & 4 \end{bmatrix}\boldsymbol{x} = \begin{bmatrix} 19 \\ 6 \\ 7 \end{bmatrix}$

5-2. Repeat Problem 5-1 using partial pivoting.

5-3. Solve the following linear system using Gaussian elimination without pivoting and then with partial pivoting. Use exact arithmetic in the computations. Check your answers with MATLAB

$$\begin{bmatrix} 3 & -4 & 5 \\ -6 & 2 & 4 \\ 2 & -5 & -7 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} -15 \\ -10 \\ 6 \end{bmatrix}.$$

5-4. a. Without pivoting, find the Doolittle factorization of $\boldsymbol{A} = \begin{bmatrix} 3 & 6 & 12 \\ -1 & 0 & 2 \\ 3 & 2 & 1 \end{bmatrix}$ by hand.

Check your result with doolit.m.

b. Use the factor−forward−backward scheme with exact arithmetic to solve the system $\boldsymbol{Ax} = [\,5 \quad 4 \quad -5\,]^T$.

5-5. Without pivoting, find the Doolittle factorization of $\begin{bmatrix} 3 & 6 & 12 \\ -1 & 0 & 2 \\ 3 & 2 & 1 \end{bmatrix}$ by hand. Check your result with doolit.m.

5-6. Using the factor−forward−backward scheme with exact arithmetic, solve the tridiagonal system

$$\begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} 2 \\ 4 \\ 2 \end{bmatrix}.$$

5-7. Consider the 4×4 tridiagonal system $\boldsymbol{Tx} = \boldsymbol{b}$. Factor $\boldsymbol{T}$ and then solve the system given below. Use the factor−forward−backward scheme with exact arithmetic.

$$\begin{bmatrix} 3 & 2 & 0 & 0 \\ 1 & -1 & 1 & 0 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 1 & 3 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} 4 \\ 4 \\ -6 \\ -8 \end{bmatrix}$$

5-8. Given the 4×4 tridiagonal matrix $\boldsymbol{T} = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$.

Factor $\boldsymbol{T} = \boldsymbol{LU}$ and then solve the system $\boldsymbol{Tx} = \boldsymbol{LUx} = [\,1 \quad 2 \quad 2 \quad 1\,]^T$. Use exact arithmetic. If $\boldsymbol{T}$ is extended to an $n \times n$ form, generalize the factors.

5-9. Given $\boldsymbol{A} = \begin{bmatrix} 1 & 1 \\ 3 & 4 \end{bmatrix}$. Using partial pivoting, compute $\boldsymbol{P}$, $\boldsymbol{L}$ and $\boldsymbol{U}$ so that $\boldsymbol{PA} = \boldsymbol{LU}$.

5-10. Matrices may be factored in various ways to produce $\boldsymbol{A} = \boldsymbol{LU}$. *Crout's* scheme places ones on the main diagonal of $\boldsymbol{U}$ and allows the main diagonal elements of $\boldsymbol{L}$ to take on other values. In other words,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix}.$$

Determine the Crout factorization of $\begin{bmatrix} 3 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 3 \end{bmatrix}$ using exact arithmetic.

5-11. Find the maximum number of non-zero entries in an $n \times n$ tridiagonal matrix.

5-12. Consider the 2×2 system $\begin{bmatrix} 3 & 2 \\ -1 & 5 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} 7 \\ 3 \end{bmatrix}$. Beginning with $\widehat{\boldsymbol{x}}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, use the Jacobi method to compute $\widehat{\boldsymbol{x}}_2$. Use hand computations and check your results with jacobi.m.

5-13. Using the M-file jacobi.m as a guide, write an M-file to implement the Gauss-Seidel method. Test your M-file by reproducing the data at the end of Section 5.5.

5-14. Repeat Problem 5-12 using the Gauss-Seidel method.

5-15. Using the Jacobi method compute two iterations for the linear system $\boldsymbol{Ax} = \boldsymbol{b}$. Do the computations by hand and check your answers with jacobi.m.

a. $\boldsymbol{A} = \begin{bmatrix} 3 & 2 & 1 \\ -1 & 1 & 0 \\ 5 & 2 & 4 \end{bmatrix}$, $\boldsymbol{b} = \begin{bmatrix} 1 \\ -1 \\ -3 \end{bmatrix}$, $\widehat{\boldsymbol{x}}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

b. $\boldsymbol{A} = \begin{bmatrix} 5 & 6 & 2 \\ 1 & 2 & 1 \\ -3 & 2 & 4 \end{bmatrix}$, $\boldsymbol{b} = \begin{bmatrix} 19 \\ 6 \\ 7 \end{bmatrix}$, $\widehat{\boldsymbol{x}}_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

5-16. Repeat problem 5-15 use the Gauss-Seidel method.

5-17. With $\widehat{\boldsymbol{x}}_0 = [\,0 \quad 0 \quad 0 \quad 0\,]^T$ as a beginning vector, determine the number of iterations required to give four decimal place accuracy using the Jacobi method. Assume that the exact solution is given by MATLAB's backslash command. Use the default **short** format.

$$\begin{bmatrix} 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 1 & 4 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} 1 \\ 3 \\ 3 \\ 1 \end{bmatrix}.$$

5-18. Repeat Problem 5-17 using the Gauss-Seidel method.

5-19. Consider the tridiagonal system

$$\begin{bmatrix} 3 & 7 & 0 & 0 \\ 16 & 5 & 3 & 0 \\ 0 & 1 & 6.5 & 3 \\ 0 & 0 & 2.5 & 4 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} -1.25 \\ 13.25 \\ 14.375 \\ 12.125 \end{bmatrix}.$$

Attempt to solve the system with either the Jacobi or Gauss-Seidel method. Compare your results with those obtained using the backslash command.

5-20. Consider the tridiagonal system

$$\begin{bmatrix} 7 & 3 & 0 & 0 \\ 5 & 16 & 3 & 0 \\ 0 & 1 & 6.5 & 3 \\ 0 & 0 & 2.5 & 4 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} 3.75 \\ -5 \\ 14.375 \\ 12.125 \end{bmatrix}.$$

Although not dramatically different than the previous problem, attempt to solve the system with either the Jacobi or Gauss-Seidel method. The coefficient matrix in this problem is said to be *diagonally dominant* in which case the iterative methods will converge. Compare your results with those obtained using the backslash command. After 10 iterations compute the accuracy of the computations.

5-21. Consider $\boldsymbol{Ax} = \boldsymbol{b}$. If the matrix $\boldsymbol{A}$ is ill-conditioned, small changes in $\boldsymbol{b}$ will frequently produce large changes in the solution vector $\boldsymbol{x}$. *Hilbert's* matrix is a classic example. See >> **help hilb**. Set $\boldsymbol{A} =$ **hilb**(10). Use MATLAB to solve the linear system when $\boldsymbol{b} = [\,9\,8\,7\,6\,5\,4\,3\,2\,1\,0\,]^T$. Replace the fifth entry of $\boldsymbol{b}$ with 4.995, a 0.1% decrease, and solve the new system. Compute the percent change in x_1. Percent change is the relative change times 100%.