# CHAPTER 11

**11.1** First, the decomposition is implemented as

$e_2 = -0.4/0.8 = -0.5$
$f_2 = 0.8 - (-0.5)(-0.4) = 0.6$
$e_3 = -0.4/0.6 = -0.66667$
$f_3 = 0.8 - (-0.66667)(-0.4) = 0.53333$

Transformed system is

$$\begin{bmatrix} 0.8 & -0.4 & 0 \\ -0.5 & 0.6 & -0.4 \\ 0 & -0.66667 & 0.53333 \end{bmatrix}$$

which is decomposed as

$$[L] = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & -0.66667 & 1 \end{bmatrix} \qquad [U] = \begin{bmatrix} 0.8 & -0.4 & 0 \\ 0 & 0.6 & -0.4 \\ 0 & 0 & 0.53333 \end{bmatrix}$$

The right hand side becomes

$r_1 = 41$
$r_2 = 25 - (-0.5)(41) = 45.5$
$r_3 = 105 - (-0.66667)45.5 = 135.3333$

which can be used in conjunction with the $[U]$ matrix to perform back substitution and obtain the solution

$x_3 = 135.3333/0.53333 = 253.75$
$x_2 = (45.5 - (-0.4)253.75)/0.6 = 245$
$x_1 = (41 - (-0.4)245)/0.8 = 173.75$

**11.2** As in Example 11.1, the *LU* decomposition is

$$[L] = \begin{bmatrix} 1 & & & \\ -0.49 & 1 & & \\ & -0.645 & 1 & \\ & & -0.717 & 1 \end{bmatrix} \qquad [U] = \begin{bmatrix} 2.04 & -1 & & \\ & 1.550 & -1 & \\ & & 1.395 & -1 \\ & & & 1.323 \end{bmatrix}$$

To compute the first column of the inverse

$$[L]\{D\} = \begin{Bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{Bmatrix}$$

Solving this gives

$$\{D\} = \begin{Bmatrix} 1 \\ 0.490196 \\ 0.316296 \\ 0.226775 \end{Bmatrix}$$

Back substitution, $[U]\{X\} = \{D\}$, can then be implemented to give to first column of the inverse

$$\{X\} = \begin{Bmatrix} 0.755841 \\ 0.541916 \\ 0.349667 \\ 0.171406 \end{Bmatrix}$$

For the second column

$$[L]\{D\} = \begin{Bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{Bmatrix}$$

which leads to

$$\{X\} = \begin{Bmatrix} 0.541916 \\ 1.105509 \\ 0.713322 \\ 0.349667 \end{Bmatrix}$$

For the third column

$$[L]\{D\} = \begin{Bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{Bmatrix}$$

which leads to

$$\{X\} = \begin{Bmatrix} 1.25 \\ 2.5 \\ 1.25 \end{Bmatrix}$$

For the fourth column

$$[L]\{D\} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{Bmatrix}$$

which leads to

$$\{X\} = \begin{Bmatrix} 0.171406 \\ 0.349667 \\ 0.541916 \\ 0.755841 \end{Bmatrix}$$

Therefore, the matrix inverse is

$$[A]^{-1} = \begin{bmatrix} 0.755841 & 0.541916 & 0.349667 & 0.171406 \\ 0.541916 & 1.105509 & 0.713322 & 0.349667 \\ 0.349667 & 0.713322 & 1.105509 & 0.541916 \\ 0.171406 & 0.349667 & 0.541916 & 0.755841 \end{bmatrix}$$

**11.3** First, the decomposition is implemented as

$e_2 = -0.020875/2.01475 = -0.01036$
$f_2 = 2.014534$
$e_3 = -0.01036$
$f_3 = 2.014534$
$e_4 = -0.01036$
$f_4 = 2.014534$

Transformed system is

$$\begin{bmatrix} 2.01475 & -0.02875 & & \\ -0.01036 & 2.014534 & -0.02875 & \\ & -0.01036 & 2.014534 & -0.02875 \\ & & -0.01036 & 2.014534 \end{bmatrix}$$

which is decomposed as

$$[L] = \begin{bmatrix} 1 & & & \\ -0.01036 & 1 & & \\ & -0.01036 & 1 & \\ & & -0.01036 & 1 \end{bmatrix} \qquad [U] = \begin{bmatrix} 2.01475 & -0.02875 & & \\ & 2.014534 & -0.02875 & \\ & & 2.014534 & -0.02875 \\ & & & 2.014534 \end{bmatrix}$$

Forward substitution yields

$r_1 = 4.175$
$r_2 = 0.043258$
$r_3 = 0.000448$
$r_4 = 2.087505$

Back substitution

$x_4 = 1.036222$
$x_3 = 0.01096$
$x_2 = 0.021586$
$x_1 = 2.072441$

**11.4** We can use MATLAB to verify the results of Example 11.2,

```
>> L=[2.4495 0 0;6.1237 4.1833 0;22.454 20.916 6.1106]
L =
    2.4495         0         0
    6.1237    4.1833         0
   22.4540   20.9160    6.1106

>> L*L'
ans =
    6.0001   15.0000   55.0011
   15.0000   54.9997  224.9995
```

```
   55.0011  224.9995  979.0006
```

**11.5**

$$l_{11} = \sqrt{6} = 2.44949$$

$$l_{21} = \frac{15}{2.44949} = 6.123724$$

$$l_{22} = \sqrt{55 - 6.123724^2} = 4.1833$$

$$l_{31} = \frac{55}{2.44949} = 22.45366$$

$$l_{32} = \frac{225 - 6.123724(22.45366)}{4.1833} = 20.9165$$

$$l_{33} = \sqrt{979 - 22.45366^2 - 20.9165^2} = 6.110101$$

Thus, the Cholesky decomposition is

$$[L] = \begin{bmatrix} 2.44949 & & \\ 6.123724 & 4.1833 & \\ 22.45366 & 20.9165 & 6.110101 \end{bmatrix}$$

The solution can then be generated by first using forward substitution to modify the right-hand-side vector,

$$[L]\{D\} = \{B\}$$

which can be solved for

$$\{D\} = \begin{Bmatrix} 62.29869 \\ 48.78923 \\ 11.36915 \end{Bmatrix}$$

Then, we can use back substitution to determine the final solution,

$$[L]^T\{X\} = \{D\}$$

which can be solved for

$$\{D\} = \begin{Bmatrix} 2.478571 \\ 2.359286 \\ 1.860714 \end{Bmatrix}$$

**11.6**

$$l_{11} = \sqrt{8} = 2.828427$$

$$l_{21} = \frac{20}{2.828427} = 7.071068$$

$$l_{22} = \sqrt{80 - 7.071068^2} = 5.477226$$

$$l_{31} = \frac{15}{2.828427} = 5.303301$$

$$l_{32} = \frac{50 - 7.071068(5.303301)}{5.477226} = 2.282177$$

$$l_{33} = \sqrt{60 - 5.303301^2 - 2.282177^2} = 5.163978$$

Thus, the Cholesky decomposition is

$$[L] = \begin{bmatrix} 2.828427 & & \\ 7.071068 & 5.477226 & \\ 5.303301 & 2.282177 & 5.163978 \end{bmatrix}$$

**11.7** Using MATLAB:

```
>> A=[9 0 0;0 25 0;0 0 4];
>> U=chol(A)

U =
     3     0     0
     0     5     0
     0     0     2
```

Thus, the factorization of this diagonal matrix consists of another diagonal matrix where the elements are the square root of the original. This is consistent with Eqs. 11.3 and 11.4, which for a diagonal matrix reduce to

$$u_{ii} = \sqrt{a_{ii}}$$

$$u_{ij} = 0 \quad \text{for } i \neq j$$

**11.8 (a)** The first iteration can be implemented as

$$x_1 = \frac{41 + 0.4x_2}{0.8} = \frac{41 + 0.4(0)}{0.8} = 51.25$$

$$x_2 = \frac{25 + 0.4x_1 + 0.4x_3}{0.8} = \frac{25 + 0.4(51.25) + 0.4(0)}{0.8} = 56.875$$

$$x_3 = \frac{105 + 0.4x_2}{0.8} = \frac{105 + 0.4(56.875)}{0.8} = 159.6875$$

Second iteration:

$$x_1 = \frac{41 + 0.4(56.875)}{0.8} = 79.6875$$

$$x_2 = \frac{25 + 0.4(79.6875) + 0.4(159.6875)}{0.8} = 150.9375$$

$$x_3 = \frac{105 + 0.4(150.9375)}{0.8} = 206.7188$$

The error estimates can be computed as

$$\varepsilon_{a,1} = \left| \frac{79.6875 - 51.25}{79.6875} \right| \times 100\% = 35.69\%$$

$$\varepsilon_{a,2} = \left|\frac{150.9375 - 56.875}{150.9375}\right| \times 100\% = 62.32\%$$

$$\varepsilon_{a,3} = \left|\frac{206.7188 - 159.6875}{206.7188}\right| \times 100\% = 22.75\%$$

The remainder of the calculation proceeds until all the errors fall below the stopping criterion of 5%. The entire computation can be summarized as

| iteration | unknown | value | $\varepsilon_a$ | maximum $\varepsilon_a$ |
|---|---|---|---|---|
| 1 | $x$1 | 51.25 | 100.00% | |
| | $x$2 | 56.875 | 100.00% | |
| | $x$3 | 159.6875 | 100.00% | 100.00% |
| 2 | $x$1 | 79.6875 | 35.69% | |
| | $x$2 | 150.9375 | 62.32% | |
| | $x$3 | 206.7188 | 22.75% | 62.32% |
| 3 | $x$1 | 126.7188 | 37.11% | |
| | $x$2 | 197.9688 | 23.76% | |
| | $x$3 | 230.2344 | 10.21% | 37.11% |
| 4 | $x$1 | 150.2344 | 15.65% | |
| | $x$2 | 221.4844 | 10.62% | |
| | $x$3 | 241.9922 | 4.86% | 15.65% |
| 5 | $x$1 | 161.9922 | 7.26% | |
| | $x$2 | 233.2422 | 5.04% | |
| | $x$3 | 247.8711 | 2.37% | 7.26% |
| 6 | $x$1 | 167.8711 | 3.50% | |
| | $x$2 | 239.1211 | 2.46% | |
| | $x$3 | 250.8105 | 1.17% | 3.50% |

Thus, after 6 iterations, the maximum error is 3.5% and we arrive at the result: $x_1$ = 167.8711, $x_2$ = 239.1211 and $x_3$ = 250.8105.

**(b)** The same computation can be developed with relaxation where $\lambda$ = 1.2.

First iteration:

$$x_1 = \frac{41 + 0.4x_2}{0.8} = \frac{41 + 0.4(0)}{0.8} = 51.25$$

Relaxation yields: $x_1 = 1.2(51.25) - 0.2(0) = 61.5$

$$x_2 = \frac{25 + 0.4x_1 + 0.4x_3}{0.8} = \frac{25 + 0.4(61.5) + 0.4(0)}{0.8} = 62$$

Relaxation yields: $x_2 = 1.2(62) - 0.2(0) = 74.4$

$$x_3 = \frac{105 + 0.4x_2}{0.8} = \frac{105 + 0.4(74.4)}{0.8} = 168.45$$

Relaxation yields: $x_3 = 1.2(168.45) - 0.2(0) = 202.14$

Second iteration:

$$x_1 = \frac{41 + 0.4(74.4)}{0.8} = 88.45$$

Relaxation yields: $x_1 = 1.2(88.45) - 0.2(61.5) = 93.84$

$$x_2 = \frac{25 + 0.4(93.84) + 0.4(202.14)}{0.8} = 179.24$$

Relaxation yields: $x_2 = 1.2(179.24) - 0.2(74.4) = 200.208$

$$x_3 = \frac{105 + 0.4(200.208)}{0.8} = 231.354$$

Relaxation yields: $x_3 = 1.2(231.354) - 0.2(202.14) = 237.1968$

The error estimates can be computed as

$$\varepsilon_{a,1} = \left|\frac{93.84 - 61.5}{93.84}\right| \times 100\% = 34.46\%$$

$$\varepsilon_{a,2} = \left|\frac{200.208 - 74.4}{200.208}\right| \times 100\% = 62.84\%$$

$$\varepsilon_{a,3} = \left|\frac{237.1968 - 202.14}{237.1968}\right| \times 100\% = 14.78\%$$

The remainder of the calculation proceeds until all the errors fall below the stopping criterion of 5%. The entire computation can be summarized as

| iteration | unknown | value | relaxation | $\varepsilon_a$ | maximum $\varepsilon_a$ |
|---|---|---|---|---|---|
| 1 | *x*1 | 51.25 | 61.5 | 100.00% | |
| | *x*2 | 62 | 74.4 | 100.00% | |
| | *x*3 | 168.45 | 202.14 | 100.00% | 100.000% |
| 2 | *x*1 | 88.45 | 93.84 | 34.46% | |
| | *x*2 | 179.24 | 200.208 | 62.84% | |
| | *x*3 | 231.354 | 237.1968 | 14.78% | 62.839% |
| 3 | *x*1 | 151.354 | 162.8568 | 42.38% | |
| | *x*2 | 231.2768 | 237.49056 | 15.70% | |
| | *x*3 | 249.99528 | 252.55498 | 6.08% | 42.379% |
| 4 | *x*1 | 169.99528 | 171.42298 | 5.00% | |
| | *x*2 | 243.23898 | 244.38866 | 2.82% | |
| | *x*3 | 253.44433 | 253.6222 | 0.42% | 4.997% |

Thus, relaxation speeds up convergence. After 6 iterations, the maximum error is 4.997% and we arrive at the result: $x_1$ = 171.423, $x_2$ = 244.389 and $x_3$ = 253.622.

**11.9** The first iteration can be implemented as

$$c_1 = \frac{3300 + 3c_2 + c_3}{15} = \frac{3300 + 3(0) + 0}{15} = 220$$
$$c_2 = \frac{1200 + 3c_1 + 6c_3}{18} = \frac{1200 + 3(220) + 6(0)}{18} = 103.3333$$
$$c_3 = \frac{2400 + 4c_1 + c_2}{12} = \frac{2400 + 4(220) + 103.3333}{12} = 281.9444$$

Second iteration:

$$c_1 = \frac{3300 + 3c_2 + c_3}{15} = \frac{3300 + 3(103.3333) + 281.9444}{15} = 259.463$$
$$c_2 = \frac{1200 + 3c_1 + 6c_3}{18} = \frac{1200 + 3(259.463) + 6(281.9444)}{18} = 203.892$$
$$c_3 = \frac{2400 + 4c_1 + c_2}{12} = \frac{2350 + 4(259.463) + 203.892}{12} = 303.4787$$

The error estimates can be computed as

$$\varepsilon_{a,1} = \left|\frac{259.463 - 220}{259.463}\right| \times 100\% = 15.21\%$$
$$\varepsilon_{a,2} = \left|\frac{203.892 - 103.3333}{203.892}\right| \times 100\% = 49.32\%$$
$$\varepsilon_{a,3} = \left|\frac{303.4787 - 281.9444}{303.4787}\right| \times 100\% = 7.1\%$$

The remainder of the calculation proceeds until all the errors fall below the stopping criterion of 5%. The entire computation can be summarized as

| iteration | unknown | value | $\varepsilon_a$ | maximum $\varepsilon_a$ |
|---|---|---|---|---|
| 1 | $c_1$ | 220 | 100.00% | |
| | $c_2$ | 103.3333 | 100.00% | |
| | $c_3$ | 281.9444 | 100.00% | 100.00% |
| 2 | $c_1$ | 259.463 | 15.21% | |
| | $c_2$ | 203.892 | 49.32% | |
| | $c_3$ | 303.4787 | 7.10% | 49.32% |
| 3 | $c_1$ | 281.0103 | 7.67% | |
| | $c_2$ | 214.6613 | 5.02% | |
| | $c_3$ | 311.5585 | 2.59% | 7.67% |
| 4 | $c_1$ | 283.7028 | 0.95% | |
| | $c_2$ | 217.8033 | 1.44% | |
| | $c_3$ | 312.7179 | 0.37% | 1.44% |

Thus, after 4 iterations, the maximum error is 1.44% and we arrive at the result: $c_1$ = 283.7028, $c_2$ = 217.8033 and $c_3$ = 312.7179.

**11.10** The first iteration can be implemented as

$$c_1 = \frac{3800 + 3c_2 + c_3}{15} = \frac{3300 + 3(0) + 0}{15} = 220$$

$$c_2 = \frac{1200 + 3c_1 + 6c_3}{18} = \frac{1200 + 3(0) + 6(0)}{18} = 66.6667$$

$$c_3 = \frac{2400 + 4c_1 + c_2}{12} = \frac{2350 + 4(0) + 0}{12} = 200$$

Second iteration:

$$c_1 = \frac{3300 + 3c_2 + c_3}{15} = \frac{3300 + 3(66.6667) + 200}{15} = 246.6667$$

$$c_2 = \frac{1200 + 3c_1 + 6c_3}{18} = \frac{1200 + 3(220) + 6(200)}{18} = 170$$

$$c_3 = \frac{2400 + 4c_1 + c_2}{12} = \frac{2400 + 4(220) + 66.6667}{12} = 278.8889$$

The error estimates can be computed as

$$\varepsilon_{a,1} = \left|\frac{246.6667 - 220}{246.6667}\right| \times 100\% = 10.81\%$$

$$\varepsilon_{a,2} = \left|\frac{170 - 66.6667}{170}\right| \times 100\% = 60.78\%$$

$$\varepsilon_{a,3} = \left|\frac{278.8889 - 200}{278.8889}\right| \times 100\% = 28.29\%$$

The remainder of the calculation proceeds until all the errors fall below the stopping criterion of 5%. The entire computation can be summarized as

| iteration | unknown | value | $\varepsilon_a$ | maximum $\varepsilon_a$ |
|---|---|---|---|---|
| 1 | c1 | 220 | 100.00% | |
| | c2 | 66.66667 | 100.00% | |
| | c3 | 200 | 100.00% | 100.000% |
| 2 | c1 | 246.6667 | 10.81% | |
| | c2 | 170 | 60.78% | |
| | c3 | 278.8889 | 28.29% | 60.784% |
| 3 | c1 | 272.5926 | 9.51% | |
| | c2 | 200.7407 | 15.31% | |
| | c3 | 296.3889 | 5.90% | 15.314% |
| 4 | c1 | 279.9074 | 2.61% | |
| | c2 | 210.8951 | 4.81% | |
| | c3 | 307.5926 | 3.64% | 4.815% |

Thus, after 4 iterations, the maximum error is 4.79% and we arrive at the result: $c_1$ = 315.5402, $c_2$ = 219.0664 and $c_3$ = 315.6211.

**11.11** The first iteration can be implemented as

$$x_1 = \frac{27 - 2x_2 + x_3}{10} = \frac{27 - 2(0) + 0}{10} = 2.7$$

$$x_2 = \frac{-61.5 + 3x_1 - 2x_3}{-6} = \frac{-61.5 + 3(2.7) - 2(0)}{-6} = 8.9$$

$$x_3 = \frac{-21.5 - x_1 - x_2}{5} = \frac{-21.5 - (2.7) - 8.9}{5} = -6.62$$

Second iteration:

$$x_1 = \frac{27 - 2(8.9) - 6.62}{10} = 0.258$$

$$x_2 = \frac{-61.5 + 3(0.258) - 2(-6.62)}{-6} = 7.914333$$

$$x_3 = \frac{-21.5 - (0.258) - 7.914333}{5} = -5.934467$$

The error estimates can be computed as

$$\varepsilon_{a,1} = \left|\frac{0.258 - 2.7}{0.258}\right| \times 100\% = 947\%$$

$$\varepsilon_{a,2} = \left|\frac{7.914333 - 8.9}{7.914333}\right| \times 100\% = 12.45\%$$

$$\varepsilon_{a,3} = \left|\frac{-5.934467 - (-6.62)}{-5.934467}\right| \times 100\% = 11.55\%$$

The remainder of the calculation proceeds until all the errors fall below the stopping criterion of 5%. The entire computation can be summarized as

| iteration | unknown | value | $\varepsilon_a$ | maximum $\varepsilon_a$ |
|---|---|---|---|---|
| 1 | *x*1 | 2.7 | 100.00% | |
| | *x*2 | 8.9 | 100.00% | |
| | *x*3 | -6.62 | 100.00% | 100% |
| 2 | *x*1 | 0.258 | 946.51% | |
| | *x*2 | 7.914333 | 12.45% | |
| | *x*3 | -5.93447 | 11.55% | 946% |
| 3 | *x*1 | 0.523687 | 50.73% | |
| | *x*2 | 8.010001 | 1.19% | |
| | *x*3 | -6.00674 | 1.20% | 50.73% |
| 4 | *x*1 | 0.497326 | 5.30% | |
| | *x*2 | 7.999091 | 0.14% | |
| | *x*3 | -5.99928 | 0.12% | 5.30% |
| 5 | *x*1 | 0.500253 | 0.59% | |
| | *x*2 | 8.000112 | 0.01% | |
| | *x*3 | -6.00007 | 0.01% | 0.59% |

Thus, after 5 iterations, the maximum error is 0.59% and we arrive at the result: $x_1$ = 0.500253, $x_2$ = 8.000112 and $x_3$ = –6.00007.

**11.12** The equations should first be rearranged so that they are diagonally dominant,

$$6x_1 - x_2 - x_3 = 3$$
$$6x_1 + 9x_2 + x_3 = 40$$
$$-3x_1 + x_2 + 12x_3 = 50$$

Each can be solved for the unknown on the diagonal as

$$x_1 = \frac{3 + x_2 + x_3}{6}$$
$$x_2 = \frac{40 - 6x_1 - x_3}{9}$$
$$x_3 = \frac{50 + 3x_1 - x_2}{12}$$

**(a)** The first iteration can be implemented as

$$x_1 = \frac{3 + 0 + 0}{6} = 0.5$$
$$x_2 = \frac{40 - 6(0.5) - 0}{9} = 4.11111$$
$$x_3 = \frac{50 + 3(0.5) - 4.11111}{12} = 3.949074$$

Second iteration:

$$x_1 = \frac{3 + 4.11111 + 3.949074}{6} = 1.843364$$
$$x_2 = \frac{40 - 6(1.843364) - 3.949074}{9} = 2.776749$$
$$x_3 = \frac{50 + 3(1.843364) - 2.776749}{12} = 4.396112$$

The error estimates can be computed as

$$\varepsilon_{a,1} = \left|\frac{1.843364 - 0.5}{1.843364}\right| \times 100\% = 72.88\%$$
$$\varepsilon_{a,2} = \left|\frac{2.776749 - 4.11111}{2.776749}\right| \times 100\% = 48.05\%$$
$$\varepsilon_{a,3} = \left|\frac{4.396112 - 3.949074}{4.396112}\right| \times 100\% = 10.17\%$$

The remainder of the calculation proceeds until all the errors fall below the stopping criterion of 5%. The entire computation can be summarized as

| iteration | unknown | value | $\varepsilon_a$ | maximum $\varepsilon_a$ |
|---|---|---|---|---|
| 1 | $x1$ | 0.5 | 100.00% | |
| | $x2$ | 4.111111 | 100.00% | |
| | $x3$ | 3.949074 | 100.00% | 100.00% |
| 2 | $x1$ | 1.843364 | 72.88% | |
| | $x2$ | 2.776749 | 48.05% | |
| | $x3$ | 4.396112 | 10.17% | 72.88% |
| 3 | $x1$ | 1.695477 | 8.72% | |
| | $x2$ | 2.82567 | 1.73% | |

| | | | | |
|---|---|---|---|---|
| | $x3$ | 4.355063 | 0.94% | 8.72% |
| 4 | $x1$ | 1.696789 | 0.08% | |
| | $x2$ | 2.829356 | 0.13% | |
| | $x3$ | 4.355084 | 0.00% | 0.13% |

Thus, after 4 iterations, the maximum error is 0.13% and we arrive at the result: $x_1$ = 1.696789, $x_2$ = 2.829356 and $x_3$ = 4.355084.

**(b)** First iteration: To start, assume $x_1 = x_2 = x_3 = 0$

$$x_1^{new} = \frac{3+0+0}{6} = 0.5$$

Apply relaxation

$$x_1 = 0.95(0.5) + (1-0.95)0 = 0.475$$
$$x_2^{new} = \frac{40-6(0.475)-0}{9} = 4.12778$$
$$x_2 = 0.95(4.12778) + (1-0.95)0 = 3.92139$$
$$x_3^{new} = \frac{50+3(0.475)-3.92139}{12} = 3.95863$$
$$x_3 = 0.95(3.95863) + (1-0.95)0 = 3.76070$$

Note that error estimates are not made on the first iteration, because all errors will be 100%.

Second iteration:

$$x_1^{new} = \frac{3+3.92139+3.76070}{6} = 1.78035$$
$$x_1 = 0.95(1.78035) + (1-0.95)(0.475) = 1.71508$$

At this point, an error estimate can be made

$$\varepsilon_{a,1} = \left|\frac{1.71508-0.475}{1.71508}\right|100\% = 72.3\%$$

Because this error exceeds the stopping criterion, it will not be necessary to compute error estimates for the remainder of this iteration.

$$x_2^{new} = \frac{40-6(1.71508)-3.76070}{9} = 2.88320$$
$$x_2 = 0.95(2.88320) + (1-0.95)3.92139 = 2.93511$$
$$x_3^{new} = \frac{50+3(1.71508)-2.93511}{12} = 4.35084$$
$$x_3 = 0.95(4.35084) + (1-0.95)3.76070 = 4.32134$$

The computations can be continued for one more iteration. The entire calculation is summarized in the following table.

| iteration | $x_1$ | $x_{1r}$ | $\varepsilon_{a1}$ | $x_2$ | $x_{2r}$ | $\varepsilon_{a2}$ | $x_3$ | $x_{3r}$ | $\varepsilon_{a3}$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.50000 | 0.47500 | 100.0% | 4.12778 | 3.92139 | 100.0% | 3.95863 | 3.76070 | 100.0% |
| 2 | 1.78035 | 1.71508 | 72.3% | 2.88320 | 2.93511 | 33.6% | 4.35084 | 4.32134 | 13.0% |
| 3 | 1.70941 | 1.70969 | 0.3% | 2.82450 | 2.83003 | 3.7% | 4.35825 | 4.35641 | 0.8% |

After 3 iterations, the approximate errors fall below the stopping criterion with the final result: $x_1$ = 1.70969, $x_2$ = 2.82450 and $x_3$ = 4.35641. Note that the exact solution is $x_1$ = 1.69737, $x_2$ = 2.82895 and $x_3$ = 4.35526

**11.13** The equations must first be rearranged so that they are diagonally dominant

$$-8x_1 + x_2 - 2x_3 = -20$$
$$2x_1 - 6x_2 - x_3 = -38$$
$$-3x_1 - x_2 + 7x_3 = -34$$

**(a)** The first iteration can be implemented as

$$x_1 = \frac{-20 - x_2 + 2x_3}{-8} = \frac{-20 - 0 + 2(0)}{-8} = 2.5$$
$$x_2 = \frac{-38 - 2x_1 + x_3}{-6} = \frac{-38 - 2(2.5) + 0}{-6} = 7.166667$$
$$x_3 = \frac{-34 + 3x_1 + x_2}{7} = \frac{-34 + 3(2.5) + 7.166667}{7} = -2.761905$$

Second iteration:

$$x_1 = \frac{-20 - 7.166667 + 2(-2.761905)}{-8} = 4.08631$$
$$x_2 = \frac{-38 - 2x_1 + x_3}{-6} = \frac{-38 - 2(4.08631) + (-2.761905)}{-6} = 8.155754$$
$$x_3 = \frac{-34 + 3x_1 + x_2}{7} = \frac{-34 + 3(4.08631) + 8.155754}{7} = -1.94076$$

The error estimates can be computed as

$$\varepsilon_{a,1} = \left|\frac{4.08631 - 2.5}{4.08631}\right| \times 100\% = 38.82\%$$
$$\varepsilon_{a,2} = \left|\frac{8.155754 - 7.166667}{8.155754}\right| \times 100\% = 12.13\%$$
$$\varepsilon_{a,3} = \left|\frac{-1.94076 - (-2.761905)}{-1.94076}\right| \times 100\% = 42.31\%$$

The remainder of the calculation proceeds until all the errors fall below the stopping criterion of 5%. The entire computation can be summarized as

| iteration | unknown | value | $\varepsilon_a$ | maximum $\varepsilon_a$ |
|---|---|---|---|---|
| 0 | $x$1 | 0 | | |
| | $x$2 | 0 | | |

| | | | | |
|---|---|---|---|---|
| | *x*3 | 0 | | |
| 1 | *x*1 | 2.5 | 100.00% | |
| | *x*2 | 7.166667 | 100.00% | |
| | *x*3 | -2.7619 | 100.00% | 100.00% |
| 2 | *x*1 | 4.08631 | 38.82% | |
| | *x*2 | 8.155754 | 12.13% | |
| | *x*3 | -1.94076 | 42.31% | 42.31% |
| 3 | *x*1 | 4.004659 | 2.04% | |
| | *x*2 | 7.99168 | 2.05% | |
| | *x*3 | -1.99919 | 2.92% | 2.92% |

Thus, after 3 iterations, the maximum error is 2.92% and we arrive at the result: $x_1$ = 4.004659, $x_2$ = 7.99168 and $x_3$ = –1.99919.

**(b)** The same computation can be developed with relaxation where $\lambda$ = 1.2.

First iteration:

$$x_1 = \frac{-20 - x_2 + 2x_3}{-8} = \frac{-20 - 0 + 2(0)}{-8} = 2.5$$

Relaxation yields: $x_1 = 1.2(2.5) - 0.2(0) = 3$

$$x_2 = \frac{-38 - 2x_1 + x_3}{-6} = \frac{-38 - 2(3) + 0}{-6} = 7.333333$$

Relaxation yields: $x_2 = 1.2(7.333333) - 0.2(0) = 8.8$

$$x_3 = \frac{-34 + 3x_1 + x_2}{7} = \frac{-34 + 3(3) + 8.8}{7} = -2.3142857$$

Relaxation yields: $x_3 = 1.2(-2.3142857) - 0.2(0) = -2.7771429$

Second iteration:

$$x_1 = \frac{-20 - x_2 + 2x_3}{-8} = \frac{-20 - 8.8 + 2(-2.7771429)}{-8} = 4.2942857$$

Relaxation yields: $x_1 = 1.2(4.2942857) - 0.2(3) = 4.5531429$

$$x_2 = \frac{-38 - 2x_1 + x_3}{-6} = \frac{-38 - 2(4.5531429) - 2.7771429}{-6} = 8.3139048$$

Relaxation yields: $x_2 = 1.2(8.3139048) - 0.2(8.8) = 8.2166857$

$$x_3 = \frac{-34 + 3x_1 + x_2}{7} = \frac{-34 + 3(4.5531429) + 8.2166857}{7} = -1.7319837$$

Relaxation yields: $x_3 = 1.2(-1.7319837) - 0.2(-2.7771429) = -1.5229518$

The error estimates can be computed as

$$\varepsilon_{a,1} = \left|\frac{4.5531429 - 3}{4.5531429}\right| \times 100\% = 34.11\%$$

$$\varepsilon_{a,2} = \left|\frac{8.2166857 - 8.8}{8.2166857}\right| \times 100\% = 7.1\%$$

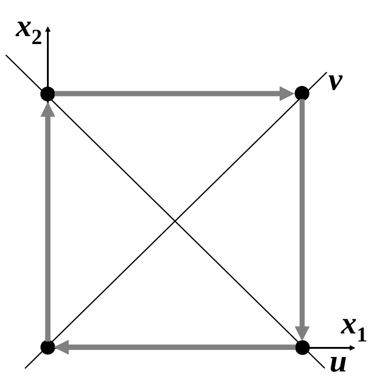
$$\varepsilon_{a,3} = \left|\frac{-1.5229518 - (-2.7771429)}{-1.5229518}\right| \times 100\% = 82.35\%$$

The remainder of the calculation proceeds until all the errors fall below the stopping criterion of 5%. The entire computation can be summarized as

| iteration | unknown | value | relaxation | $\varepsilon_a$ | maximum $\varepsilon_a$ |
|---|---|---|---|---|---|
| 1 | $x1$ | 2.5 | 3 | 100.00% | |
| | $x2$ | 7.3333333 | 8.8 | 100.00% | |
| | $x3$ | -2.314286 | -2.777143 | 100.00% | 100.000% |
| 2 | $x1$ | 4.2942857 | 4.5531429 | 34.11% | |
| | $x2$ | 8.3139048 | 8.2166857 | 7.10% | |
| | $x3$ | -1.731984 | -1.522952 | 82.35% | 82.353% |
| 3 | $x1$ | 3.9078237 | 3.7787598 | 20.49% | |
| | $x2$ | 7.8467453 | 7.7727572 | 5.71% | |
| | $x3$ | -2.12728 | -2.248146 | 32.26% | 32.257% |
| 4 | $x1$ | 4.0336312 | 4.0846055 | 7.49% | |
| | $x2$ | 8.0695595 | 8.12892 | 4.38% | |
| | $x3$ | -1.945323 | -1.884759 | 19.28% | 19.280% |
| 5 | $x1$ | 3.9873047 | 3.9678445 | 2.94% | |
| | $x2$ | 7.9700747 | 7.9383056 | 2.40% | |
| | $x3$ | -2.022594 | -2.050162 | 8.07% | 8.068% |
| 6 | $x1$ | 4.0048286 | 4.0122254 | 1.11% | |
| | $x2$ | 8.0124354 | 8.0272613 | 1.11% | |
| | $x3$ | -1.990866 | -1.979007 | 3.60% | 3.595% |

Thus, relaxation actually seems to retard convergence. After 6 iterations, the maximum error is 3.595% and we arrive at the result: $x_1 = 4.0122254$, $x_2 = 8.0272613$ and $x_3 = -1.979007$.

**11.14** As shown below, for slopes of 1 and –1 the Gauss-Seidel technique will neither converge nor diverge but will oscillate interminably.



**11.15** As ordered, none of the sets will converge. However, if Set 1 and 2 are reordered so that they are diagonally dominant, they will converge on the solution of (1, 1, 1).

Set 1:

$$
\begin{aligned}
9x + 3y + z &= 13 \\
2x + 5y - z &= 6 \\
-6x \quad + 8z &= 2
\end{aligned}
$$

Set 2:

$$
\begin{aligned}
4x + 2y - 2z &= 4 \\
x + 5y - z &= 5 \\
x + y + 6z &= 8
\end{aligned}
$$

At face value, because it is not strictly diagonally dominant, Set 2 would seem to be divergent. However, since it is very close to being diagonally dominant, a solution can be obtained.

The third set is not diagonally dominant and will diverge for most orderings. However, the following arrangement will converge albeit at a very slow rate:

Set 3:

$$
\begin{aligned}
-3x + 4y + 5z &= 6 \\
2y - z &= 1 \\
-2x + 2y - 4z &= -3
\end{aligned}
$$

**11.16** Using MATLAB:

**(a)** The results for the first system will come out as expected.

```
>> A=[1 4 9;4 9 16;9 16 25]
>> B=[14 29 50]'
>> x=A\B
x =
    1.0000
    1.0000
    1.0000

>> inv(A)
ans =
    3.8750   -5.5000    2.1250
   -5.5000    7.0000   -2.5000
    2.1250   -2.5000    0.8750

>> cond(A,inf)
ans =
  750.0000
```

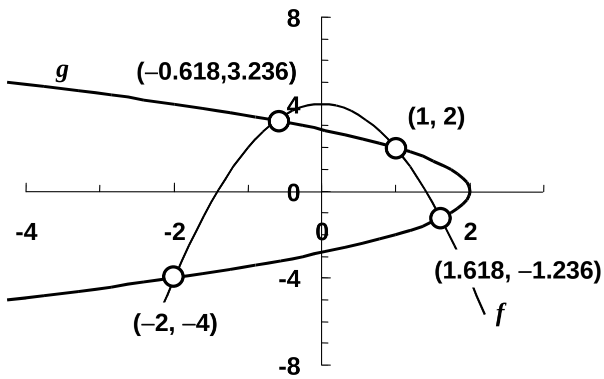
**(b)** However, for the 4×4 system, the ill-conditioned nature of the matrix yields poor results:

```
>> A=[1 4 9 16;4 9 16 25;9 16 25 36;16 25 36 49];
>> B=[30 54 86 126]';
>> x=A\B
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 3.037487e-019.

x =
    0.5496
    2.3513
   -0.3513
    1.4504

>> cond(A,inf)
Warning: Matrix is close to singular or badly scaled.
```

```
        Results may be inaccurate. RCOND = 3.037487e-019.
> In cond at 48

ans =
  3.2922e+018
```

Note that using other software such as Excel yields similar results. For example, the condition number computed with Excel is $5\times10^{17}$.

**11.17 (a)** As shown, there are 4 roots, one in each quadrant.




**(b)** It might be expected that if an initial guess was within a quadrant, the result would be the root in the quadrant. However a sample of initial guesses spanning the range yield the following roots:

| | | | | | |
|---|---|---|---|---|---|
| 6 | (-2, -4) | (-0.618,3.236) | (-0.618,3.236) | (1,2) | (-0.618,3.236) |
| 3 | (-0.618,3.236) | (-0.618,3.236) | (-0.618,3.236) | (1,2) | (-0.618,3.236) |
| 0 | (1,2) | (1.618, -1.236) | (1.618, -1.236) | (1.618, -1.236) | (1.618, -1.236) |
| -3 | (-2, -4) | (-2, -4) | (1.618, -1.236) | (1.618, -1.236) | (1.618, -1.236) |
| -6 | (-2, -4) | (-2, -4) | (-2, -4) | (1.618, -1.236) | (-2, -4) |
| | -6 | -3 | 0 | 3 | 6 |

We have highlighted the guesses that converge to the roots in their quadrants. Although some follow the pattern, others jump to roots that are far away. For example, the guess of (–6, 0) jumps to the root in the first quadrant.

This underscores the notion that root location techniques are highly sensitive to initial guesses and that open methods like the Solver can locate roots that are not in the vicinity of the initial guesses.

**11.18** Define the quantity of transistors, resistors, and computer chips as $x_1$, $x_2$ and $x_3$. The system equations can then be defined as

$$4x_1 + 3x_2 + 2x_3 = 960$$
$$x_1 + 3x_2 + x_3 = 510$$
$$2x_1 + x_2 + 3x_3 = 610$$

The solution can be implemented in Excel as shown below:

| | A | B | C | D |
|---|---|---|---|---|
| 1 | A: | | | B: |
| 2 | 4 | 3 | 2 | 960 |
| 3 | 1 | 3 | 1 | 510 |
| 4 | 2 | 1 | 3 | 610 |
| 5 | | | | |
| 6 | AI: | | | X: |
| 7 | 0.421053 | -0.36842 | -0.15789 | 120 |
| 8 | -0.05263 | 0.421053 | -0.10526 | 100 |
| 9 | -0.26316 | 0.105263 | 0.473684 | 90 |

The following view shows the formulas that are employed to determine the inverse in cells A7:C9 and the solution in cells D7:D9.

| | A | B | C | D |
|---|---|---|---|---|
| 1 | A: | | | B: |
| 2 | 4 | 3 | 2 | 960 |
| 3 | 1 | 3 | 1 | 510 |
| 4 | 2 | 1 | 3 | 610 |
| 5 | | | | |
| 6 | AI: | | | X: |
| 7 | =MINVERSE(A2:C4) | =MINVERSE(A2:C4) | =MINVERSE(A2:C4) | =MMULT(A7:C9,D2:D4) |
| 8 | =MINVERSE(A2:C4) | =MINVERSE(A2:C4) | =MINVERSE(A2:C4) | =MMULT(A7:C9,D2:D4) |
| 9 | =MINVERSE(A2:C4) | =MINVERSE(A2:C4) | =MINVERSE(A2:C4) | =MMULT(A7:C9,D2:D4) |

Here is the same solution generated in MATLAB:

```
>> A=[4 3 2;1 3 1;2 1 3];
>> B=[960 510 610]';
>> x=A\B

x =
   120
   100
    90
```

In both cases, the answer is $x_1 = 120$, $x_2 = 100$, and $x_3 = 90$

**11.19** The spectral condition number can be evaluated as

```
>> A = hilb(10);
>> N = cond(A)

N =
  1.6025e+013
```

The digits of precision that could be lost due to ill-conditioning can be calculated as

```
>> c = log10(N)

c =
   13.2048
```

Thus, about 13 digits could be suspect. A right-hand side vector can be developed corresponding to a solution of ones:

```
>> b=[sum(A(1,:)); sum(A(2,:)); sum(A(3,:)); sum(A(4,:)); sum(A(5,:));
sum(A(6,:)); sum(A(7,:)); sum(A(8,:)); sum(A(9,:)); sum(A(10,:))]
```

```
b =
    2.9290
    2.0199
    1.6032
    1.3468
    1.1682
    1.0349
    0.9307
    0.8467
    0.7773
    0.7188
```

The solution can then be generated by left division

```
>> x = A\b

x =
    1.0000
    1.0000
    1.0000
    1.0000
    0.9999
    1.0003
    0.9995
    1.0005
    0.9997
    1.0001
```

The maximum and mean errors can be computed as

```
>> e=max(abs(x-1))

e =
  5.3822e-004

>> e=mean(abs(x-1))

e =
  1.8662e-004
```

Thus, some of the results are accurate to only about 3 to 4 significant digits. Because MATLAB represents numbers to 15 significant digits, this means that about 11 to 12 digits are suspect.

**11.20** First, the Vandermonde matrix can be set up

```
>> x1 = 4;x2=2;x3=7;x4=10;x5=3;x6=5;
>> A = [x1^5 x1^4 x1^3 x1^2 x1 1;x2^5 x2^4 x2^3 x2^2 x2 1;x3^5 x3^4 x3^3 x3^2
x3 1;x4^5 x4^4 x4^3 x4^2 x4 1;x5^5 x5^4 x5^3 x5^2 x5 1;x6^5 x6^4 x6^3 x6^2 x6
1]

A =
        1024         256          64          16           4           1
          32          16           8           4           2           1
       16807        2401         343          49           7           1
      100000       10000        1000         100          10           1
         243          81          27           9           3           1
        3125         625         125          25           5           1
```

The spectral condition number can be evaluated as

```
>> N = cond(A)

N =
  1.4492e+007
```

The digits of precision that could be lost due to ill-conditioning can be calculated as

```
>> c = log10(N)

c =
    7.1611
```

Thus, about 7 digits might be suspect. A right-hand side vector can be developed corresponding to a solution of ones:

```
>> b=[sum(A(1,:));sum(A(2,:));sum(A(3,:));sum(A(4,:));sum(A(5,:)); sum(A(6,:))]

b =
        1365
          63
       19608
      111111
         364
        3906
```

The solution can then be generated by left division

```
>> format long
>> x=A\b

x =
   1.00000000000000
   0.99999999999991
   1.00000000000075
   0.99999999999703
   1.00000000000542
   0.99999999999630
```

The maximum and mean errors can be computed as

```
>> e = max(abs(x-1))

e =
    5.420774940034789e-012

>> e = mean(abs(x-1))

e =
    2.154110223528960e-012
```

Some of the results are accurate to about 12 significant digits. Because MATLAB represents numbers to about 15 significant digits, this means that about 3 digits are suspect. Thus, for this case, the condition number tends to exaggerate the impact of ill-conditioning.

**11.21**

```
>> Aug = [A eye(size(A))]
```

Here's an example session of how it can be employed.

```
>> A = rand(3)
A =
    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
    0.6068    0.7621    0.8214
>> Aug = [A eye(size(A))]
Aug =

    0.9501    0.4860    0.4565    1.0000         0         0
    0.2311    0.8913    0.0185         0    1.0000         0
    0.6068    0.7621    0.8214         0         0    1.0000
```

**11.22** The terms can be collected to give

$$\begin{bmatrix} 0 & -7 & 5 \\ 0 & 4 & 7 \\ -4 & 3 & -7 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 50 \\ -30 \\ 40 \end{Bmatrix}$$

Here is the MATLAB session:

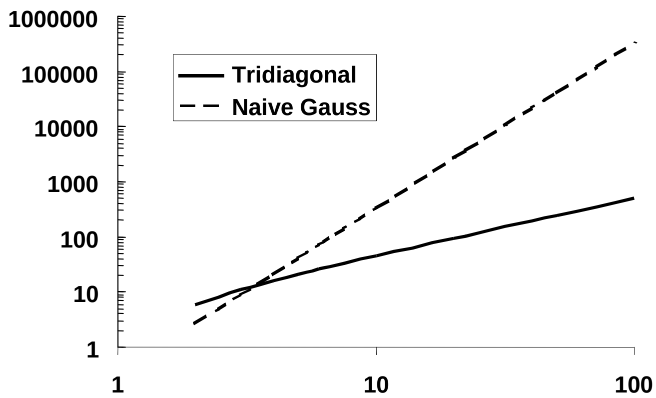
```
>> A = [0 -7 5;0 4 7;-4 3 -7];
>> b = [50;-30;40];
>> x = A\b
x =
  -15.1812
   -7.2464
   -0.1449
>> AT = A'
AT =
     0     0    -4
    -7     4     3
     5     7    -7
>> AI = inv(A)
AI =
   -0.1775   -0.1232   -0.2500
   -0.1014    0.0725         0
    0.0580    0.1014         0
```

**11.23** The flop counts for the tridiagonal algorithm in Fig. 11.2 can be determined as

```
                                                 mult/div     add/subt
    Sub Decomp(e, f, g, n)
    Dim k As Integer
    For k = 2 To n
      e(k) = e(k) / f(k - 1)                     '(n – 1)
      f(k) = f(k) - e(k) * g(k - 1)              '(n – 1)     (n – 1)
    Next k
    End Sub

    Sub Substitute(e, f, g, r, n, x)
    Dim k As Integer
    For k = 2 To n
      r(k) = r(k) - e(k) * r(k - 1)              '(n – 1)     (n – 1)
    Next k
    x(n) = r(n) / f(n)                           '   1
    For k = n - 1 To 1 Step -1
      x(k) = (r(k) - g(k) * x(k + 1)) / f(k)     '2(n – 1)    (n – 1)
    Next k
    End Sub
```

```
Sum =                                    5(n-1) + 1    (3n – 3)
```

The multiply/divides and add/subtracts can be summed to yield $8n - 7$ as opposed to $n^3/3$ for naive Gauss elimination. Therefore, a tridiagonal solver is well worth using.




**11.24** Here is a VBA macro to obtain a solution for a tridiagonal system using the Thomas algorithm. It is set up to duplicate the results of Example 11.1.

```
Option Explicit

Sub TriDiag()
Dim i As Integer, n As Integer
Dim e(10) As Double, f(10) As Double, g(10) As Double
Dim r(10) As Double, x(10) As Double
n = 4
e(2) = -1: e(3) = -1: e(4) = -1
f(1) = 2.04: f(2) = 2.04: f(3) = 2.04: f(4) = 2.04
g(1) = -1: g(2) = -1: g(3) = -1
r(1) = 40.8: r(2) = 0.8: r(3) = 0.8: r(4) = 200.8
Call Thomas(e, f, g, r, n, x)
For i = 1 To n
  MsgBox x(i)
Next i
End Sub

Sub Thomas(e, f, g, r, n, x)
Call Decomp(e, f, g, n)
Call Substitute(e, f, g, r, n, x)
End Sub

Sub Decomp(e, f, g, n)
Dim k As Integer
For k = 2 To n
  e(k) = e(k) / f(k - 1)
  f(k) = f(k) - e(k) * g(k - 1)
Next k
End Sub

Sub Substitute(e, f, g, r, n, x)
Dim k As Integer
For k = 2 To n
  r(k) = r(k) - e(k) * r(k - 1)
Next k
x(n) = r(n) / f(n)
For k = n - 1 To 1 Step -1
```

```
  x(k) = (r(k) - g(k) * x(k + 1)) / f(k)
Next k
End Sub
```

**11.25** Here is a VBA macro to obtain a solution of a symmetric system with Cholesky decomposition. It is set up to duplicate the results of Example 11.2.

```
Option Explicit

Sub TestChol()
Dim i As Integer, j As Integer
Dim n As Integer
Dim a(10, 10) As Double
n = 3
a(1, 1) = 6: a(1, 2) = 15: a(1, 3) = 55
a(2, 1) = 15: a(2, 2) = 55: a(2, 3) = 225
a(3, 1) = 55: a(3, 2) = 225: a(3, 3) = 979
Call Cholesky(a, n)
'output results to worksheet
Sheets("Sheet1").Select
Range("a3").Select
For i = 1 To n
  For j = 1 To n
    ActiveCell.Value = a(i, j)
    ActiveCell.Offset(0, 1).Select
  Next j
  ActiveCell.Offset(1, -n).Select
Next i
Range("a3").Select
End Sub

Sub Cholesky(a, n)
Dim i As Integer, j As Integer, k As Integer
Dim sum As Double
For k = 1 To n
  For i = 1 To k - 1
    sum = 0
    For j = 1 To i - 1
      sum = sum + a(i, j) * a(k, j)
    Next j
    a(k, i) = (a(k, i) - sum) / a(i, i)
  Next i
  sum = 0
  For j = 1 To k - 1
    sum = sum + a(k, j) ^ 2
  Next j
  a(k, k) = Sqr(a(k, k) - sum)
Next k
End Sub
```

**11.26** Here is a VBA macro to obtain a solution of a linear diagonally-dominant system with the Gauss-Seidel method. It is set up to duplicate the results of Example 11.3.

```
Option Explicit

Sub Gausseid()
Dim n As Integer, imax As Integer, i As Integer
Dim a(3, 3) As Double, b(3) As Double, x(3) As Double
Dim es As Double, lambda As Double
n = 3
a(1, 1) = 3: a(1, 2) = -0.1: a(1, 3) = -0.2
```

```
a(2, 1) = 0.1: a(2, 2) = 7: a(2, 3) = -0.3
a(3, 1) = 0.3: a(3, 2) = -0.2: a(3, 3) = 10
b(1) = 7.85: b(2) = -19.3: b(3) = 71.4
es = 0.1
imax = 20
lambda = 1#
Call Gseid(a, b, n, x, imax, es, lambda)
For i = 1 To n
  MsgBox x(i)
Next i
End Sub

Sub Gseid(a, b, n, x, imax, es, lambda)
Dim i As Integer, j As Integer, iter As Integer, sentinel As Integer
Dim dummy As Double, sum As Double, ea As Double, old As Double
For i = 1 To n
  dummy = a(i, i)
  For j = 1 To n
    a(i, j) = a(i, j) / dummy
  Next j
  b(i) = b(i) / dummy
Next i
For i = 1 To n
  sum = b(i)
  For j = 1 To n
    If i <> j Then sum = sum - a(i, j) * x(j)
  Next j
  x(i) = sum
Next i
iter = 1
Do
  sentinel = 1
  For i = 1 To n
    old = x(i)
    sum = b(i)
    For j = 1 To n
      If i <> j Then sum = sum - a(i, j) * x(j)
    Next j
    x(i) = lambda * sum + (1# - lambda) * old
    If sentinel = 1 And x(i) <> 0 Then
      ea = Abs((x(i) - old) / x(i)) * 100
      If ea > es Then sentinel = 0
    End If
  Next i
  iter = iter + 1
  If sentinel = 1 Or iter >= imax Then Exit Do
Loop
End Sub
```

**11.27** We can substitute centered finite divided differences into this equation to give

$$0 = D\frac{c_{i-1} - 2c_i + c_{i+1}}{\Delta x^2} - U\frac{c_{i+1} - c_{i-1}}{2\Delta x} - kc_i$$

Collecting terms yields

$$-\left(\frac{D}{\Delta x^2} + \frac{U}{2\Delta x}\right)c_{i-1} + \left(2\frac{D}{\Delta x^2} + k\right)c_i - \left(\frac{D}{\Delta x^2} - \frac{U}{2\Delta x}\right)c_{i+1} = 0$$

Using a value of Δx = 2 along with the other parameters yields,

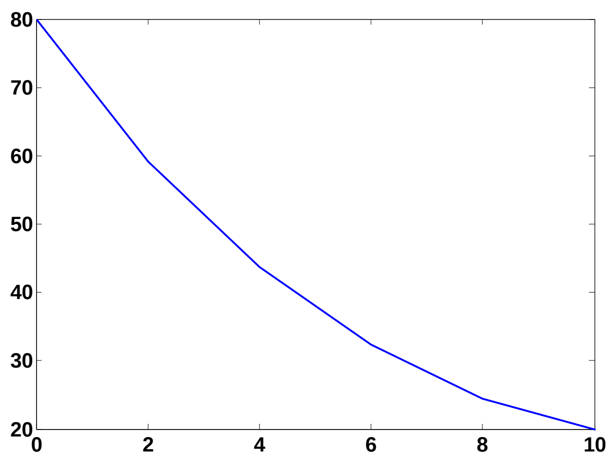
$$-0.75c_{i-1} + 1.2c_i - 0.25c_{i+1} = 0$$

This equation can then be written for the 4 interior nodes and the result expressed in matrix form as

$$\begin{bmatrix} 1.2 & -0.25 & 0 & 0 \\ -0.75 & 1.2 & -0.25 & 0 \\ 0 & -0.75 & 1.2 & -0.25 \\ 0 & 0 & -0.75 & 1.2 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{Bmatrix} = \begin{Bmatrix} 60 \\ 0 \\ 0 \\ 5 \end{Bmatrix}$$

The solution can then be generated the solution with a tool like MATLAB

```
>> A=[1.2 -0.25 0 0
-0.75 1.2 -0.25 0
0 -0.75 1.2 -0.25
0 0 -0.75 1.2];
>> b=[60;0;0;5];
>> c=A\b;
>> c'
ans =
   80.0000   59.1013   43.6860   32.3892   24.4099   20.0000
>> x=[0:2:10]';
>> c=[80; c; 20];
>> plot(x,c)
```




**11.28**

```
function x=pentasol(A,b)
% pentasol: pentadiagonal system solver banded system
%   x=pentasol(A,b):
%      Solve a pentadiagonal system Ax=b
% input:
%   A = pentadiagonal matrix
%   b = right hand side vector
% output:
%   x = solution vector
% Error checks
[m,n]=size(A);
if m~=n,error('Matrix must be square');end
if length(b)~=m,error('Matrix and vector must have the same number of
rows');end
```

```
x=zeros(n,1);
% Extract bands
d=[0;0;diag(A,-2)];
e=[0;diag(A,-1)];
f=diag(A);
g=diag(A,1);
h=diag(A,2);
delta=zeros(n,1);
epsilon=zeros(n-1,1);
gamma=zeros(n-2,1);
alpha=zeros(n,1);
c=zeros(n,1);
z=zeros(n,1);

% Decomposition
delta(1)=f(1);
epsilon(1)=g(1)/delta(1);
gamma(1)=h(1)/delta(1);
alpha(2)=e(2);
delta(2)=f(2)-alpha(2)*epsilon(1);
epsilon(2)=(g(2)-alpha(2)*gamma(1))/delta(2);
gamma(2)=h(2)/delta(2);
for k=3:n-2
  alpha(k)=e(k)-d(k)*epsilon(k-2);
  delta(k)=f(k)-d(k)*gamma(k-2)-alpha(k)*epsilon(k-1);
  epsilon(k)=(g(k)-alpha(k)*gamma(k-1))/delta(k);
  gamma(k)=h(k)/delta(k);
end
alpha(n-1)=e(n-1)-d(n-1)*epsilon(n-3);
delta(n-1)=f(n-1)-d(n-1)*gamma(n-3)-alpha(n-1)*epsilon(n-2);
epsilon(n-1)=(g(n-1)-alpha(n-1)*gamma(n-2))/delta(n-1);
alpha(n)=e(n)-d(n)*epsilon(n-2);
delta(n)=f(n)-d(n)*gamma(n-2)-alpha(n)*epsilon(n-1);
% Forward substitution
c(1)=b(1)/delta(1);
c(2)=(b(2)-alpha(2)*c(1))/delta(2);
for k=3:n
  c(k)=(b(k)-d(k)*c(k-2)-alpha(k)*c(k-1))/delta(k);
end
% Back substitution
x(n)=c(n);
x(n-1)=c(n-1)-epsilon(n-1)*x(n);
for k=n-2:-1:1
  x(k)=c(k)-epsilon(k)*x(k+1)-gamma(k)*x(k+2);
end
```

Test of function:

```
>> A=[8 -2 -1 0 0
-2 9 -4 -1 0
-1 3 7 -1 -2
0 -4 -2 12 -5
0 0 -7 -3 15];
>> b=[5 2 1 1 5]';
>> x=pentasol(A,b)
x =
    0.7993
    0.5721
    0.2503
    0.5491
    0.5599
```