4.7 ITERATIVE METHODS

A system of linear equations can also be solved by using an iterative approach. The process, in principle, is the same as in the fixed-point iteration method used for solving a single nonlinear equation (see Section 3.7). In an iterative process for solving a system of equations, the equations are written in an explicit form in which each unknown is written in terms of the other unknown. The explicit form for a system of four equations is illustrated in Fig. 4-21.

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 &= b_1\\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 &= b_2\\
a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 &= b_3\\
a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 &= b_4
\end{aligned}
$$

(a)

Writing the equations in an explicit form. $\longrightarrow$

$$
\begin{aligned}
x_1 &= [b_1 - (a_{12}x_2 + a_{13}x_3 + a_{14}x_4)]/a_{11}\\
x_2 &= [b_2 - (a_{21}x_1 + a_{23}x_3 + a_{24}x_4)]/a_{22}\\
x_3 &= [b_3 - (a_{31}x_1 + a_{32}x_2 + a_{34}x_4)]/a_{33}\\
x_4 &= [b_4 - (a_{21}x_1 + a_{42}x_2 + a_{43}x_3)]/a_{44}
\end{aligned}
$$

(b)

Figure 4-21: Standard (*a*) and explicit (*b*) forms of a system of four equations.

The solution process starts by assuming initial values for the unknowns (first estimated solution). In the first iteration, the first assumed solution is substituted on the right-hand side of the equations, and the new values that are calculated for the unknowns are the second estimated solution. In the second iteration, the second solution is substituted back in the equations to give new values for the unknowns, which are the third estimated solution. The iterations continue in the same manner, and when the method does work, the solutions that are obtained as successive iterations converge toward the actual solution. For a system with n equations, the explicit equations for the $[x_i]$ unknowns are:

$$x_i = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1,\, j\neq i}^{j=n} a_{ij}x_j\right] \qquad i = 1, 2, \ldots, n \tag{4.51}$$

Condition for convergence

For a system of n equations $[a][x] = [b]$, a sufficient condition for convergence is that in each row of the matrix of coefficients $[a]$ the absolute value of the diagonal element is greater than the sum of the absolute values of the off-diagonal elements.

$$|a_{ii}| > \sum_{j=1,\, j\neq i}^{j=n} |a_{ij}| \tag{4.52}$$

This condition is sufficient but not necessary for convergence when the iteration method is used. When condition (4.52) is satisfied, the matrix $[a]$ is classified as ***diagonally dominant***, and the iteration process converges toward the solution. The solution, however, might converge even when Eq. (4.52) is not satisfied.

Two specific iterative methods for executing the iterations, the Jacobi and Gauss–Seidel methods, are presented next. The difference between the two methods is in the way that the new calculated values of the unknowns are used. In the Jacobi method, the estimated values of the unknowns that are used on the right-hand side of Eq. (4.51) are updated all at once at the end of each iteration. In the Gauss–Seidel method, the value of each unknown is updated (and used in the calculation of the new estimate of the rest of the unknowns in the same iteration) when a new estimate for this unknown is calculated.

4.7.1 Jacobi Iterative Method

In the Jacobi method, an initial (first) value is assumed for each of the unknowns, $x_1^{(1)}, x_2^{(1)}, \ldots, x_n^{(1)}$. If no information is available regarding the approximate values of the unknown, the initial value of all the unknowns can be assumed to be zero. The second estimate of the solution $x_1^{(2)}, x_2^{(2)}, \ldots, x_n^{(2)}$ is calculated by substituting the first estimate in the right-hand side of Eqs. (4.51):

$$x_i^{(2)} = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1,\, j\neq i}^{j=n} a_{ij}x_j^{(1)}\right] \qquad i = 1, 2, \ldots, n \tag{4.53}$$

In general, the $(k+1)$th estimate of the solution is calculated from the (k)th estimate by:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1,\, j\neq i}^{j=n} a_{ij}x_j^{(k)}\right] \qquad i = 1, 2, \ldots, n \tag{4.54}$$

The iterations continue until the differences between the values that are obtained in successive iterations are small. The iterations can be stopped when the absolute value of the estimated relative error (see Section 3.2) of all the unknowns is smaller than some predetermined value:

$$\left|\frac{x_i^{(k+1)} - x_i^{(k)}}{x_i^{(k)}}\right| < \varepsilon \qquad i = 1, 2, \ldots, n \tag{4.55}$$

4.7.2 Gauss-Seidel Iterative Method

In the Gauss–Seidel method, initial (first) values are assumed for the unknowns $x_2, x_3, \ldots, x_n$ (all of the unknowns except x_1). If no information is available regarding the approximate value of the unknowns, the initial value of all the unknowns can be assumed to be zero. The first assumed values of the unknowns are substituted in Eq. (4.51) with $i = 1$ to calculate the value of x_1. Next, Eq. (4.51) with $i = 2$ is used for calculating a new value for x_2. This is followed by using Eq. (4.51) with $i = 3$ for calculating a new value for x_3. The process continues until $i = n$, which is the end of the first iteration. Then, the second iteration starts with $i = 1$ where a new value for x_1 is calculated, and so on. In the Gauss–Seidel method, the current values of the unknowns are used for calculating the new value of the next unknown. In other words, as a new value of an unknown is calculated, it is immediately used for the next application of Eq. (4.51). (In the Jacobi method, the values of the unknowns obtained in one iteration are used as a complete set for calculating the new values of the unknowns in the next iteration. The values of the unknowns are not updated in the middle of the iteration.)

Applying Eq. (4.51) to the Gauss–Seidel method gives the iteration formula:

$$x_1^{(k+1)} = \frac{1}{a_{11}}\left[b_1 - \sum_{j=2}^{j=n} a_{1j}x_j^{(k)}\right]$$

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left[b_i - \left(\sum_{j=1}^{j=i-1} a_{ij}x_j^{(k+1)} + \sum_{j=i+1}^{j=n} a_{ij}x_j^{(k)}\right)\right] \quad i = 2, 3, \ldots, n-1 \tag{4.56}$$

$$x_n^{(k+1)} = \frac{1}{a_{nn}}\left[b_n - \sum_{j=1}^{j=n-1} a_{nj}x_j^{(k+1)}\right]$$

Notice that the values of the unknowns in the $k+1$ iteration, $x_i^{(k+1)}$, are calculated by using the values $x_j^{(k+1)}$ obtained in the $k+1$ iteration for $j < i$ and using the values $x_j^{(k)}$ for $j > i$. The criterion for stopping the iterations is the same as in the Jacobi method, Eq. (4.55). The Gauss–Seidel method converges faster than the Jacobi method and requires less computer memory when programmed. The method is illustrated for a system of four equations in Example 4-8.

Example 4-8: Solving a set of four linear equations using Gauss–Seidel method.

Solve the following set of four linear equations using the Gauss–Seidel iteration method.

$$\begin{aligned}
9x_1 - 2x_2 + 3x_3 + 2x_4 &= 54.5 \\
2x_1 + 8x_2 - 2x_3 + 3x_4 &= -14 \\
-3x_1 + 2x_2 + 11x_3 - 4x_4 &= 12.5 \\
-2x_1 + 3x_2 + 2x_3 + 10x_4 &= -21
\end{aligned}$$

SOLUTION

First, the equations are written in an explicit form (see Fig. 4.21):

$$\begin{aligned}
x_1 &= [54.5 - (-2x_2 + 3x_3 + 2x_4)]/9 \\
x_2 &= [-14 - (2x_1 - 2x_3 + 3x_4)]/8 \\
x_3 &= [12.5 - (-3x_1 + 2x_2 - 4x_4)]/11 \\
x_4 &= [-21 - (-2x_1 + 3x_2 + 2x_3)]/10
\end{aligned} \tag{4.57}$$

As a starting point, the initial value of all the unknowns, $x_1^{(1)}, x_2^{(1)}, x_3^{(1)}$, and $x_4^{(1)}$, is assumed to be zero. The first two iterations are calculated manually, and then a MATLAB program is used for calculating the values of the unknowns in seven iterations.

Manual calculation of the first two iterations:

The second estimate of the solution ($k = 2$) is calculated in the first iteration by using Eqs. (4.57). The values that are substituted for x_i in the right-hand side of the equations are the most recent known values. This means that when the first equation is used to calculate $x_1^{(2)}$, all the x_i values are zero. Then, when the second equation is used to calculate $x_2^{(2)}$, the new value $x_1^{(2)}$ is substituted for x_1, but the older values $x_3^{(1)}$ and $x_4^{(1)}$ are substituted for x_3 and x_4, and so on:

$$x_1^{(2)} = [54.5-(-2\cdot 0+3\cdot 0+2\cdot 0)]/9 = 6.056$$

$$x_2^{(2)} = [-14-(2\cdot 6.056-(2\cdot 0)+3\cdot 0)]/8 = -3.264$$

$$x_3^{(2)} = [12.5-(-3\cdot 6.056+2\cdot -3.264-(4\cdot 0))]/11 = 3.381$$

$$x_4^{(2)} = [-21-(-2\cdot 6.056+3\cdot -3.264+2\cdot 3.381)]/10 = -0.5860$$

The third estimate of the solution ($k = 3$) is calculated in the second iteration:

$$x_1^{(3)} = [54.5-(-2\cdot -3.264+3\cdot 3.381+2\cdot -0.5860)]/9 = 4.333$$

$$x_2^{(3)} = [-14-(2\cdot 4.333-(2\cdot 3.381)+3\cdot -0.5860)]/8 = -1.768$$

$$x_3^{(3)} = [12.5-(-3\cdot 4.333+2\cdot -1.768-(4\cdot -0.5860))]/11 = 2.427$$

$$x_4^{(3)} = [-21-(-2\cdot 4.333+3\cdot -1.768+2\cdot 2.427)]/10 = -1.188$$

MATLAB program that calculates the first seven iterations:

The following is a MATLAB program in a script file that calculates the first seven iterations of the solution by using Eqs. (4.57):

Program 4-8: Script file. Gauss–Seidel iteration.

```
k = 1; x1 = 0; x2 = 0; x3 = 0; x4 = 0;
disp('  k          x1         x2         x3          x4')
fprintf(' %2.0f      %-8.5f  %-8.5f  %-8.5f  %-8.5f \n', k, x1, x2, x3, x4)
for k = 2 : 8
   x1=(54.5-(-2*x2+3*x3+2*x4))/9;
   x2=(-14-(2*x1-2*x3+3*x4))/8;
   x3=(12.5-(-3*x1+2*x2-4*x4))/11;
   x4=(-21-(-2*x1+3*x2+2*x3))/10;
   fprintf(' %2.0f     %-8.5f  %-8.5f  %-8.5f  %-8.5f \n', k, x1, x2, x3, x4)
end
```

When the program is executed, the following results are displayed in the Command Window.

```
  k         x1         x2        x3         x4
  1      0.00000   0.00000  0.00000   0.00000
  2      6.05556  -3.26389  3.38131  -0.58598
  3      4.33336  -1.76827  2.42661  -1.18817
```

```
   4       5.11778  -1.97723  2.45956  -0.97519
   5       5.01303  -2.02267  2.51670  -0.99393
   6       4.98805  -1.99511  2.49806  -1.00347
   7       5.00250  -1.99981  2.49939  -0.99943
   8       5.00012  -2.00040  2.50031  -0.99992
```

The results show that the solution converges toward the exact solution, which is $x_1 = 5$, $x_2 = -2$, $x_3 = 2.5$, and $x_4 = -1$.

4.8 USE OF MATLAB BUILT-IN FUNCTIONS FOR SOLVING A SYSTEM OF LINEAR EQUATIONS

MATLAB has mathematical operations and built-in functions that can be used for solving a system of linear equations and for carrying out other matrix operations that are described in this chapter.

4.8.1 Solving a System of Equations Using MATLAB's Left and Right Division

***Left division* \ :** Left division can be used to solve a system of n equations written in matrix form $[a][x] = [b]$, where $[a]$ is the $(n \times n)$ matrix of coefficients, $[x]$ is an $(n \times 1)$ column vector of the unknowns, and $[b]$ is an $(n \times 1)$ column vector of constants.

```
x = a\b
```

For example, the solution of the system of equations in Examples 4-1 and 4-2 is calculated by (Command Window):

```
>> a=[4 -2 -3 6; -6 7 6.5 -6; 1 7.5 6.25 5.5; -12 22 15.5 -1];
>> b=[12; -6.5; 16; 17];
>> x=a\b
x =
   2.0000
   4.0000
  -3.0000
   0.5000
```

***Right division* / :** Right division is used to solve a system of n equations written in matrix form $[x][a] = [b]$, where $[a]$ is the $(n \times n)$ matrix of coefficients, $[x]$ is a $(1 \times n)$ row vector of the unknowns, and $[b]$ is a $(1 \times n)$ row vector of constants.

```
x = b/a
```

For example, the solution of the system of equations in Examples 4-1 and 4-2 is calculated by (Command Window):

```
>> a=[4 -6 1 -12; -2 7 7.5 22; -3 6.5 6.25 15.5; 6 -6 5.5 -1];
>> b=[12 -6.5 16 17];
>> x=b/a
x =
   2.0000    4.0000   -3.0000    0.5000
```

Notice that the matrix $[a]$ used in the right division calculation is the transpose of the matrix used in the left division calculation.

4.8.2 Solving a System of Equations Using MATLAB's Inverse Operation

In matrix form, the system of equations $[a][x]=[b]$ can be solved for $[x]$. Multiplying both sides from the left by $[a]^{-1}$ (the inverse of $[a]$) gives:

$$[a]^{-1}[a][x] = [a]^{-1}[b] \tag{4.58}$$

Since $[a]^{-1}[a] = [I]$ (identity matrix), and $[I][x] = [x]$, Eq. (4.58) reduces to:

$$[x] = [a]^{-1}[b] \tag{4.59}$$

In MATLAB, the inverse of a matrix $[a]$ can be calculated either by raising the matrix to the power of –1 or by using the `inv(a)` function. Once the inverse is calculated, the solution is obtained by multiplying the vector $[b]$ by the inverse. This is demonstrated for the solution of the system in Examples 4-1 and 4-2.

```
>> a=[4 -2 -3 6; -6 7 6.5 -6; 1 7.5 6.25 5.5; -12 22 15.5 -1];
>> b=[12; -6.5; 16; 17];
>> x=a^-1*b
x =
   2.0000
   4.0000
  -3.0000
   0.5000
```

The same result is obtained by typing >> x = inv(a)*b.

4.8.3 MATLAB's Built-In Function for LU Decomposition

MATLAB has a built-in function, called `lu`, that decomposes a matrix $[a]$ into the product $[L][U]$, such that $[a] = [L][U]$ where $[L]$ is a lower triangular matrix and $[U]$ is an upper triangular matrix. One form of the function is:

```
[L, U, P] = lu(a)
```

`L` is a lower triangular matrix.
`U` is an upper triangular matrix.
`P` is a permutation matrix.

`a` is the matrix to be decomposed.

MATLAB uses partial pivoting when determining the factorization. Consequently, the matrices $[L]$ and $[U]$ that are determined by MATLAB are the factorization of a matrix with rows that may be in a different order than in $[a]$. The permutation matrix $[P]$ (a matrix with 1s and 0s) contains the information about the pivoting. Multiplying $[a]$ by the matrix $[P]$ gives the matrix whose decomposition is given by $[L]$ and $[U]$ (see Section 4.5.3):

$$[L][U] = [P][a] \tag{4.60}$$

The matrix $[P][a]$ has the same rows as $[a]$ but in a different order. If MATLAB does not use partial pivoting when the function `lu` is used, then the permutation matrix $[P]$ is the identity matrix.

If the matrices $[L]$ and $[U]$ that are determined by the `lu` function are subsequently used for solving a system of equations $[a][x] = [b]$ (by using Eqs. (4.23) and (4.22)), then the vector $[b]$ has to be multiplied by the permutation matrix $[P]$. This pivots the rows in $[b]$ to be consistent with the pivoting in $[a]$. The following shows a MATLAB solution of the system of equations from Examples 4-1 and 4-2 using the function `lu`.

Decomposition of $[a]$ using MATLAB's `lu` function.

```
>> a=[4 -2 -3 6; -6 7 6.5 -6; 1 7.5 6.25 5.5; -12 22 15.5 -1];
>> b=[12; -6.5; 16; 17];
>> [L, U, P]=lu(a)
L =
    1.0000         0         0         0
   -0.0833    1.0000         0         0
   -0.3333    0.5714    1.0000         0
    0.5000   -0.4286   -0.9250    1.0000
U =
  -12.0000   22.0000   15.5000   -1.0000
         0    9.3333    7.5417    5.4167
         0         0   -2.1429    2.5714
         0         0         0   -0.800
```

```
P =
  0    0    0    1
  0    0    1    0
  1    0    0    0
  0    1    0    0
>> y=L\(P*b)
```

Multiplying $[P][a]$ gives the pivoted $[a]$

```
>> P*a
ans =
 -12.0000  22.0000  15.5000  -1.0000
   1.0000   7.5000   6.2500   5.5000
   4.0000  -2.0000  -3.0000   6.0000
  -6.0000   7.0000   6.5000  -6.0000
```

Solve for y in Eq. (4.23).

Vector $[b]$ is multiplied by the permutation matrix.

```
y =
  17.0000
  17.4167
   7.7143
  -0.4000
>> x=U\y
x =
   2.0000
   4.0000
  -3.0000
   0.5000
```

Solve for x in Eq. (4.22).

4.8.4 Additional MATLAB Built-In Functions

MATLAB has many built-in functions that can be useful in the analysis of systems of equations. Several of these functions are presented in Table 4-1. Note that the operations that are related to some of the functions in the table are discussed in Section 4.9.

Table 4-1: Built-in MATLAB functions for matrix operations and analysis.

Function	Description	Example

`inv(A)`	Inverse of a matrix.
	A is a square matrix. Returns the inverse of A.
		`>> A=[-3 1 0.6; 0.2 -4 3; 0.1 0.5 2];`
		`>> Ain=inv(A)`
		`Ain =`
		`  -0.3310   -0.0592    0.1882`
		`  -0.0035   -0.2111    0.3178`
		`   0.0174    0.0557    0.4111`

Table 4-1: Built-in MATLAB functions for matrix operations and analysis. (Continued)

Function	Description	Example

`d=det(A)`	Determinant of a matrix
	`A` is a square matrix, `d` is the determinant of `A`.
		`>> A=[-3 1 0.6; 0.2 -4 3; 0.1 0.5 2];`
		`>> d=det(A)`
		`d =`
		`   28.7000`

`n=norm(A)`
 	
`n=norm(A,p)`
	Vector and matrix norm
	`A` is a vector or a matrix, `n` is its norm.
	***When A is a vector*:**
	`norm(A,p)` returns:
	`sum(abs(A.^p)^(1/p).`
	`p=inf` The infinity norm (see Eq. (4.70)).
	`norm(A)` Returns the Euclidean 2-norm (see Eq. (4.72)), same as `norm(A,2).`
		`>> A=[2 0 7 -9];`
		`>> n=norm(A,1)`
		`n =`
		`   18`
		`>> n=norm(A,inf)`
		`n =`
		`     9`
		`>> n=norm(A,2)`
		`n =`
		`   11.5758`

	***When A is a matrix*:**
	`norm(A,p)` returns:
	`p=1` The 1-norm (largest column sum of `A` (see Eq. (4.74)).
	`p=2` The largest singular value, same as `norm(A)` (see Eq. (4.75)). This is not the Euclidean norm (see Eq. (4.76)).
	`p=inf` The infinity norm (see Eq. (4.73)).
		`>> A=[1 3 -2; 0 -1 4; 5 2 3];`
		`>> n=norm(A,1)`
		`n =`
		`     9`
		`>> n=norm(A,2)`
		`n =`
		`   6.4818`
		`>> n=norm(A,inf)`
		`n =`
		`   10`

`c=cond(A)`
 	
`c=cond(A,p)`
	Condition number (see Eq. (4.86))
	`A` is a square matrix, `c` is the condition number of `A`.
	`cond(A)` The same as `p=2`.
	`p=1` The 1-norm condition number.
	`p=2` The 2-norm condition number.
	`p=inf` The infinity norm condition number.
		`>> a=[9 -2 3 2; 2 8 -2 3; -3 2 11 -4; -2 3 2 10];`
		`>> cond(a,inf)`
		`ans =`
		`   3.8039`
		See the end of Example 4-10.

4.9 TRIDIAGONAL SYSTEMS OF EQUATIONS

Tridiagonal systems of linear equations have a matrix of coefficients with zero as their entries except along the diagonal, above-diagonal, and below-diagonal elements. A tridiagonal system of n equations in matrix form is shown in Eq. (4.61) and is illustrated for a system of five equations in Fig. 4-22.

$$\begin{bmatrix} A_{11} & A_{12} & 0 & 0 & 0 \\ A_{21} & A_{22} & A_{23} & 0 & 0 \\ 0 & A_{32} & A_{33} & A_{34} & 0 \\ 0 & 0 & A_{43} & A_{44} & A_{45} \\ 0 & 0 & 0 & A_{54} & A_{55} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{bmatrix}$$

Figure 4-22: Tridiagonal system of five equations.

$$\begin{bmatrix} A_{11} & A_{12} & 0 & 0 & 0 & 0 & 0 & 0 \\ A_{21} & A_{22} & A_{23} & 0 & 0 & 0 & 0 & 0 \\ 0 & A_{32} & A_{33} & A_{34} & 0 & 0 & 0 & 0 \\ & & \cdots & \cdots & \cdots & & & \\ & & & \cdots & \cdots & \cdots & & \\ 0 & 0 & 0 & 0 & A_{n-2,\,n-3} & A_{n-2,\,n-2} & A_{n-2,\,n-1} & 0 \\ 0 & 0 & 0 & 0 & 0 & A_{n-1,\,n-2} & A_{n-1,\,n-1} & A_{n-1,\,n} \\ 0 & 0 & 0 & 0 & 0 & 0 & A_{n,\,n-1} & A_{n,\,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ \cdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ \cdots \\ \cdots \\ B_{n-2} \\ B_{n-1} \\ B_n \end{bmatrix} \tag{4.61}$$

The matrix of coefficients of tridiagonal systems has many elements that are zero (especially when the system contains a large number of equations). The system can be solved with the standard methods (Gauss, Gauss–Jordan, LU decomposition), but then a large number of zero elements are stored and a large number of needless operations (with zeros) are executed. To save computer memory and computing time, special numerical methods have been developed for solving tridiagonal systems of equations. One of these methods, the Thomas algorithm, is described in this section.

Many applications in engineering and science require the solution of tridiagonal systems of equations. Some numerical methods for solving differential equations also involve the solution of such systems.

Thomas algorithm for solving tridiagonal systems

The Thomas algorithm is a procedure for solving tridiagonal systems of equations. The method of solution in the Thomas algorithm is similar to the Gaussian elimination method in which the system is first changed to upper triangular form and then solved using back substitution. The Thomas algorithm, however, is much more efficient because only the nonzero elements of the matrix of coefficients are stored, and only the necessary operations are executed. (Unnecessary operations on the zero elements are eliminated.)

The Thomas algorithm starts by assigning the nonzero elements of the tridiagonal matrix of coefficients $[A]$ to three vectors. The diagonal elements A_{ii} are assigned to vector d (d stands for diagonal) such that $d_i = A_{ii}$. The above diagonal elements $A_{i,\,i+1}$ are assigned to vector a (a stands for above diagonal) such that $a_i = A_{i,\,i+1}$, and the below diagonal elements $A_{i-1,\,i}$ are assigned to vector b (b stands for below diagonal), such that $b_i = A_{i-1,\,i}$. With the nonzero elements in the matrix of coefficients stored as vectors, the system of equations has the form:

$$\begin{bmatrix} d_1 & a_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ b_2 & d_2 & a_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & b_3 & d_3 & a_3 & 0 & 0 & 0 & 0 \\ & & \cdots & \cdots & \cdots & & & \\ & & & \cdots & \cdots & \cdots & & \\ 0 & 0 & 0 & 0 & b_{n-2} & d_{n-2} & a_{n-2} & 0 \\ 0 & 0 & 0 & 0 & 0 & b_{n-1} & d_{n-1} & a_{n-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & b_n & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ \cdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ \cdots \\ \cdots \\ B_{n-2} \\ B_{n-1} \\ B_n \end{bmatrix} \tag{4.62}$$

It should be emphasized here that in Eq. (4.62) the matrix of coefficients is displayed as a matrix (with the 0s), but in the Thomas algorithm only the vectors b, d, and a are stored.

Next, the first row is normalized by dividing the row by d_1. This makes the element d_1 (to be used as the pivot element) equal to 1:

$$\begin{bmatrix} 1 & a'_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ b_2 & d_2 & a_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & b_3 & d_3 & a_3 & 0 & 0 & 0 & 0 \\ & & \cdots & \cdots & \cdots & & & \\ & & & \cdots & \cdots & \cdots & & \\ 0 & 0 & 0 & 0 & b_{n-2} & d_{n-2} & a_{n-2} & 0 \\ 0 & 0 & 0 & 0 & 0 & b_{n-1} & d_{n-1} & a_{n-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & b_n & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ \cdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} B'_1 \\ B_2 \\ B_3 \\ \cdots \\ \cdots \\ B_{n-2} \\ B_{n-1} \\ B_n \end{bmatrix} \tag{4.63}$$

where $a'_1 = a_1/d_1$ and $B'_1 = B_1/d_1$.

Now the element b_2 is eliminated. The first row (the pivot row) is multiplied by b_2 and then is subtracted from the second row:

$$\begin{bmatrix} 1 & a'_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & d'_2 & a_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & b_3 & d_3 & a_3 & 0 & 0 & 0 & 0 \\ & & \cdots & \cdots & \cdots & & & \\ & & & \cdots & \cdots & \cdots & & \\ 0 & 0 & 0 & 0 & b_{n-2} & d_{n-2} & a_{n-2} & 0 \\ 0 & 0 & 0 & 0 & 0 & b_{n-1} & d_{n-1} & a_{n-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & b_n & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ \cdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} B'_1 \\ B'_2 \\ B_3 \\ \cdots \\ \cdots \\ B_{n-2} \\ B_{n-1} \\ B_n \end{bmatrix} \tag{4.64}$$

where $d'_2 = d_2 - b_2 a'_1$, and $B'_2 = B_2 - B_1 b_2$.

The operations performed with the first and second row are repeated with the second and third rows. The second row is normalized by dividing the row by d'_2. This makes the element d'_2 (to be used as the pivot element) equal to 1. The second row is then used to eliminate b_3 in the third row.

This process continues row after row until the system of equations is transformed to be upper triangular with 1s along the diagonal:

$$\begin{bmatrix} 1 & a'_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & a'_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & a'_3 & 0 & 0 & 0 & 0 \\ & & \cdots & \cdots & \cdots & & & \\ & & & \cdots & \cdots & \cdots & & \\ 0 & 0 & 0 & 0 & 0 & 1 & a'_{n-2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & a'_{n-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ \cdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} B'_1 \\ B''_2 \\ B''_3 \\ \cdots \\ \cdots \\ B''_{n-2} \\ B''_{n-1} \\ B''_n \end{bmatrix} \tag{4.65}$$

Once the matrix of coefficients is in upper triangular form, the values of the unknowns are calculated by using back substitution.

In mathematical form, the Thomas algorithm can be summarized in the following steps:

***Step 1*:** Define the vectors $b = [0, b_2, b_3, \ldots, b_n]$, $d = [d_1, d_2, \ldots, d_n]$, $a = [a_1, a_2, \ldots, a_{n-1}]$, and $B = [B_1, B_2, \ldots, B_n]$.

***Step 2*:** Calculate: $a_1 = \dfrac{a_1}{d_1}$ and $B_1 = \dfrac{B_1}{d_1}$.

***Step 3*:** For $i = 2, 3, \ldots, n-1$, calculate:

$$a_i = \frac{a_i}{d_i - b_i a_{i-1}} \text{ and } B_i = \frac{B_i - b_i B_{i-1}}{d_i - b_i a_{i-1}}$$

***Step 4*:** Calculate: $B_n = \dfrac{B_n - b_n B_{n-1}}{d_n - b_n a_{n-1}}$

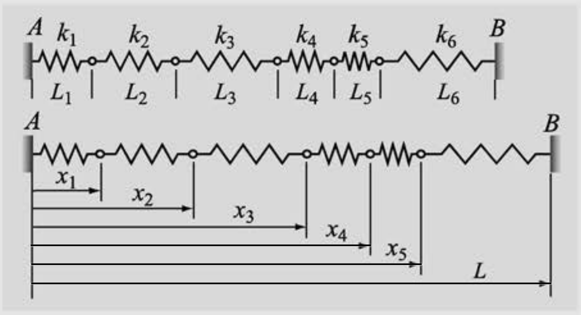
***Step 5*:** Calculate the solution using back substitution:

$x_n = B_n$ and for $i = n-1, n-2, n-3, \ldots, 2, 1$, $x_i = B_i - a_i x_{i+1}$

A solution of a tridiagonal system of equations, using a user-defined MATLAB function, is shown in Example 4-9.

Example 4-9: Solving a tridiagonal system of equations using the Thomas algorithm.

Six springs with different spring constants k_i and unstretched lengths L_i are attached to each other in series. The endpoint B is then displaced such that the distance between points A and B is $L = 1.5$ m. Determine the positions $x_1, x_2, \ldots, x_5$ of the endpoints of the springs.



The spring constants and the unstretched lengths of the springs are:

spring	1	2	3	4	5	6
k (kN/m)	8	9	15	12	10	18
L (m)	0.18	0.22	0.26	0.19	0.15	0.30

SOLUTION

The force, F, in a spring is given by:

$$F = k\delta$$

where k is the spring constant and δ is the extension of the spring beyond its unstretched length. Since the springs are connected in series, the force in all of the springs is the same. Consequently, it is possible to write five equations that equate the force in every two adjacent springs. For example, the condition that the force in the first spring is equal to the force in the second spring gives:

$$k_1(x_1 - L_1) = k_2[(x_2 - x_1) - L_2]$$

Similarly, four additional equations can be written:

$$k_2[(x_2 - x_1) - L_2] = k_3[(x_3 - x_2) - L_3]$$

$$k_3[(x_3 - x_2) - L_3] = k_4[(x_4 - x_3) - L_4]$$

$$k_4[(x_4 - x_3) - L_4] = k_5[(x_5 - x_4) - L_5]$$

$$k_5[(x_5 - x_4) - L_5] = k_6[(L - x_5) - L_6]$$

The five equations form a system that is tridiagonal. In matrix form the system is:

$$\begin{bmatrix} k_1 + k_2 & -k_2 & 0 & 0 & 0 \\ -k_2 & k_2 + k_3 & -k_3 & 0 & 0 \\ 0 & -k_3 & k_3 + k_4 & -k_4 & 0 \\ 0 & 0 & -k_4 & k_4 + k_5 & -k_5 \\ 0 & 0 & 0 & -k_5 & k_5 + k_6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} k_1L_1 - k_2L_2 \\ k_2L_2 - k_3L_3 \\ k_3L_3 - k_4L_4 \\ k_4L_4 - k_5L_5 \\ k_5L_5 + k_6L - k_6L_6 \end{bmatrix} \tag{4.66}$$

The system of equations (4.66) is solved with a user-defined MATLAB function `Tridiagonal`, which is listed next.

Program 4-9: User-defined function. Solving a tridiagonal system of equations.

```
function x = Tridiagonal(A,B)
% The function solves a tridiagonal system of linear equations [a][x]=[b]
% using the Thomas algorithm.
% Input variables:
```

```
% A  The matrix of coefficients.
% B  Right-hand-side column vector of constants.
% Output variable:
% x  A column vector with the solution.

[nR, nC]=size(A);
for i=1:nR
   d(i)=A(i,i);          Define the vector d with the elements of the diagonal.
end
for i=1:nR-1
   ad(i)=A(i,i+1);       Define the vector ad with the above diagonal elements.     Step 1.
end
for i=2:nR
   bd(i)=A(i,i-1);       Define the vector bd with the below diagonal elements.
end
ad(1)=ad(1)/d(1);
B(1)=B(1)/d(1);                                                                 Step 2.
for i=2:nR-1
   ad(i)=ad(i)/(d(i)-bd(i)*ad(i-1));
   B(i)=(B(i)-bd(i)*B(i-1))/(d(i)-bd(i)*ad(i-1));                               Step 3.
end
B(nR)=(B(nR)-bd(nR)*B(nR-1))/(d(nR)-bd(nR)*ad(nR - 1));                         Step 4.
x(nR,1)=B(nR);
for i=nR-1:-1:1
   x(i,1)=B(i)-ad(i)*x(i+1);                                                    Step 5.
end
```

The user-defined function `Tridiagonal` is next used in a script file program to solve the system in Eq. (4.66).

```
% Example 4-9
k1=8000; k2=9000; k3=15000; k4=12000; k5=10000; k6=18000;
L=1.5; L1=0.18; L2=0.22; L3=0.26; L4=0.19; L5=0.15; L6=0.30;
a=[k1 + k2, -k2, 0, 0, 0; -k2, k2+k3, -k3, 0, 0; 0, -k3, k3+k4, -k4, 0
    0, 0, -k4, k4+k5, -k5; 0, 0, 0, -k5, k5+k6];
b=[k1*L1-k2*L2; k2*L2-k3*L3; k3*L3-k4*L4; k4*L4-k5*L5; k5*L5+k6*L-k6*L6];
Xs=Tridiagonal(a,b)
```

When the script file is executed, the following solution is displayed in the Command Window.

```
Xs =
   0.2262
   0.4872
   0.7718
   0.9926
   1.1795
>>
```

$$\begin{bmatrix}x_1\\x_2\\x_3\\x_4\\x_5\end{bmatrix}$$

4.10 ERROR, RESIDUAL, NORMS, AND CONDITION NUMBER

A numerical solution of a system of equations is seldom an exact solution. Even though direct methods (Gauss, Gauss–Jordan, *LU* decomposition) can be exact, they are still susceptible to round-off errors when implemented on a computer. This is especially true with large systems and with ill-conditioned systems (see Section 4.11). Solutions that are obtained with iterative methods are approximate by nature. This section describes measures that can be used for quantifying the accuracy, or estimating the magnitude of the error, of a numerical solution.

4.10.1 Error and Residual

If $[x_{NS}]$ is a computed approximate numerical solution of a system of n equations $[a][x]=[b]$ and $[x_{TS}]$ is the true (exact) solution, then the true error is the vector:

$$[e] = [x_{TS}]-[x_{NS}] \tag{4.67}$$

The true error, however, cannot in general be calculated because the true solution is not known.

An alternative measure of the accuracy of a solution is the residual $[r]$, which is defined by:

$$[r] = [a][x_{TS}]-[a][x_{NS}] = [b]-[a][x_{NS}] \tag{4.68}$$

In words, $[r]$ measures how well the system of equations is satisfied when $[x_{NS}]$ is substituted for $[x]$. (This is equivalent to the tolerance in $f(x)$ when the solution of a single equation is considered. See Eq. (3.5) in Section 3.2.) The vector $[r]$ has n elements, and if the numerical solution is close to the true solution, then all the elements of $[r]$ are small. It should be remembered that $[r]$ does not really indicate how small the error is in the solution $[x]$. $[r]$ only shows how well the right-hand side of the equations is satisfied when $[x_{NS}]$ is substituted for $[x]$ in the original equations. This depends on the magnitude of the elements of the matrix $[a]$. As shown next in Example 4-10, it is possible to have an approximate numerical solution that has a large true error but gives a small residual.

A more accurate estimate of the error in a numerical solution can be obtained by using quantities that measure the size, or magnitude, of vectors and matrices. For numbers, it is easy to determine which one is large or small by comparing their absolute values. It is more difficult to measure the magnitude (size) of vectors and matrices. This is done by a quantity called ***norm***, which is introduced next.

Example 4-10: Error and residual.

The true (exact) solution of the system of equations:

$$1.02x_1 + 0.98x_2 = 2$$
$$0.98x_1 + 1.02x_2 = 2$$

is $x_1 = x_2 = 1$.

Calculate the true error and the residual for the following two approximate solutions:

(*a*) $x_1 = 1.02$, $x_2 = 1.02$.

(b) $x_1 = 2$, $x_2 = 0$.

SOLUTION

In matrix form, the given system of equations is $[a][x] = [b]$, where $[a] = \begin{bmatrix} 1.02 & 0.98 \\ 0.98 & 1.02 \end{bmatrix}$ and $[b] = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$.

The true solution is $[x_{TS}] = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

The true error and the residual are given by Eqs. (4.67) and (4.68), respectively. Applying these equations to the two approximate solutions gives:

(*a*) In this case $[x_{NS}] = \begin{bmatrix} 1.02 \\ 1.02 \end{bmatrix}$. Consequently, the error and residual are:

$$[e] = [x_{TS}] - [x_{NS}] = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1.02 \\ 1.02 \end{bmatrix} = \begin{bmatrix} -0.02 \\ -0.02 \end{bmatrix} \quad \text{and}$$

$$[r] = [b] - [a][x_{NS}] = [b] - [a][x_{NS}] = \begin{bmatrix} 2 \\ 2 \end{bmatrix} - \begin{bmatrix} 1.02 & 0.98 \\ 0.98 & 1.02 \end{bmatrix}\begin{bmatrix} 1.02 \\ 1.02 \end{bmatrix} = \begin{bmatrix} -0.04 \\ -0.04 \end{bmatrix}$$

In this case, both the error and the residual are small.

(*b*) In this case, $[x_{NS}] = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$. Consequently, the error and residual are:

$$[e] = [x_{TS}] - [x_{NS}] = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad \text{and}$$

$$[r] = [b] - [a][x_{NS}] = [b] - [a][x_{NS}] = \begin{bmatrix} 2 \\ 2 \end{bmatrix} - \begin{bmatrix} 1.02 & 0.98 \\ 0.98 & 1.02 \end{bmatrix}\begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.04 \\ 0.04 \end{bmatrix}$$

In this case, the error is large but the residual is small.

This example shows that a small residual does not necessarily guarantee a small error. Whether or not a small residual implies a small error depends on the "magnitude" of the matrix $[a]$.

4.10.2 Norms and Condition Number

A ***norm*** is a real number assigned to a matrix or vector that satisfies the following four properties:

(*i*) The norm of a vector or matrix denoted by $\|[a]\|$ is a positive quantity. It is equal to zero only if the object $[a]$ itself is zero. In other words, $\|[a]\| \geq 0$ and $\|[a]\| = 0$ only if $[a] = 0$. This statement means that all vectors or matrices except for the zero vector or zero matrix have a positive magnitude.

(*ii*) For all numbers α, $\|\alpha[a]\| = |\alpha|\|[a]\|$. This statement means that the two objects $[a]$ and $[-a]$ have the same "magnitude" and that the magnitude of $[10a]$ is 10 times the magnitude of $[a]$.

(*iii*) For matrices and vectors, $\|[a][x]\| \leq \|[a]\|\|[x]\|$, which means that the norm of a product of two matrices is equal to or smaller than the product of the norms of each matrix.

(*iv*) For any two vectors or matrices $[a]$ and $[b]$,

$$\|[a+b]\| \leq \|[a]\| + \|[b]\| \tag{4.69}$$

This statement is known as the ***triangle inequality*** because for vectors $[a]$ and $[b]$ it states that the sum of the lengths of two sides of a triangle can never be smaller than the length of the third side.

Any norm of a vector or a matrix must satisfy the four properties listed above in order to qualify as a legitimate measure of its "magnitude." Different ways of calculating norms for vectors and matrices are described next.

Vector norms

For a given vector $[v]$ of n elements, the ***infinity norm*** written as $\|v\|_\infty$ is defined by:

$$\|v\|_\infty = \max_{1 \leq i \leq n} |v_i| \tag{4.70}$$

In words, $\|v\|_\infty$ is a number equal to the element v_i with the largest absolute value.

The ***1-norm*** written as $\|v\|_1$ is defined by:

$$\|v\|_1 = \sum_{i=1}^{n} |v_i| \tag{4.71}$$

In words, $\|v\|_1$ is the sum of the absolute values of the elements of the vector.

The ***Euclidean 2-norm*** written as $\|v\|_2$ is defined by:

$$\|v\|_2 = \left(\sum_{i=1}^{n} v_i^2\right)^{1/2} \tag{4.72}$$

In words, $\|v\|_2$ is the square root of the sum of the square of the elements. It is also called the magnitude of the vector $[v]$.

Matrix norms

The matrix ***infinity norm*** is given by:

$$\|[a]\|_\infty = \max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}| \tag{4.73}$$

In words, the absolute values of the elements in each row of the matrix are added. The value of the largest sum is assigned to $\|a\|_\infty$.

The matrix ***1-norm*** is calculated by:

$$\|[a]\|_1 = \max_{1 \le j \le n} \sum_{j=1}^{n} |a_{ij}| \tag{4.74}$$

It is similar to the infinity norm, except that the summation of the absolute values of the elements is done for each column, and the value of the largest sum is assigned to $\|a\|_1$.

The 2-norm of a matrix is evaluated as the spectral norm:

$$\|[a]\|_2 = \max\left(\frac{\|[a][v]\|}{\|[v]\|}\right) \tag{4.75}$$

where $[v]$ is an eigenvector of the matrix $[a]$ corresponding to an eigenvalue λ. (Eigenvalues and eigenvectors are covered in Chapter 5.) The 2-norm of a matrix is calculated by MATLAB using a technique called singular value decomposition, where the matrix $[a]$ is factored into $[a] = [u][d][v]$, where $[u]$ and $[v]$ are orthogonal matrices (special matrices with the property $[u]^{-1} = [u]^T$), and where $[d]$ is a diagonal matrix. The largest value of the diagonal elements of $[d]$ is used as the 2-norm of the matrix $[a]$.

The Euclidean norm for an $m \times n$ matrix $[a]$ (which is different from the 2-norm of a matrix) is given by:

$$\|[a]\|_{Euclidean} = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2\right)^{1/2} \tag{4.76}$$

Using norms to determine bounds on the error of numerical solutions

From Eqs. (4.67) and (4.68), the residual can be written in terms of the error $[e]$ as:

$$[r] = [a][x_{TS}] - [a][x_{NS}] = [a]([x_{TS}] - [x_{NS}]) = [a][e] \tag{4.77}$$

If the matrix $[a]$ is invertible (otherwise the system of equations does not have a solution), the error can be expressed as:

$$[e] = [a]^{-1}[r] \tag{4.78}$$

Applying property (*iii*) of the matrix norm to Eq. (4.78) gives:

$$\|[e]\| = \|[a]^{-1}[r]\| \le \|[a]^{-1}\|\|[r]\| \tag{4.79}$$

From Eq. (4.77), the residual $[r]$ is:

$$[r] = [a][e] \tag{4.80}$$

Applying property (*iii*) of the matrix norm to Eq. (4.80) gives:

$$\|[r]\| = \|[a][e]\| \le \|[a]\|\|[e]\| \tag{4.81}$$

The last equation can be rewritten as:

$$\frac{\|[r]\|}{\|[a]\|} \le \|[e]\| \tag{4.82}$$

Equations (4.79) and (4.82) can be combined and written in the form:

$$\frac{\|[r]\|}{\|[a]\|} \le \|[e]\| = \|[a]^{-1}[r]\| \le \|[a]^{-1}\|\|[r]\| \tag{4.83}$$

To use Eq. (4.83), two new quantities are defined. One is the ***relative error*** defined by $\|[e]\|/\|[x_{TS}]\|$, and the second is the ***relative residual*** defined by $\|[r]\|/\|[b]\|$. For an approximate numerical solution, the residual can be calculated from Eq. (4.68). With the residual known, Eq. (4.83) can be used for obtaining an upper bound and a lower bound on the relative error in terms of the relative residual. This is done by dividing Eq. (4.83) by $\|[x_{TS}]\|$, and rewriting the equation in the form:

$$\frac{1}{\|[a]\|}\frac{\|b\|}{\|[x_{TS}]\|}\frac{\|[r]\|}{\|[b]\|} \le \frac{\|[e]\|}{\|[x_{TS}]\|} \le \|[a]^{-1}\|\frac{\|[b]\|}{\|[x_{TS}]\|}\frac{\|[r]\|}{\|[b]\|} \tag{4.84}$$

Since $[a][x_{TS}] = [b]$, property (*iii*) of matrix norms gives: $\|[b]\| \le \|[a]\|\|[x_{TS}]\|$ or $\frac{\|[b]\|}{\|[x_{TS}]\|} \le \|[a]\|$, and this means that $\|[a]\|$ can be substituted for $\frac{\|[b]\|}{\|[x_{TS}]\|}$ in the right-hand side of Eq. (4.84). Similarly, since $[x_{TS}] = [a]^{-1}[b]$, property (*iii*) of matrix norms gives $\|[x_{TS}]\| \le \|[a]^{-1}\|\|[b]\|$ or $\frac{1}{\|[a]^{-1}\|} \le \frac{\|[b]\|}{\|[x_{TS}]\|}$, and this means that $\frac{1}{\|[a]^{-1}\|}$ can be substituted for $\frac{\|[b]\|}{\|[x_{TS}]\|}$ in the left-hand side of Eq. (4.84). With these substitutions, Eq. (4.84) becomes:

$$\frac{1}{\|[a]\|\|[a]^{-1}\|}\frac{\|[r]\|}{\|[b]\|} \le \frac{\|[e]\|}{\|[x_{TS}]\|} \le \|[a]^{-1}\|\|[a]\|\frac{\|[r]\|}{\|[b]\|} \tag{4.85}$$

Equation (4.85) is the main result of this section. It provides a means for bounding the error in a numerical solution of a system of equations. Equation (4.85) states that the true relative error, $\frac{\|[e]\|}{\|[x_{TS}]\|}$ (which is not known), is bounded between $\frac{1}{\|[a]\|\|[a]^{-1}\|}$ times the relative residual, $\frac{\|[r]\|}{\|[b]\|}$ (lower bound), and $\|[a]^{-1}\|\|[a]\|$ times the relative residual (upper bound). The relative residual can be calculated from the approximate numerical solution so that the true relative error can be bounded if the quantity $\|[a]\|\|[a]^{-1}\|$ (called condition number) can be calculated.

Condition number

The number $\|[a]\|\|[a]^{-1}\|$ is called the ***condition number*** of the matrix $[a]$. It is written as:

$$Cond[a] = \|[a]\|\|[a]^{-1}\| \tag{4.86}$$

- The condition number of the identity matrix is 1. The condition number of any other matrix is 1 or greater.
- If the condition number is approximately 1, then the true relative error is of the same order of magnitude as the relative residual.
- If the condition number is much larger than 1, then a small relative residual does not necessarily imply a small true relative error.
- For a given matrix, the value of the condition number depends on the matrix norm that is used.
- The inverse of a matrix has to be known in order to calculate the condition number of the matrix.

Example 4-11 illustrates the calculation of error, residual, norms, and condition number.

Example 4-11: Calculating error, residual, norm and condition number.

Consider the following set of four equations (the same that was solved in Example 4-8).

$$\begin{aligned} 9x_1 - 2x_2 + 3x_3 + 2x_4 &= 54.5 \\ 2x_1 + 8x_2 - 2x_3 + 3x_4 &= -14 \\ -3x_1 + 2x_2 + 11x_3 - 4x_4 &= 12.5 \\ -2x_1 + 3x_2 + 2x_3 + 10x_4 &= -21 \end{aligned}$$

The true solution of this system is $x_1 = 5$, $x_2 = -2$, $x_3 = 2.5$, and $x_4 = -1$. When this system was solved in Example 4-8 with the Gauss–Seidel iteration method, the numerical solution in the sixth iteration was $x_1 = 4.98805$, $x_2 = -1.99511$, $x_3 = 2.49806$, and $x_4 = -1.00347$.

(*a*) Determine the true error, $[e]$, and the residual, $[r]$.

(*b*) Determine the infinity norms of the true solution, $[x_{TS}]$, the error, $[e]$, the residual, $[r]$, and the vector $[b]$.

(*c*) Determine the inverse of $[a]$, the infinity norm of $[a]$ and $[a]^{-1}$, and the condition number of the matrix $[a]$.

(*d*) Substitute the quantities from parts (*b*) and (*c*) in Eq. (4.85) and discuss the results.

SOLUTION

First, the equations are written in matrix form:

$$\begin{bmatrix} 9 & -2 & 3 & 2 \\ 2 & 8 & -2 & 3 \\ -3 & 2 & 11 & -4 \\ -2 & 3 & 2 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 54.5 \\ -14 \\ 12.5 \\ -21 \end{bmatrix}$$

(*a*) The true solution is $x_{TS} = \begin{bmatrix} 5 \\ -2 \\ 2.5 \\ -1 \end{bmatrix}$, and the approximate numerical solution is $x_{NS} = \begin{bmatrix} 4.98805 \\ -1.99511 \\ 2.49806 \\ -1.00347 \end{bmatrix}$.

The error is then: $[e] = [x_{TS}] - [x_{NS}] = \begin{bmatrix} 5 \\ -2 \\ 2.5 \\ -1 \end{bmatrix} - \begin{bmatrix} 4.98805 \\ -1.99511 \\ 2.49806 \\ -1.00347 \end{bmatrix} = \begin{bmatrix} 0.0119 \\ -0.0049 \\ 0.0019 \\ 0.0035 \end{bmatrix}$.

The residual is given by Eq. (4.77) $[r] = [a][e]$. It is calculated with MATLAB (Command Window):

```
>> a=[9 -2 3 2; 2 8 -2 3; -3 2 11 -4; -2 3 2 10];
>> e=[0.0119; -0.0049; 0.0019; 0.0035];
>> r=a*e
r =
   0.13009000000000
  -0.00869000000000
  -0.03817000000000
   0.00001000000000
```

(*b*) The infinity norm of a vector is defined in Eq. (4.70): $\|v\|_\infty = \max\limits_{1 \le i \le n} |v_i|$. Using this equation to calculate infinity norm of the true solution, the residual, and the vector $[b]$ gives:

$$\|x_{TS}\|_\infty = \max\limits_{1 \le i \le 4} |x_{TS\,i}| = max[|5|, |-2|, |2.5|, |-1|] = 5$$

$$\|e\|_\infty = \max\limits_{1 \le i \le 4} |e_i| = max[|0.0119|, |-0.0049|, |0.0019|, |0.0035|] = 0.0119$$

$$\|r\|_\infty = \max\limits_{1 \le i \le 4} |r_i| = max[|0.13009|, |-0.00869|, |-0.03817|, |0.00001|] = 0.13009$$

$$\|b\|_\infty = \max\limits_{1 \le i \le 4} |b_i| = max[|54.5|, |-14|, |12.5|, |-21|] = 54.5$$

(*c*) The inverse of $[a]$ is calculated by using MATLAB's `inv` function (Command Window):

```
>> aINV=inv(a)
aINV =
   0.0910    0.0386   -0.0116   -0.0344
  -0.0206    0.1194    0.0308   -0.0194
   0.0349   -0.0200    0.0727    0.0281
   0.0174   -0.0241   -0.0261    0.0933
```

The infinity norms of $[a]$ and $[a]^{-1}$ are calculated by using Eq. (4.73), $\|[a]\|_\infty = \max\limits_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}|$:

$$\|[a]\|_\infty = \max_{1 \le i \le 4} \sum_{j=1}^{n} |a_{ij}| = max[|9|+|-2|+|3|+|2|, |2|+|8|+|-2|+|3|, |-3|+|2|+|11|+|-4|, |9|+|-2|+|3|+|2|]$$

$$\|[a]\|_\infty = max[16, 15, 20, 16] = 20$$

$$\left\|[a]^{-1}\right\|_\infty = \max_{1 \le i \le 4} \sum_{j=1}^{n} |a_{ij}^{-1}| = max[|0.091| + |0.0386| + |-0.0116| + |-0.0344|, |-0.0206| + |0.1194| +$$
$$|0.0308| + |-0.0194|, |0.0349| + |-0.02| + |0.0727| + |0.0281|, |0.0174| + |-0.0241| + |-0.0261| + |0.0933|]$$

$$\left\|[a]^{-1}\right\|_\infty = max[0.1756, 0.1902, 0.1557, 0.1609] = 0.1902$$

The condition number of the matrix $[a]$ is calculated by using Eq. (4.86):

$$Cond[a] = \|[a]\|\left\|[a]^{-1}\right\| = 20 \cdot 0.1902 = 3.804$$

(*d*) Substituting all the variables calculated in parts (*b*) and (*c*) in Eq.(4.85) gives:

$$\frac{1}{\|[a]\|\left\|[a]^{-1}\right\|}\frac{\|[r]\|}{\|[b]\|} \le \frac{\|[e]\|}{\|[x_{TS}]\|} \le \left\|[a]^{-1}\right\|\|[a]\|\frac{\|[r]\|}{\|[b]\|}$$

$$\frac{1}{3.804}\frac{0.13009}{54.5} \le \frac{\|[e]\|}{\|[x_{TS}]\|} \le 3.804\frac{0.13009}{54.5}$$

$$\frac{1}{3.804}0.002387 \le \frac{\|[e]\|}{\|[x_{TS}]\|} \le 3.804 \cdot 0.002387, \quad \text{or} \quad 6.275 \times 10^{-4} \le \frac{\|[e]\|}{\|[x_{TS}]\|} \le 0.00908$$

These results indicate that the magnitude of the true relative error is between 6.275×10^{-4} and 0.00908. In this problem, the magnitude of the true relative error can be calculated because the true solution is known.

The magnitude of the true relative error is:

$$\frac{\|[e]\|}{\|[x_{TS}]\|} = \frac{0.0119}{5} = 0.00238$$, which is within the bounds calculated by Eq. (4.85).

4.11 ILL-CONDITIONED SYSTEMS

An ill-conditioned system of equations is one in which small variations in the coefficients cause large changes in the solution. The matrix of coefficients of ill-conditioned systems generally has a condition number that is significantly greater than 1. As an example, consider the system:

$$\begin{aligned} 6x_1 - 2x_2 &= 10 \\ 11.5x_1 - 3.85x_2 &= 17 \end{aligned} \tag{4.87}$$

The solution of this system is:

$$x_1 = \frac{a_{12}b_2 - a_{22}b_1}{a_{12}a_{21} - a_{11}a_{22}} = \frac{-2 \cdot 17 - (-3.85 \cdot 10)}{-2 \cdot 11.5 - (6 \cdot -3.85)} = \frac{4.5}{0.1} = 45$$

$$x_2 = \frac{a_{21}b_1 - a_{11}b_2}{a_{12}a_{21} - a_{11}a_{22}} = \frac{11.5 \cdot 10 - (6 \cdot 17)}{-2 \cdot 11.5 - (6 \cdot -3.85)} = \frac{13}{0.1} = 130$$

If a small change is made in the system by changing a_{22} to 3.84,

$$\begin{aligned} 6x_1 - 2x_2 &= 10 \\ 11.5x_1 - 3.84x_2 &= 17 \end{aligned} \tag{4.88}$$

then the solution is:

$$x_1 = \frac{a_{12}b_2 - a_{22}b_1}{a_{12}a_{21} - a_{11}a_{22}} = \frac{-2 \cdot 17 - (-3.84 \cdot 10)}{-2 \cdot 11.5 - (6 \cdot -3.84)} = \frac{4.4}{0.04} = 110$$

$$x_2 = \frac{a_{21}b_1 - a_{11}b_2}{a_{12}a_{21} - a_{11}a_{22}} = \frac{11.5 \cdot 10 - (6 \cdot 17)}{-2 \cdot 11.5 - (6 \cdot -3.84)} = \frac{13}{0.04} = 325$$

It can be observed that there is a very large difference between the solutions of the two systems. A careful examination of the solutions of Eqs. (4.87) and (4.88) shows that the numerator of the equation for x_2 in both solutions is the same and that there is only a small difference in the numerator of the equation for x_1. At the same time, there is a large difference (a factor of 2.5) between the denominators of the two equations. The denominators of both equations are the determinants of the matrices of coefficients $[a]$.

The fact that the system in Eqs. (4.87) is ill-conditioned is evident from the value of the condition number. For this system:

$$[a] = \begin{bmatrix} 6 & -2 \\ 11.5 & -3.85 \end{bmatrix} \quad \text{and} \quad [a]^{-1} = \begin{bmatrix} 38.5 & -20 \\ 115 & -60 \end{bmatrix}$$

Using the infinity norm, Eq. (4.73), the condition number for the system is:

$$Cond[a] = \|[a]\|\|[a]^{-1}\| = 15.35 \cdot 175 = 2686.25$$

Using the 1-norm, Eq. (4.74), the condition number for the system is:

$$Cond[a] = \|[a]\|\|[a]^{-1}\| = 17.5 \cdot 153.5 = 2686.25$$

Using the 2-norm, the condition number for the system is (the norms were calculated with MATLAB built-in `norm(a,2)` function):

$$Cond[a] = \|[a]\|\|[a]^{-1}\| = 13.6774 \cdot 136.774 = 1870.7$$

These results show that with any norm used, the condition number of the matrix of coefficients of the system in Eqs. (4.87) is much larger than 1. This means that the system is likely ill-conditioned.

When an ill-conditioned system of equations is being solved numerically, there is a high probability that the solution obtained will have a large error or that a solution will not be obtained at all. In general, it is difficult to quantify the value of the condition number that can precisely identify an ill-conditioned system. This depends on the precision of the computer used and other factors. Thus, in practice, one needs to worry only about whether or not the condition number is much larger than 1, and not about its exact value. Furthermore, it might not be possible to calculate the determinant and the condition number for an ill-conditioned system anyway because the mathematical operations done in these calculations are similar to the operations required in solving the system.

4.12 PROBLEMS

Problems to be solved by hand

Solve the following problems by hand. When needed, use a calculator, or write a MATLAB script file to carry out the calculations. If using MATLAB, do not use built-in functions for operations with matrices.

4.1 Solve the following system of equations using the Gauss elimination method:

$$\begin{aligned} 2x_1 + x_2 - x_3 &= 1 \\ x_1 + 2x_2 + x_3 &= 8 \\ -x_1 + x_2 - x_3 &= -5 \end{aligned}$$

4.2 Given the system of equations $[a][x] = [b]$, where $a = \begin{bmatrix} 2 & -2 & 1 \\ 3 & 2 & -5 \\ -1 & 2 & 3 \end{bmatrix}$, $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$, and $b = \begin{bmatrix} 10 \\ -16 \\ 8 \end{bmatrix}$,
determine the solution using the Gauss elimination method.

4.3 Consider the following system of two linear equations: $\begin{aligned} 0.0003x_1 + 1.566x_2 &= 1.569 \\ 0.3454x_1 - 2.436x_2 &= 1.018 \end{aligned}$.

(*a*) Solve the system with the Gauss elimination method using rounding with four significant figures.

(*b*) Switch the order of the equations, and solve the system with the Gauss elimination method using rounding with four significant figures.

Check the answers by substituting the solution back in the equations.

4.4 Solve the following system of equations using the Gauss elimination method.

$$\begin{aligned} 2x_1 + x_2 - x_3 + 2x_4 &= 0 \\ x_1 - 2x_2 + x_3 - 4x_4 &= 3 \\ 3x_1 - x_2 - 2x_3 - x_4 &= -3 \\ -x_1 + 2x_2 + x_3 - 2x_4 &= 13 \end{aligned}$$

4.5 Solve the following system of equations with the Gauss elimination method.

$$\begin{aligned} 2x_1 + x_2 - x_3 + 4x_4 &= 19 \\ -x_1 - 2x_2 + x_3 + 2x_4 &= -3 \\ 2x_1 + 4x_2 + 2x_3 + x_4 &= 25 \\ -x_1 + x_2 - x_3 - 2x_4 &= -5 \end{aligned}$$

4.6 Solve the following system of equations using the Gauss–Jordan method.

$$\begin{aligned} 4x_1 + x_2 + 2x_3 &= 21 \\ 2x_1 - 2x_2 + 2x_3 &= 8 \\ x_1 - 2x_2 + 4x_3 &= 16 \end{aligned}$$

4.7 Solve the system of equations given in Problem 4.2 using the Gauss–Jordan method.

4.8 Given the system of equations $[a][x] = [b]$, where $a = \begin{bmatrix} 4 & 5 & -2 \\ 2 & -5 & 2 \\ 6 & 2 & 4 \end{bmatrix}$, $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$, and $b = \begin{bmatrix} -6 \\ 24 \\ 30 \end{bmatrix}$, determine the solution using the Gauss–Jordan method.

4.9 Solve the following system of equations with the Gauss–Jordan elimination method.

$$\begin{aligned} 4x_1 + 3x_2 + 2x_3 + x_4 &= 17 \\ 2x_1 - x_2 + 2x_3 - 4x_4 &= 11 \\ x_1 + 2x_2 - 2x_3 - x_4 &= 8 \\ -2x_1 + 4x_2 + 5x_3 - x_4 &= 15 \end{aligned}$$

4.10 Determine the *LU* decomposition of the matrix $a = \begin{bmatrix} 2 & 4 & 6 \\ 3 & 5 & 1 \\ 6 & -2 & 2 \end{bmatrix}$ using the Gauss elimination procedure.

4.11 Determine the *LU* decomposition of the matrix $a = \begin{bmatrix} 6 & 12 & 24 \\ 2 & 11 & 29 \\ 4 & 10 & 24 \end{bmatrix}$ using Crout's method.

4.12 Solve the following system with *LU* decomposition using Crout's method.

$$\begin{bmatrix} 2 & -6 & 6 \\ 3 & -7 & 13 \\ -2 & 2 & -11 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2 \\ -13 \\ 21 \end{bmatrix}$$

4.13 Find the inverse of the matrix $\begin{bmatrix} 10 & 12 & 0 \\ 0 & 2 & 8 \\ 2 & 4 & 8 \end{bmatrix}$ using the Gauss–Jordan method.

4.14 Given the matrix $a = \begin{bmatrix} -1 & 2 & 2 \\ 0 & 2 & -0.5 \\ 0.5 & 1 & -2 \end{bmatrix}$, determine the inverse of $[a]$ using the Gauss–Jordan method.

4.15 Carry out the first three iterations of the solution of the following system of equations using the Gauss–Seidel iterative method. For the first guess of the solution, take the value of all the unknowns to be zero.

$$\begin{aligned} 8x_1 + 2x_2 + 3x_3 &= 51 \\ 2x_1 + 5x_2 + x_3 &= 23 \\ -3x_1 + x_2 + 6x_3 &= 20 \end{aligned}$$

4.16 Carry out the first three iterations of the solution of the following system of equations using the Gauss–Seidel iterative method. For the first guess of the solution, take the value of all the unknowns to be zero.

$$\begin{bmatrix} 4 & 0 & 1 & 0 & 1 \\ 2 & 5 & -1 & 1 & 0 \\ 1 & 0 & 3 & -1 & 0 \\ 0 & 1 & 0 & 4 & -2 \\ 1 & 0 & -1 & 0 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 32 \\ 19 \\ 14 \\ -2 \\ 41 \end{bmatrix}$$

4.17 Find the condition number of the matrix in Problem 4.13 using the infinity norm.

4.18 Find the condition number of the matrix in Problem 4.14 using the infinity norm.

4.19 Find the condition number of the matrix in Problem 4.13 using the 1-norm.

4.20 Find the condition number of the matrix in Problem 4.14 using the 1-norm.

Problems to be programmed in MATLAB

Solve the following problems using the MATLAB environment. Do not use MATLAB's built-in functions for operations with matrices.

4.21 Modify the user-defined function `GaussPivot` in Program 4-2 (Example 4-3) such that in each step of the elimination the pivot row is switched with the row that has a pivot element with the largest absolute numerical value. For the function name and arguments use `x = GaussPivotLarge(a,b)`, where `a` is the matrix of coefficients, `b` is the right-hand-side column of constants, and `x` is the solution.
(*a*) Use the `GaussPivotLarge` function to solve the system of linear equations in Eq. (4.17).
(*b*) Use the `GaussPivotLarge` function to solve the system:

$$\begin{bmatrix} 0 & 3 & 8 & -5 & -1 & 6 \\ 3 & 12 & -4 & 8 & 5 & -2 \\ 8 & 0 & 0 & 10 & -3 & 7 \\ 3 & 1 & 0 & 0 & 0 & 4 \\ 0 & 0 & 4 & -6 & 0 & 2 \\ 3 & 0 & 5 & 0 & 0 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 34 \\ 20 \\ 45 \\ 36 \\ 60 \\ 28 \end{bmatrix}$$

4.22 Write a user-defined MATLAB function that solves a system of n linear equations, $[a][x] = [b]$, with the Gauss–Jordan method. The program should include pivoting in which the pivot row is switched with the row that has a pivot element with the largest absolute numerical value. For the function name and arguments use `x = GaussJordan(a,b)`, where `a` is the matrix of coefficients, `b` is the right-hand-side column of constants, and `x` is the solution.
(*a*) Use the `GaussJordan` function to solve the system:

$$\begin{aligned} 2x_1 + x_2 + 4x_3 - 2x_4 &= 19 \\ -3x_1 + 4x_2 + 2x_3 - x_4 &= 1 \\ 3x_1 + 5x_2 - 2x_3 + x_4 &= 8 \\ -2x_1 + 3x_2 + 2x_3 + 4x_4 &= 13 \end{aligned}$$

(*b*) Use the `GaussJordan` function to solve the system:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & -3 & 2 & 5 & -4 & 6 \\ 6 & 1 & -2 & 4 & 3 & 5 \\ 3 & 2 & -1 & 4 & 5 & 6 \\ 4 & -2 & -1 & 3 & 6 & 5 \\ 5 & -6 & -3 & 4 & -2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 91 \\ 37 \\ 63 \\ 81 \\ 69 \\ -4 \end{bmatrix}$$

4.23 Write a user-defined MATLAB function that decomposes an $n \times n$ matrix $[A]$ into a lower triangular matrix $[L]$ and an upper triangular matrix $[U]$ (such that $[A] = [L][U]$) using the Gauss elimination method (without pivoting). For the function name and arguments, use `[L,U] = LUdecompGauss(A)`, where the input argument `A` is the matrix to be decomposed and the output arguments `L` and `U` are the corresponding upper and lower triangular matrices. Use `LUdecompGauss` to determine the LU decomposition of the following matrix:

$$\begin{bmatrix} 4 & -1 & 3 & 2 \\ -8 & 0 & -3 & -3.5 \\ 2 & -3.5 & 10 & 3.75 \\ -8 & -4 & 1 & -0.5 \end{bmatrix}$$

4.24 Write a user-defined MATLAB function that determines the inverse of a matrix using the Gauss–Jordan method. For the function name and arguments use `Ainv = Inverse(A)`, where `A` is the matrix to be inverted, and `Ainv` is the inverse of the matrix. Use the `Inverse` function to calculate the inverse of:

(*a*) The matrix $\begin{bmatrix} -1 & 2 & 1 \\ 2 & 2 & -4 \\ 0.2 & 1 & 0.5 \end{bmatrix}$. (*b*) The matrix $\begin{bmatrix} -1 & -2 & 1 & 2 \\ 1 & 1 & -4 & -2 \\ 1 & -2 & -4 & -2 \\ 2 & -4 & 1 & -2 \end{bmatrix}$.

4.25 Write a user-defined MATLAB function that calculates the 1-norm of any matrix. For the function name and arguments use `N = OneNorm(A)`, where `A` is the matrix and `N` is the value of the norm. Use the function for calculating the 1-norm of:

(*a*) The matrix $A = \begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -1.5 \end{bmatrix}$. (*b*) The matrix $B = \begin{bmatrix} 4 & -1 & 0 & 1 & 0 \\ -1 & 4 & -1 & 0 & 1 \\ 0 & -1 & 4 & -1 & 0 \\ 1 & 0 & -1 & 4 & -1 \\ 0 & 1 & 0 & -1 & 4 \end{bmatrix}$.

4.26 Write a user-defined MATLAB function that calculates the infinity norm of any matrix. For the function name and arguments use `N = InfinityNorm(A)`, where `A` is the matrix, and `N` is the value of the norm. Use the function for calculating the infinity norm of:

(*a*) The matrix $A = \begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -1.5 \end{bmatrix}$. (*b*) The matrix $B = \begin{bmatrix} 4 & -1 & 0 & 1 & 0 \\ -1 & 4 & -1 & 0 & 1 \\ 0 & -1 & 4 & -1 & 0 \\ 1 & 0 & -1 & 4 & -1 \\ 0 & 1 & 0 & -1 & 4 \end{bmatrix}$.

4.27 Write a user-defined MATLAB function that calculates the condition number of an $(n \times n)$ matrix by using the 1-norm. For the function name and arguments use `c = CondNumb_One(A)`, where `A` is the matrix and `c` is the value of the condition number. Within the function, use the user-defined functions `Inverse` from Problem 4.24 and `OneNorm` from Problem 4.25. Use the function `CondNumb_One` for calculating the condition number of the matrices in Problem 4.25.

4.28 Write a user-defined MATLAB function that calculates the condition number of an $(n \times n)$ matrix by using the infinity norm. For the function name and arguments use `c = CondNumb_Inf(A)`, where `A` is the matrix and `c` is the value of the condition number. Within the function, use the user-defined functions `Inverse` from Problem 4.24 and `InfinityNorm` from Problem 4.26. Use the function `CondNumb_Inf` for calculating the condition number of the matrices in Problem 4.25.

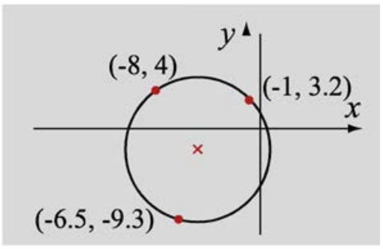
Problems in math, science, and engineering

Solve the following problems using the MATLAB environment. As stated, use the MATLAB programs that are presented in the chapter, programs developed in previously solved problems, or MATLAB's built-in functions.

4.29 In a Cartesian coordinate system the equation of a circle with its center at point (a, b) and radius r is:

$$(x-a)^2 + (y-b)^2 = r^2$$

Given three points, $(-1, 3.2)$, $(-8, 4)$, and $(-6.5, -9.3)$, determine the equation of the circle that passes through the points.



Solve the problem by deriving a system of three linear equations (substitute the points in the equation) and solve the system.

(*a*) Use the user-defined function `GaussPivotLarge` developed in Problem 4.21.

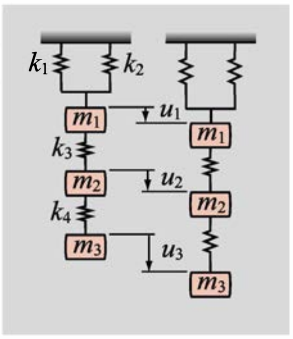
(*b*) Solve the system of equations using MATLAB's left division operation.

4.30 In a 3D Cartesian coordinate system the equation of a plane is:

$$ax + by + cz = d$$

Given three points, $(2, -3, -2)$, $(5, 2, 1)$, and $(-1, 5, 4)$, determine the equation of the plane that passes through the points.

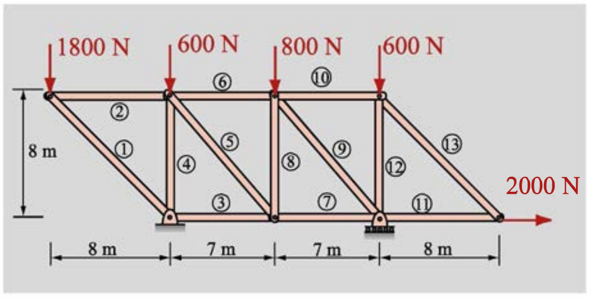
4.31 Three masses, $m_1 = 2$ kg, $m_2 = 3$ kg, and $m_3 = 1.5$ kg, are attached to springs, $k_1 = 30$ N/m, $k_2 = 25$ N/m, $k_3 = 20$ N/m, and $k_4 = 15$ N/m, as shown. Initially the masses are positioned such that the springs are in their natural length (not stretched or compressed); then the masses are slowly released and move downward to an equilibrium position as shown. The equilibrium equations of the three masses are:



$$(k_1 + k_2 + k_3)u_1 - k_3 u_2 = m_1 g$$
$$-k_3 u_1 + (k_3 + k_4)u_2 - k_4 u_3 = m_2 g$$
$$-k_4 u_2 + k_4 u_3 = m_3 g$$

where u_1, u_2, and u_3 are the relative displacement of each mass as shown. Determine the displacement of the three masses. ($g = 9.81\ m/s^2$)

4.32 The axial force F_i in each of the 13-member pin-connected truss, shown in the figure, can be calculated by solving the following system of 13 equations:



$F_2 + 0.707F_1 = 0$, $F_3 - 0.707F_1 - 2000 = 0$

$0.7071F_1 + F_4 + 6229 = 0$, $-F_2 + 0.659F_5 + F_6 = 0$

$-F_4 - 0.753F_5 - 600 = 0$, $-F_3 - 0.659F_5 + F_7 = 0$

$0.753F_5 + F_8 = 0$, $-F_6 + 0.659F_9 + F_{10} = 0$

$-F_8 - 0.753F_9 - 800 = 0$, $-F_7 - 0.659F_9 + F_{11} = 0$

$0.753F_9 + F_{12} - 2429 = 0$, $-F_{10} + 0.707F_{13} = 0$

$-F_{12} - 0.7071F_{13} - 600 = 0$

(*a*) Solve the system of equations using the user-defined function `GaussPivotLarge` developed in Problem 4.21.

(*b*) Solve the system of equations using Gauss–Seidel iteration. Does the solution converge for a starting (guess) vector whose elements are all zero?

(*c*) Solve the system of equations using MATLAB's left division operation.

4.33 A particular dessert consists of 2 lb of bananas, 3 lb of strawberries, 3 lb of cherries, and 4 lb of frozen yogurt. If the cost of the entire batch of this dessert is to be no more than \$20 (in order to yield an acceptable profit), what must the cost of each ingredient be (per pound) if the strawberries cost twice as much as the cherries, and the cherries cost \$1 per pound less than the frozen yogurt, and the frozen yogurt costs as much as half a pound of cherries and 4 pounds of bananas? (Hint: Set up a system of four equations where the unknowns are the cost (per pound) of the bananas (x_1), the cost (per pound) of the strawberries (x_2), the cost (per pound) of the cherries (x_3), the cost (per pound) of the frozen yogurt (x_4), and use the fact that all the ingredient costs have to add up to \$20.)

4.34 A particular chemical substance is produced from three different ingredients A, B, and C, each of which have be dissolved in water first before they react to form the desired substance. Suppose that a solution containing ingredient A at a concentration of 2 g/cm^3 is combined with a solution containing ingredient B at a concentration of 3.6 g/cm^3 and with a solution containing ingredient C at a concentration of 6.3 g/cm^3 to form 25.4 g of the substance. If the concentrations of A, B, and C in these solutions are changed to 4 g/cm^3, 4.3 g/cm^3, and 5.4 g/cm^3, respectively (while the volumes remain the same), then 27.7 g of the substance is produced. Finally, if the concentrations are changed to 7.2, 5.5, and 2.3 g/cm^3, respectively, then 28.3 g of the chemical is produced. Find the volumes (in cubic centimeters) of the solutions containing A, B, and C.

4.35 Mass spectrometry of a sample gives a series of peaks that represent various masses of ions of constituents within the sample. For each peak, the height of the peak I_i is influenced by the amounts of the various constituents:

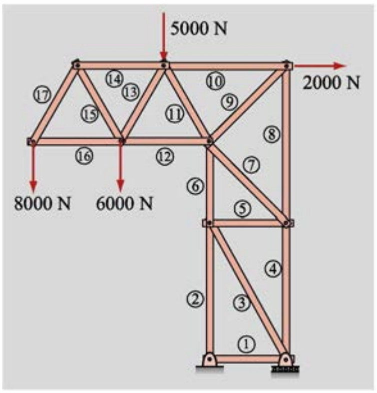
$$I_j = \sum_{i=1}^{N} C_{ij} n_j$$

where C_{ij} is the contribution of ions of species i to the height of peak j, and n_j is the amount of ions or concentration of species j. The coefficients C_{ij} for each peak are given by:

Peak identity	Species				
	CH_4	C_2H_4	C_2H_6	C_3H_6	C_3H_8
1	2	0.5	0	2.4	0.2
2	18	4	0.3	0.2	0.1
3		18	10	0	15
4			12	0	1
5				10	2
6					10

If a sample produces a mass spectrum with peak heights, $I_1 = 30.5$, $I_2 = 71.5$, $I_3 = 354.8$, $I_4 = 180$, $I_5 = 100$, and $I_6 = 36.9$, determine the concentrations of the different species in the sample.

4.36 The axial force F_i in each of the 21 member pin connected truss, shown in the figure, can be calculated by solving the following system of 21 equations:



$-F_1 - 0.342F_3 = 0$, $0.94F_3 + F_4 - 54000 = 0$

$F_5 + 0.342F_3 = 0$, $F_6 - F_2 - 0.94F_3 = 0$

$-F_5 - 0.7071F_7 = 0$, $F_8 + 0.707F_7 - F_4 = 0$

$0.707F_9 + 0.707F_5 - 0.5F_{11} - F_{12} = 0$

$-F_6 - 0.707F_7 + 0.7071F_9 + 0.866F_{11} = 0$

$-F_{10} - 0.707F_9 + 2000 = 0$, $-F_8 - 0.707F_9 = 0$

$F_{10} + 0.5F_{11} - 0.5F_{13} - F_{14} = 0$, $-0.866F_{11} - 0.866F_{13} - 5000 = 0$

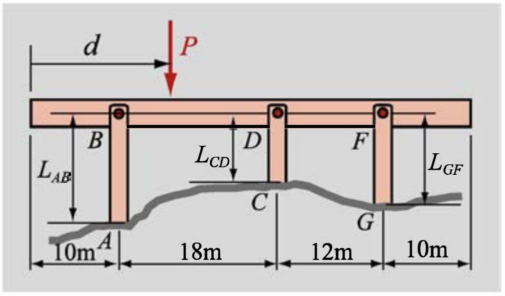
$F_{12} + 0.5F_{13} - 0.5F_{15} - F_{16} = 0$, $0.866F_{13} + 0.866F_{15} - 6000 = 0$

$F_{16} + 0.5F_{17} = 0$, $F_{14} + 0.5F_{15} - 0.5F_{17} = 0$, $-0.866F_{15} - 0.866F_{17} = 0$

(*a*) Solve the system of equations using the user-defined function `GaussJordan` developed in Problem 4.22.

(*b*) Solve the system of equations using MATLAB's left division operation.

4.37 A bridge is modeled by a rigid horizontal bar supported by three elastic vertical columns as shown. A force $P = 40\,\text{kN}$ applied to the rigid bar at a distance d from the end of the bar represents a car on the bridge. The forces in columns F_{AB}, F_{CD}, and F_{GF} can be determined from the solution of the following system of three equations:



$$F_{AB} + F_{CD} + F_{GF} = -P, \quad 10F_{AB} + 28F_{CD} + 40F_{GF} = -d \cdot P$$

$$12F_{AB}L_{AB} - 30F_{CD}L_{CD} + 18F_{GF}L_{GF} = 0$$

Once the force in each of the column is known, its elongation δ can be determined with the formula $\delta = \frac{FL}{EA}$, where E and A are the elastic modulus and the cross-sectional area of each of the columns. Write a MATLAB program in a script file that determines the forces in the three columns and their elongation for $0 \le d \le 50$ m. The program displays the three forces as a function of d in one plot, and the elongation of the three columns as a function of d in a second plot (two plots on the same page). Also given: $L_{AB} = 12$ m, $L_{CD} = 8$ m, $L_{GF} = 10$ m, $E = 70$ GPa, and $A = 25 \cdot 10^{-4}$ m^2.

4.38 A food company manufactures the five types of 1.0 lb trail mix packages that have the following composition and cost:

Mix	Peanuts (lb)	Raisins (lb)	Almonds (lb)	Chocolate Chips (lb)	Dried Plums (lb)	Total Cost of Ingredients ($)
A	0.2	0.2	0.2	0.2	0.2	1.44
B	0.35	0.15	0.35	0	0.15	1.16
C	0.1	0.3	0.1	0.1	0.4	1.38
D	0	0.3	0.1	0.4	0.2	1.78
E	0.15	0.3	0.2	0.35	0	1.61

Using the information in the table, determine the cost per pound of each of the ingredients. Write a system of linear equations and solve by using the following methods.

(*a*) Use the user-defined function `GaussJordan` that was developed in Problem 4.22.

(*b*) Use MATLAB's built-in functions.

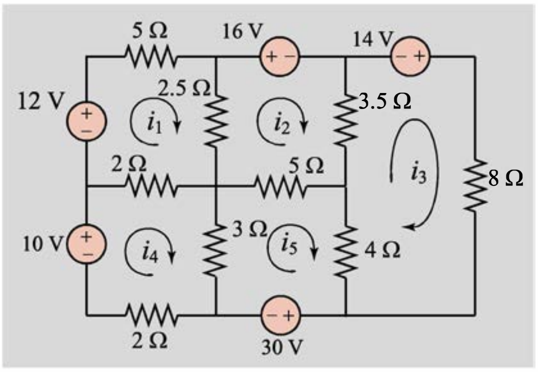
4.39 The currents, i_1, i_2, i_3, i_4, i_5, in the circuit that is shown can be determined from the solution of the following system of equations. (Obtained by applying Kirchhoff's law.)

$$9.5i_1 - 2.5i_2 - 2i_4 = 12 \ ,$$
$$-2.5i_1 + 11i_2 - 3.5i_3 - 5i_5 = -16$$
$$-3.5i_2 + 15.5i_3 - 4i_5 = 14 \ , \quad -2i_1 + 7i_4 - 3i_5 = 10$$
$$-5i_2 - 4i_3 - 3i_4 + 12i_5 = -30$$

Solve the system using the following methods.

(*a*) Use the user-defined function `GaussJordan` that was developed in Problem 4.22.

(*b*) Use MATLAB's built-in functions.



4.40 When balancing the following chemical reaction by conserving the number of atoms of each element between reactants and products:

$$P_2I_4 + aP_4 + bH_2O \leftrightarrows cPH_4I + dH_3PO_4$$

the unknown stoichiometric coefficients a, b, c, and d are given by the solution of the following system of equations:

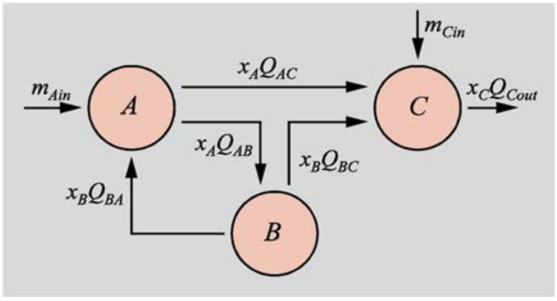
$$\begin{bmatrix} -4 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & -2 & 4 & 3 \\ 0 & -1 & 0 & 4 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 0 \\ 0 \end{bmatrix}$$

Solve for the unknown stoichiometric coefficients using

(*a*) The user-defined function `GaussJordan` that was developed in Problem 4.22.

(*b*) MATLAB's left division operation.

4.41 A certain chemical engineering process application (see figure) involves three chemical reactors A, B, and C. At steady state, the concentrations of a particular species n in each reactor has the values x_A, x_B, and x_C in units of mg/m^3. If the flow rates from reactor i (A, B, or C) to reactor j (A, B, or C) is denoted as Q_{ij} (units of m^3/s), then the mass flow rate of species n from reactor i to reactor j is x_iQ_{ij} (units of mg/s). Since this chemical species is conserved (i.e., neither produced nor destroyed) conservation of mass (of the species) for each reactor must hold. For the process shown in the figure, $Q_{AB} = 40$ m^3/s, $Q_{AC} = 80$ m^3/s, $Q_{BA} = 60$ m^3/s, $Q_{BC} = 20$ m^3/s, $Q_{Cout} = 150$ m^3/s, $m_{Cin} = 195$ mg/s, and $m_{Ain} = 1320$ mg/s. Write down the mass continuity equations for each reactor and solve them to find the concentrations x_A, x_B, and x_C in each reactor.



4.42 When balancing the following chemical reaction by conserving the number of atoms of each element between reactants and products:

$$(Cr(N_2H_4CO)_6)_4(Cr(CN)_6)_3 + aKMnO_4 + bH_2SO_4 \rightarrow$$

$$cK_2Cr_2O_7 + dMnSO_4 + eCO_2 + fKNO_3 + gK_2SO_4 + hHO_2$$

the unknown stoichiometric coefficients a through h are given by the solution of the following system of equations:

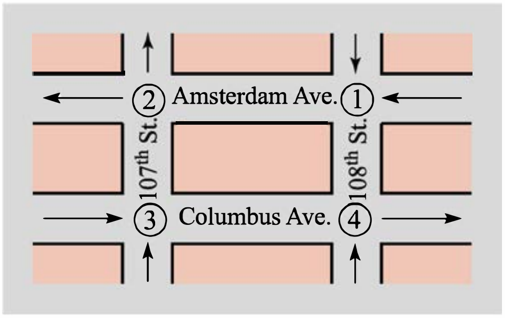
$$\begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -2 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ -4 & -4 & 7 & 4 & 2 & 3 & 4 & 1 \\ -1 & 0 & 2 & 0 & 0 & 1 & 2 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} 7 \\ 66 \\ 96 \\ 42 \\ 24 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Solve for the unknown stoichiometric coefficients using

(*a*) The user-defined function `GaussJordan` that was developed in Problem 4.22.

(*b*) MATLAB's left division operation.

4.43 Traffic congestion is encountered at the intersections shown in the figure. All the streets are one-way and in the directions shown. In order for effective movement of traffic, it is necessary that for every car that arrives at a given corner, another car must leave so that the number of cars arriving per unit time must equal the number of cars leaving per unit time. Traffic engineers gather the following information:



- 600 cars per hour come down Amsterdam Ave. to intersection #1 and 300 cars per hour enter intersection #1 on 108th St.
- 650 cars per hour leave intersection #2 along Amsterdam Ave. and 50 cars per hour leave intersection #2 along 107th St.
- 350 cars per hour come up Columbus Ave. to intersection #3 and 50 cars per hour enter intersection #3 along 107th St.
- 400 cars per hour leave intersection #4 along Columbus Ave. and 300 cars per hour enter intersection #4 from 108th St.

Find n_1, n_2, n_3, and n_4, where n_1 denotes the number of cars traveling per hour along Amsterdam Ave. from intersection #1 to intersection #2, n_2 denotes the number of cars traveling per hour along 107th St. from intersection #3 to intersection #2, n_3 denotes the number of cars traveling per hour along Columbus Ave. from intersection #3 to intersection #4, and n_4 denotes the number of cars traveling per hour along 108th St. from intersection #1 to intersection #4.