Solutions Manual

to accompany

Applied Numerical Methods

With MATLAB for Engineers and Scientists

Steven C. Chapra
Tufts University

Mc Graw Hill Higher Education

Boston Burr Ridge, IL Dubuque, IA Madison, WI New York San Francisco St. Louis
Bangkok Bogotá Caracas Kuala Lumpur Lisbon London Madrid Mexico City
Milan Montreal New Delhi Santiago Seoul Singapore Sydney Taipei Toronto

CHAPTER 1

1.1 You are given the following differential equation with the initial condition, $v(t = 0) = 0$,

$$\frac{dv}{dt} = g - \frac{c_d}{m}v^2$$

Multiply both sides by m/c_d

$$\frac{m}{c_d}\frac{dv}{dt} = \frac{m}{c_d}g - v^2$$

Define $a = \sqrt{mg / c_d}$

$$\frac{m}{c_d}\frac{dv}{dt} = a^2 - v^2$$

Integrate by separation of variables,

$$\int \frac{dv}{a^2 - v^2} = \int \frac{c_d}{m}dt$$

A table of integrals can be consulted to find that

$$\int \frac{dx}{a^2 - x^2} = \frac{1}{a}\tanh^{-1}\frac{x}{a}$$

Therefore, the integration yields

$$\frac{1}{a}\tanh^{-1}\frac{v}{a} = \frac{c_d}{m}t + C$$

If $v = 0$ at $t = 0$, then because $\tanh^{-1}(0) = 0$, the constant of integration $C = 0$ and the solution is

$$\frac{1}{a}\tanh^{-1}\frac{v}{a} = \frac{c_d}{m}t$$

This result can then be rearranged to yield

$$v = \sqrt{\frac{gm}{c_d}}\tanh\left(\sqrt{\frac{gc_d}{m}}t\right)$$

1.2 This is a transient computation. For the period from ending June 1:

Balance = Previous Balance + Deposits – Withdrawals

Balance = 1512.33 + 220.13 – 327.26 = 1405.20

The balances for the remainder of the periods can be computed in a similar fashion as tabulated below:

Date	Deposit	Withdrawal	Balance
1-May			$ 1512.33
	$ 220.13	$ 327.26	
1-Jun			$ 1405.20
	$ 216.80	$ 378.61	
1-Jul			$ 1243.39
	$ 350.25	$ 106.80	
1-Aug			$ 1586.84
	$ 127.31	$ 450.61	
1-Sep			$ 1363.54

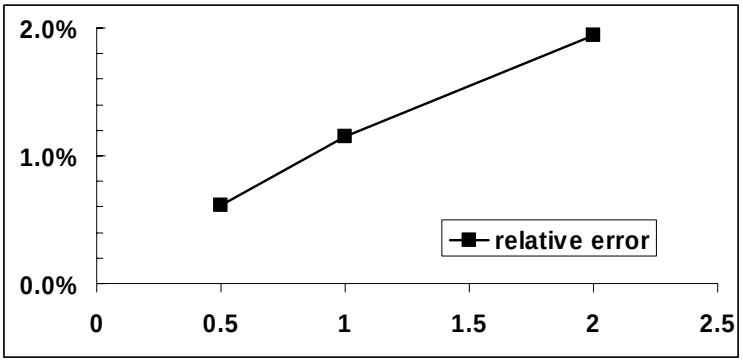
1.3 At t = 12 s, the analytical solution is 50.6175 (Example 1.1). The numerical results are:

step	v(12)	absolute relative error
2	51.6008	1.94%
1	51.2008	1.15%
0.5	50.9259	0.61%

where the relative error is calculated with

$$\text{absolute relative error} = \left|\frac{\text{analytical} - \text{numerical}}{\text{analytical}}\right| \times 100\%$$

The error versus step size can be plotted as



Thus, halving the step size approximately halves the error.

1.4 (a) The force balance is

$$\frac{dv}{dt} = g - \frac{c'}{m}v$$

Applying Laplace transforms,

$$sV - v(0) = \frac{g}{s} - \frac{c'}{m}V$$

Solve for

$$V = \frac{g}{s(s + c'/m)} + \frac{v(0)}{s + c'/m} \tag{1}$$

The first term to the right of the equal sign can be evaluated by a partial fraction expansion,

$$\frac{g}{s(s + c'/m)} = \frac{A}{s} + \frac{B}{s + c'/m} \tag{2}$$

$$\frac{g}{s(s + c'/m)} = \frac{A(s + c'/m) + Bs}{s(s + c'/m)}$$

Equating like terms in the numerators yields

$$A + B = 0$$

$$g = \frac{c'}{m}A$$

Therefore,

$$A = \frac{mg}{c'} \qquad B = -\frac{mg}{c'}$$

These results can be substituted into Eq. (2), and the result can be substituted back into Eq. (1) to give

$$V = \frac{mg/c'}{s} - \frac{mg/c'}{s + c'/m} + \frac{v(0)}{s + c'/m}$$

Applying inverse Laplace transforms yields

$$v = \frac{mg}{c'} - \frac{mg}{c'}e^{-(c'/m)t} + v(0)e^{-(c'/m)t}$$

or

$$v = v(0)e^{-(c'/m)t} + \frac{mg}{c'}\left(1 - e^{-(c'/m)t}\right)$$

where the first term to the right of the equal sign is the general solution and the second is the particular solution. For our case, $v(0) = 0$, so the final solution is

$$\boxed{v = \frac{mg}{c'}\left(1 - e^{-(c'/m)t}\right)}$$

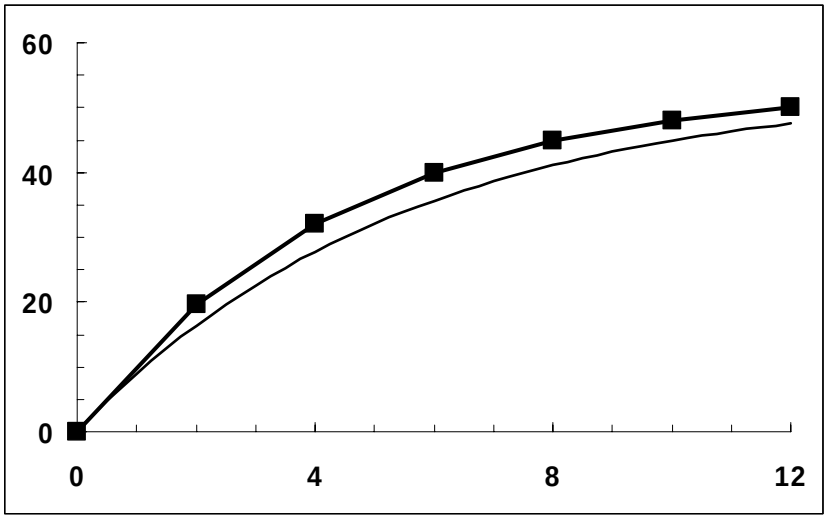
(b) The numerical solution can be implemented as

$$v(2) = 0 + \left[9.81 - \frac{12.5}{68.1}(0)\right]2 = 19.62$$

$$v(4) = 19.62 + \left[9.81 - \frac{12.5}{68.1}(19.62)\right]2 = 6.2087$$

The computation can be continued and the results summarized and plotted as:

t	*v*	*dv/dt*
0	0	9.81
2	19.6200	6.2087
4	32.0374	3.9294
6	39.8962	2.4869
8	44.8700	1.5739
10	48.0179	0.9961
12	50.0102	0.6304



Note that the analytical solution is included on the plot for comparison.

1.5 (a) The first two steps are

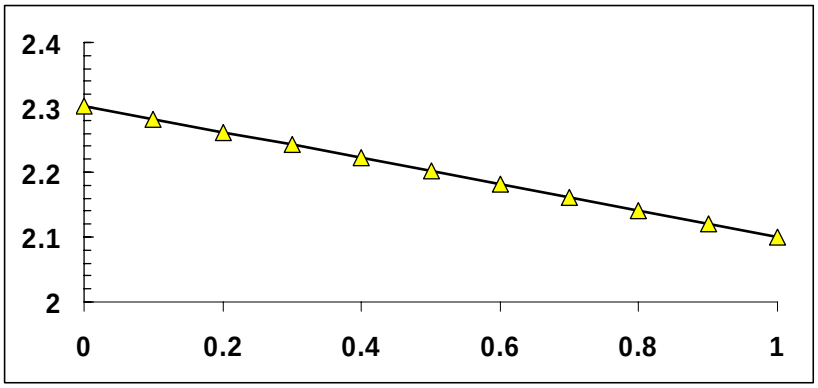
$$c(0.1) = 10 - 0.2(10)0.1 = 9.8 \text{ Bq/L}$$

$$c(0.2) = 9.8 - 0.2(9.8)0.1 = 9.604 \text{ Bq/L}$$

The process can be continued to yield

t	*c*	*dc/dt*
0	10.0000	-2.0000
0.1	9.8000	-1.9600
0.2	9.6040	-1.9208
0.3	9.4119	-1.8824
0.4	9.2237	-1.8447
0.5	9.0392	-1.8078
0.6	8.8584	-1.7717
0.7	8.6813	-1.7363
0.8	8.5076	-1.7015
0.9	8.3375	-1.6675
1	8.1707	-1.6341

(b) The results when plotted on a semi-log plot yields a straight line



The slope of this line can be estimated as

$$\frac{\ln(8.1707) - \ln(10)}{1} = -0.20203$$

Thus, the slope is approximately equal to the negative of the decay rate.

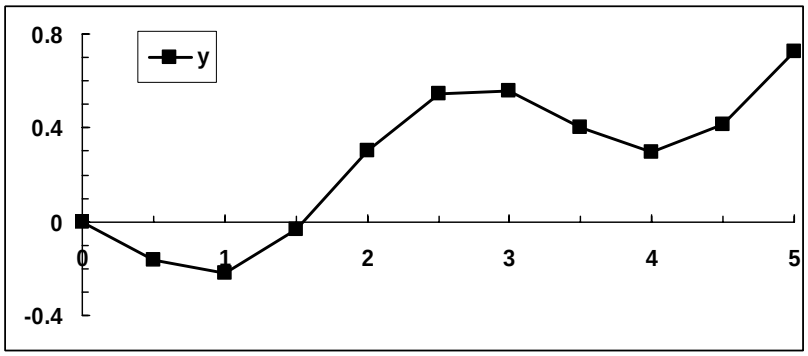
1.6 The first two steps yield

$$y(0.5) = 0 + \left[3\frac{400}{1200}\sin^2(0) - \frac{400}{1200}\right]0.5 = 0 + \left[0 - 0.33333\right]0.5 = -0.16667$$

$$y(1) = -0.16667 + \left[\sin^2(0.5) - 0.333333\right]0.5 = -0.21841$$

The process can be continued to give

t	y
0	0
0.5	-0.16667
1	-0.21841
1.5	-0.03104
2	0.299793
2.5	0.546537
3	0.558955
3.5	0.402245
4	0.297103
4.5	0.416811
5	0.727927



1.7 $v(t) = \frac{gm}{c}(1 - e^{-\left(\frac{c}{m}\right)t})$

jumper #1: $v(t) = \frac{9.8(68.1)}{12.5}(1 - e^{-\left(\frac{12.5}{68.1}\right)10}) = 44.87 \ \ m/s$

jumper #2: $44.87 = \frac{9.8(75)}{14}(1 - e^{-\left(\frac{14}{75}\right)t})$

$44.87 = 52.5 - 52.5e^{-0.18666\,t}$

$0.14533 = e^{-0.18666\,t}$

$\ln 0.14533 = \ln e^{-0.18666\,t}$

***t* = 10.33 sec**

1.8 $Q_{in} = Q_{out}$

$Q_1 = Q_2 + Q_3$

$30 = 20 + vA_3$

$10 = 5\ A_3$

$A_3 = 2\ \text{m}^2$

1.9 $\sum M_{in} - \sum M_{out} = 0$

$[1000 + 1200 + \text{MP} + 50] - [400 + 200 + 1400 + 200 + 350] = 0$

Metabolic production = 300 grams

1.10 $\sum \%\ \text{body weight} = 60$

$4.5 + 4.5 + 12 + 4.5 + 1.5 + IW = 60$

% Intracellular water body weight = 33 %

$4.5 + 4.5 + 12 + 4.5 + 1.5 + IW = 60$

$\sum \%\ \text{body water} = 100$

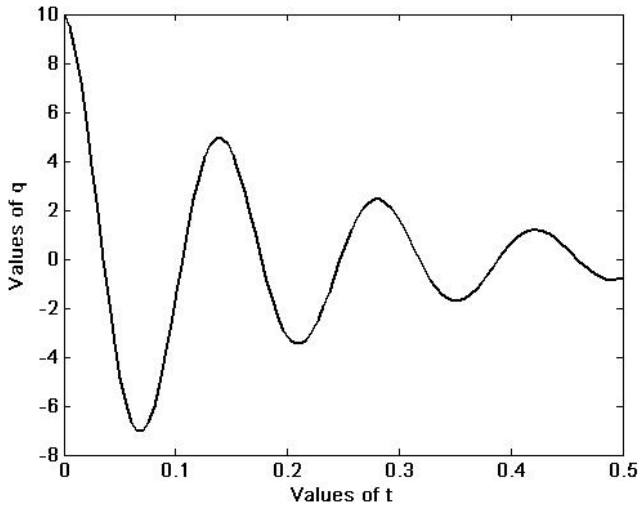
$7.5 + 7.5 + 20 + 7.5 + 55 + TW = 100$

% Transcellular water of body water = 2.5 %

CHAPTER 2

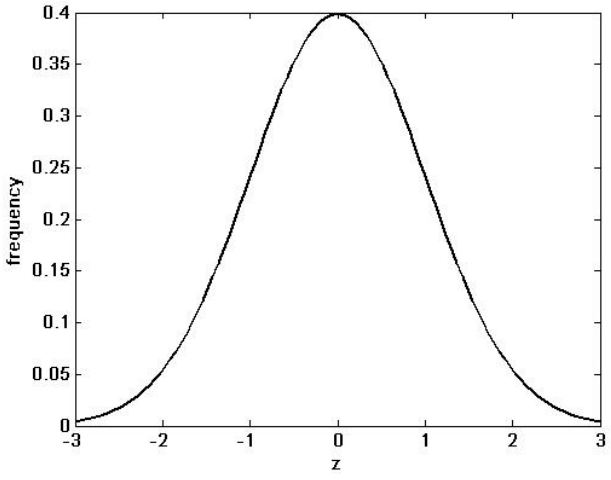
2.1

```
>> q0 = 10;R = 50;L = 5;C = 1e-4;
>> t = linspace(0,.5);
>> q = q0*exp(-R*t/(2*L)).*cos(sqrt(1/(L*C)-(R/(2*L))^2)*t);
>> plot(t,q)
```



2.2

```
>> z = linspace(-3,3);
>> f = 1/sqrt(2*pi)*exp(-z.^2/2);
>> plot(z,f)
>> xlabel('z')
>> ylabel('frequency')
```



2.3 (a)

```
>> t = linspace(5,30,6)
```

```
t =
     5    10    15    20    25    30
```

(b)

```
>> x = linspace(-3,3,7)

x =
    -3    -2    -1     0     1     2     3
```

2.4 (a)

```
>> v = -2:.75:1

v =
   -2.0000   -1.2500   -0.5000    0.2500    1.0000
```

(b)

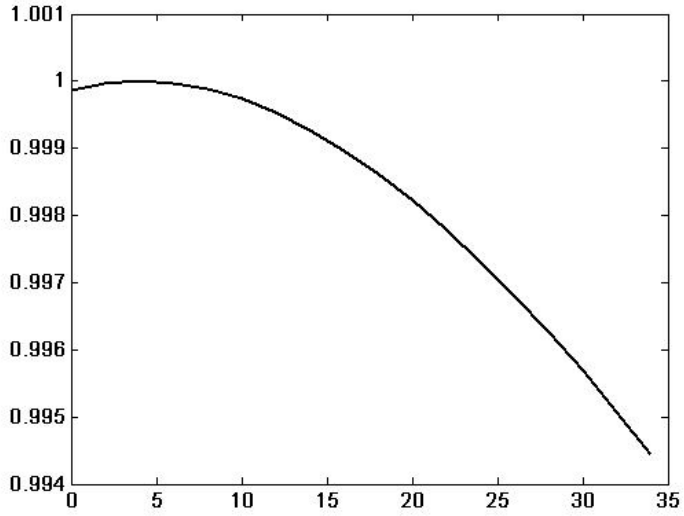
```
>> r = 6:-1:0

r =
     6     5     4     3     2     1     0
```

2.5

```
>> F = [10 12 15 9 12 16];
>> x = [0.013 0.020 0.009 0.010 0.012 0.010];
>> k = F./x

k =
  1.0e+003 *
    0.7692    0.6000    1.6667    0.9000    1.0000    1.6000

>> U = .5*k.*x.^2

U =
    0.0650    0.1200    0.0675    0.0450    0.0720    0.0800

>> max(U)

ans =
    0.1200
```

2.6

```
>> TF = 32:3.6:93.2;
>> TC = 5/9*(TF-32);
>> rho = 5.5289e-8*TC.^3-8.5016e-6*TC.^2+6.5622e-5*TC+0.99987;
>> plot(TC,rho)
```

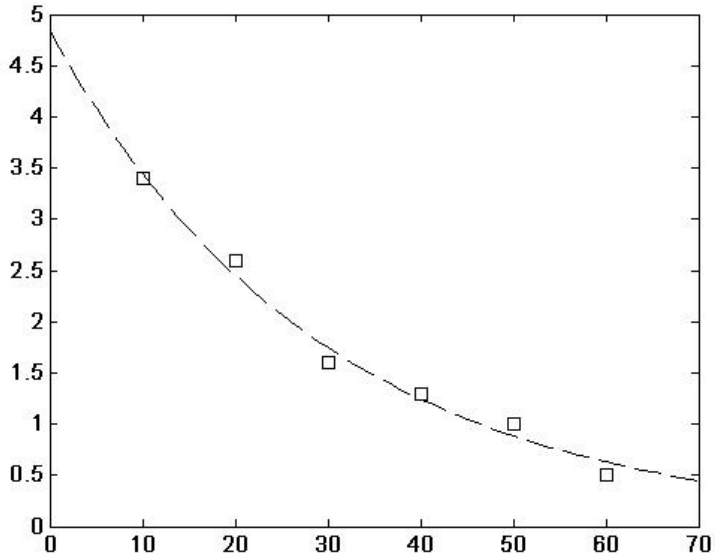


2.7

```
>> A = [.035 .0001 10 2;
.02 .0002 8 1;
.015 .001 20 1.5;
.03 .0007 24 3;
.022 .0003 15 2.5]

A =
    0.0350    0.0001   10.0000    2.0000
    0.0200    0.0002    8.0000    1.0000
    0.0150    0.0010   20.0000    1.5000
    0.0300    0.0007   24.0000    3.0000
    0.0220    0.0003   15.0000    2.5000

>> U = sqrt(A(:,2))./A(:,1).*(A(:,3).*A(:,4)./(A(:,3)+2*A(:,4))).^(2/3)

U =
    0.3624
    0.6094
    2.5167
    1.5809
    1.1971
```

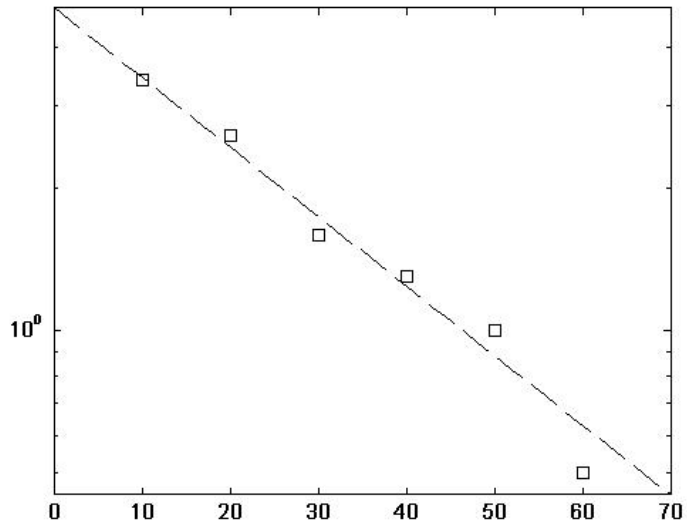
2.8

```
>> t = 10:10:60;
>> c = [3.4 2.6 1.6 1.3 1.0 0.5];
>> tf = 0:70;
>> cf = 4.84*exp(-0.034*tf);
>> plot(t,c,'s',tf,cf,'--')
```



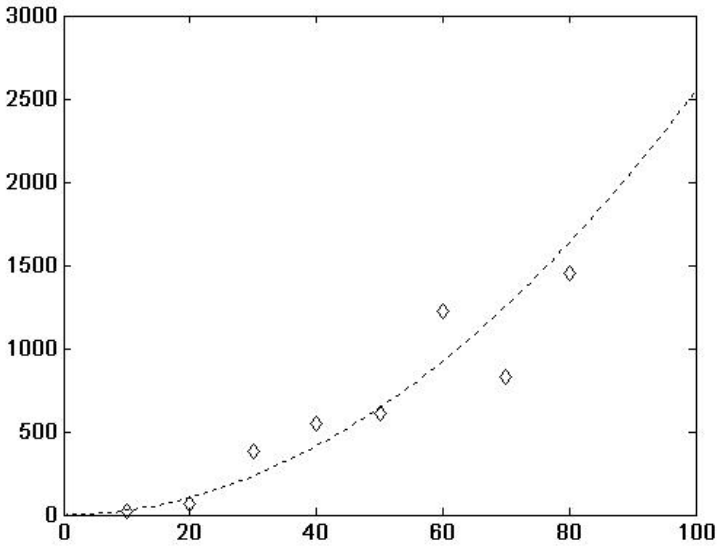
2.9

```
>> t = 10:10:60;
>> c = [3.4 2.6 1.6 1.3 1.0 0.5];
>> tf = 0:70;
>> cf = 4.84*exp(-0.034*tf);
>> semilogy(t,c,'s',tf,cf,'--')
```



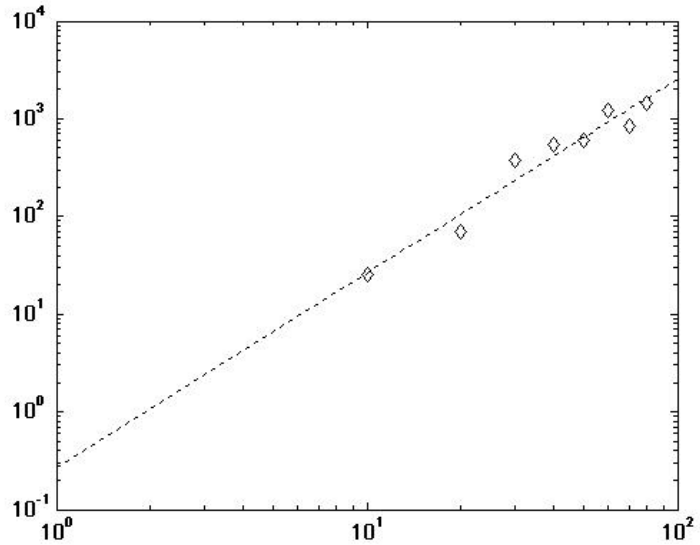
2.10

```
>> v = 10:10:80;
>> F = [25 70 380 550 610 1220 830 1450];
>> vf = 0:100;
>> Ff = 0.2741*vf.^1.9842;
>> plot(v,F,'d',vf,Ff,':')
```



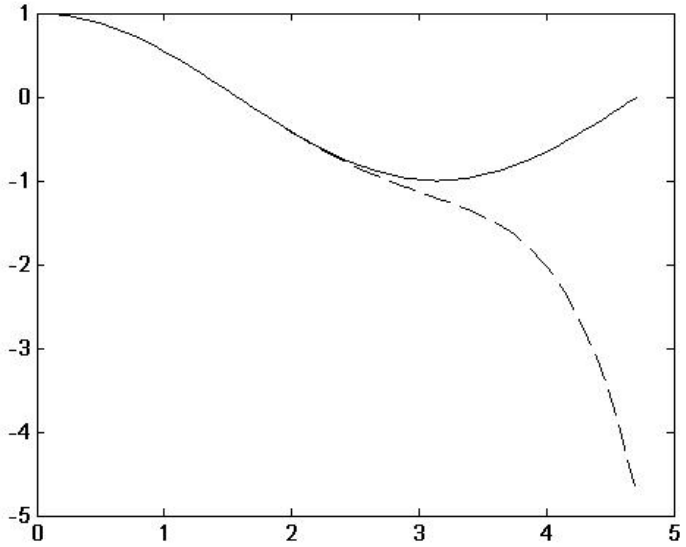
2.11

```
>> v = 10:10:80;
>> F = [25 70 380 550 610 1220 830 1450];
>> vf = 0:100;
>> Ff = 0.2741*vf.^1.9842;
>> loglog(v,F,'d',vf,Ff,':')
```



2.12

```
>> x = linspace(0,3*pi/2);
>> c = cos(x);
>> cf = 1-x.^2/2+x.^4/factorial(4)-x.^6/factorial(6);
>> plot(x,c,x,cf,'--')
```


1
0
-1
-2
-3
-4
-5
0
1
2
3
4
5

CHAPTER 3

3.1 The M-file can be written as

```
function sincomp(x,n)
i = 1;
tru = sin(x);
ser = 0;
fprintf('\n');
fprintf('order  true value     approximation     error\n');
while (1)
  if i > n, break, end
  ser = ser + (-1)^(i - 1) * x^(2*i-1) / factorial(2*i-1);
  er = (tru - ser) / tru * 100;
  fprintf('%3d  %14.10f  %14.10f  %12.8f\n',i,tru,ser,er);
  i = i + 1;
end
```

This function can be used to evaluate the test case,

```
>> sincomp(1.5,8)

order  true value     approximation     error
  1    0.9974949866    1.5000000000  -50.37669564
  2    0.9974949866    0.9375000000    6.01456523
  3    0.9974949866    1.0007812500   -0.32945162
  4    0.9974949866    0.9973911830    0.01040643
  5    0.9974949866    0.9974971226   -0.00021414
  6    0.9974949866    0.9974949557    0.00000310
  7    0.9974949866    0.9974949869   -0.00000003
  8    0.9974949866    0.9974949866    0.00000000
```

3.2 The M-file can be written as

```
function futureworth(P, i, n)
nn = 0:n;
F = P*(1+i).^nn;
y = [nn;F];
fprintf('\n  year   future worth\n');
fprintf('%5d %14.2f\n',y);
```

This function can be used to evaluate the test case,

```
>> futureworth(100000,0.08,8)

  year   future worth
    0      100000.00
    1      108000.00
    2      116640.00
    3      125971.20
    4      136048.90
    5      146932.81
    6      158687.43
    7      171382.43
    8      185093.02
```

3.3 The M-file can be written as

```
function annualpayment(P, i, n)
nn = 1:n;
A = P*i*(1+i).^nn./((1+i).^nn-1);
y = [nn;A];
fprintf('\n  year   annualpayment\n');
fprintf('%5d %14.2f\n',y);
```

This function can be used to evaluate the test case,

```
>> annualpayment(35000,.076,5)

  year   annualpayment
    1       37660.00
    2       19519.34
    3       13483.26
    4       10473.30
    5        8673.76
```

3.4 The M-file can be written as

```
function Tavg = avgtemp(Tmean, Tpeak, tstart, tend)
omega = 2*pi/365;
t = tstart:tend;
Te = Tmean + (Tpeak-Tmean)*cos(omega*(t-205));
Tavg = mean(Te);
```

This function can be used to evaluate the test cases,

```
>> avgtemp(5.2,22.1,0,59)

ans =
  -10.8418

>> avgtemp(23.1,33.6,180,242)

ans =
   33.0398
```

3.5 The M-file can be written as

```
function vol = tankvol(R, d)
if d < R
  vol = pi * d ^ 3 / 3;
elseif d <= 3 * R
  v1 = pi * R ^ 3 / 3;
  v2 = pi * R ^ 2 * (d - R);
  vol = v1 + v2;
else
  error('overtop')
end
```

This function can be used to evaluate the test cases,

```
>> tankvol(1,0.5)
ans =
    0.1309

>> tankvol(1,1.2)
ans =
    1.6755

>> tankvol(1,3.0)
ans =
    7.3304

>> tankvol(1,3.1)
??? Error using ==> tankvol
overtop
```

3.6 The M-file can be written as

```
function [r, th] = polar(x, y)
r = sqrt(x .^ 2 + y .^ 2);
if x < 0
  if y > 0
    th = atan(y / x) + pi;
  elseif y < 0
    th = atan(y / x) - pi;
  else
    th = pi;
  end
else
  if y > 0
    th = pi / 2;
  elseif y < 0
    th = -pi / 2;
  else
    th = 0;
  end
end
th = th * 180 / pi;
```

This function can be used to evaluate the test cases. For example, for the first case,

```
>> [r,th]=polar(1,1)

r =
    1.4142

th =
    90
```

The remaining cases are

x	y	r	θ
1	1	1.4142	90
1	–1	1.4142	–90
1	0	1.0000	0
–1	1	1.4142	135
–1	–1	1.4142	–135
–1	0	1.0000	180
0	1	1.0000	90
0	–1	1.0000	–90
0	0	0.0000	0

3.7 The M-file can be written as

```
function polar2(x, y)
r = sqrt(x .^ 2 + y .^ 2);
n = length(x);
for i = 1:n
  if x(i) < 0
    if y(i) > 0
      th(i) = atan(y(i) / x(i)) + pi;
    elseif y(i) < 0
      th(i) = atan(y(i) / x(i)) - pi;
    else
      th(i) = pi;
    end
  else
    if y(i) > 0
      th(i) = pi / 2;
    elseif y(i) < 0
      th(i) = -pi / 2;
    else
      th(i) = 0;
    end
  end
  th(i) = th(i) * 180 / pi;
end
ou = [x;y;r;th];
fprintf('\n     x        y       radius     angle\n');
fprintf('%8.2f %8.2f %10.4f %10.4f\n',ou);
```

This function can be used to evaluate the test cases and display the results in tabular form,

```
>> polar2(x,y)

     x        y       radius     angle
    1.00     1.00     1.4142    90.0000
    1.00    -1.00     1.4142   -90.0000
    1.00     0.00     1.0000     0.0000
   -1.00     1.00     1.4142   135.0000
   -1.00    -1.00     1.4142  -135.0000
   -1.00     0.00     1.0000   180.0000
    0.00     1.00     1.0000    90.0000
    0.00    -1.00     1.0000   -90.0000
    0.00     0.00     0.0000     0.0000
```

3.8 The M-file can be written as

```
function grade = lettergrade(score)
if score >= 90
  grade = 'A';
elseif score >= 80
  grade = 'B';
elseif score >= 70
  grade = 'C';
elseif score >= 60
  grade = 'D';
else
  grade = 'F';
end
```

This function can be tested with a few cases,

```
>> lettergrade(95)
ans =
A

>> lettergrade(45)
ans =
F

>> lettergrade(80)
ans =
B
```

3.9 The M-file can be written as

```
function Manning(A)
A(:,5) = sqrt(A(:,2))./A(:,1).*(A(:,3).*A(:,4)./(A(:,3)+2*A(:,4))).^(2/3);
fprintf('\n    n         S          B          H          U\n');
fprintf('%8.3f %8.4f %10.2f %10.2f %10.4f\n',A');
```

This function can be run to create the table,

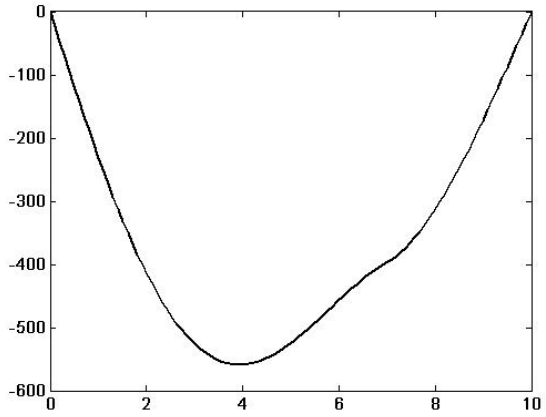
```
>> Manning(A)

    n         S          B          H          U
   0.035   0.0001      10.00       2.00     0.3624
   0.020   0.0002       8.00       1.00     0.6094
   0.015   0.0010      20.00       1.50     2.5167
   0.030   0.0007      24.00       3.00     1.5809
   0.022   0.0003      15.00       2.50     1.1971
```

3.10 The M-file can be written as

```
function beam(x)
xx = linspace(0,x);
n=length(xx);
for i=1:n
  uy(i) = -5/6.*(sing(xx(i),0,4)-sing(xx(i),5,4));
  uy(i) = uy(i) + 15/6.*sing(xx(i),8,3) + 75*sing(xx(i),7,2);
  uy(i) = uy(i) + 57/6.*xx(i)^3 - 238.25.*xx(i);
end
plot(xx,uy)

function s = sing(xxx,a,n)
if xxx > a
  s = (xxx - a).^n;
else
  s=0;
end
```

This function can be run to create the plot,

```
>> beam(10)
```

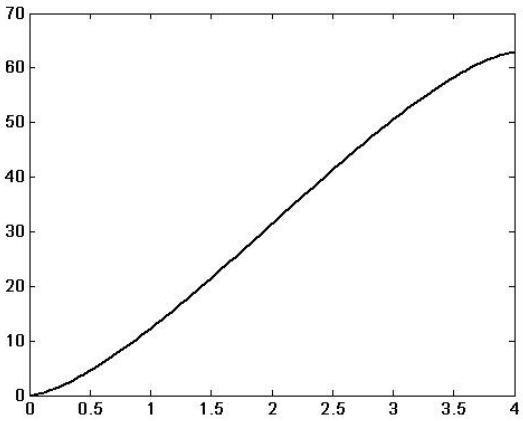


3.11 The M-file can be written as

```
function cylinder(r, L)
h = linspace(0,2*r);
V = (r^2*acos((r-h)./r)-(r-h).*sqrt(2*r*h-h.^2))*L;
plot(h, V)
```

This function can be run to the plot,

```
>> cylinder(2,5)
```


70
60
50
40
30
20
10
0
0
0.5
1
1.5
2
2.5
3
3.5
4

CHAPTER 4

4.1 The true value can be computed as

$$f'(1.22) = \frac{6(0.577)}{(1 - 3 \times 0.577^2)^2} = 2{,}352{,}911$$

Using 3-digits with chopping

$$6x = 6(0.577) = 3.462 \xrightarrow{\text{chopping}} 3.46$$
$$x = 0.577$$
$$x^2 = 0.332929 \xrightarrow{\text{chopping}} 0.332$$
$$3x^2 = 0.996$$
$$1 - 3x^2 = 0.004$$

$$f'(0.577) = \frac{3.46}{(1 - 0.996)^2} = \frac{3.46}{0.004^2} = 216{,}250$$

This represents a percent relative error of

$$\varepsilon_t = \left| \frac{2{,}352{,}911 - 216{,}250}{2{,}352{,}911} \right| = 90.8\%$$

Using 4-digits with chopping

$$6x = 6(0.577) = 3.462 \xrightarrow{\text{chopping}} 3.462$$
$$x = 0.577$$
$$x^2 = 0.332929 \xrightarrow{\text{chopping}} 0.3329$$
$$3x^2 = 0.9987$$
$$1 - 3x^2 = 0.0013$$

$$f'(0.577) = \frac{3.462}{(1 - 0.9987)^2} = \frac{3.462}{0.0013^2} = 2{,}048{,}521$$

This represents a percent relative error of

$$\varepsilon_t = \left| \frac{2{,}352{,}911 - 2{,}048{,}521}{2{,}352{,}911} \right| = 12.9\%$$

Although using more significant digits improves the estimate, the error is still considerable. The problem stems primarily from the fact that we are subtracting two nearly equal numbers in the denominator. Such subtractive cancellation is worsened by the fact that the denominator is squared.

4.2 First, the correct result can be calculated as

$$y = 1.37^3 - 7(1.37)^2 + 8(1.37) - 0.35 = 0.043053$$

(a) Using 3-digits with chopping

1.37^3	$\rightarrow$	2.571353	$\rightarrow$	2.57
$-7(1.37)^2$	$\rightarrow$	$-7(1.87)$	$\rightarrow$	–13.0
$8(1.37)$	$\rightarrow$	10.96	$\rightarrow$	10.9
				– 0.35
				–0.12

This represents an error of

$$\varepsilon_t = \left|\frac{0.043053 - 0.12}{0.043053}\right| = 178.7\%$$

(b) Using 3-digits with chopping

$$y = ((1.37 - 7)1.37 + 8)1.37 - 0.35$$

$$y = (-5.63 \times 1.37 + 8)1.37 - 0.35$$

$$y = (-7.71 + 8)1.37 - 0.35$$

$$y = 0.29 \times 1.37 - 0.35$$

$$y = 0.397 - 0.35$$

$$y = 0.047$$

This represents an error of

$$\varepsilon_t = \left|\frac{0.043053 - 0.47}{0.043053}\right| = 9.2\%$$

Hence, the second form is superior because it tends to minimize round-off error.

4.3 **(a)** For this case, $x_i = 0$ and $h = x$. Thus, the Taylor series is

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f^{(3)}(0)}{3!}x^3 + \cdots$$

For the exponential function,

$$f(0) = f'(0) = f''(0) = f^{(3)}(0) = 1$$

Substituting these values yields,

$$f(x) = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \cdots$$

which is the Maclaurin series expansion.

(b) The true value is $e^{-1} = 0.367879$ and the step size is $h = x_{i+1} - x_i = 1 - 0.25 = 0.75$. The complete Taylor series to the third-order term is

$$f(x_{i+1}) = e^{-x_i} - e^{-x_i}h + e^{-x_i}\frac{h^2}{2} - e^{-x_i}\frac{h^3}{3!}$$

Zero-order approximation:

$$f(1) = e^{-0.25} = 0.778801$$

$$\varepsilon_t = \left|\frac{0.367879 - 0.778801}{0.367879}\right| 100\% = 111.7\%$$

First-order approximation:

$$f(1) = 0.778801 - 0.778801(0.75) = 0.1947$$

$$\varepsilon_t = \left|\frac{0.367879 - 0.1947}{0.367879}\right| 100\% = 47.1\%$$

Second-order approximation:

$$f(1) = 0.778801 - 0.778801(0.75) + 0.778801\frac{0.75^2}{2} = 0.413738$$

$$\varepsilon_t = \left|\frac{0.367879 - 0.413738}{0.367879}\right| 100\% = 12.5\%$$

Third-order approximation:

$$f(1) = 0.778801 - 0.778801(0.75) + 0.778801\frac{0.75^2}{2} - 0.778801\frac{0.75^3}{6} = 0.358978$$

$$\varepsilon_t = \left|\frac{0.367879 - 0.358978}{0.367879}\right| 100\% = 2.42\%$$

4.4 Use $\varepsilon_s = 0.5\times 10^{2-2} = 0.5\%$. The true value = $\cos(\pi/4) = 0.707107\ldots$

zero-order:

$$\cos\left(\frac{\pi}{4}\right) \cong 1$$

$$\varepsilon_t = \left|\frac{0.707107 - 1}{0.707107}\right| 100\% = 41.42\%$$

first-order:

$$\cos\left(\frac{\pi}{4}\right) \cong 1 - \frac{(\pi/4)^2}{2} = 0.691575$$

$$\varepsilon_t = \left|\frac{0.707107 - 0.691575}{0.707107}\right| 100\% = 2.19\%$$

$$\varepsilon_a = \left|\frac{0.691575 - 1}{0.691575}\right| 100\% = 44.6\%$$

second-order:

$$\cos\left(\frac{\pi}{4}\right) \cong 0.691575 + \frac{(\pi/4)^4}{24} = 0.707429$$

$$\varepsilon_t = \left|\frac{0.707107 - 0.707429}{0.707107}\right| 100\% = 0.456\%$$

$$\varepsilon_a = \left|\frac{0.707429 - 0.691575}{0.707429}\right| 100\% = 2.24\%$$

third-order:

$$\cos\left(\frac{\pi}{4}\right) \cong 0.707429 - \frac{(\pi/4)^6}{720} = 0.707103$$

$$\varepsilon_t = \left|\frac{0.707107 - 0.707103}{0.707107}\right| 100\% = 0.0005\%$$

$$\varepsilon_a = \left|\frac{0.707103 - 0.707429}{0.707103}\right| 100\% = 0.046\%$$

Because $\varepsilon_a < 0.5\%$, we can terminate the computation.

4.5 Use $\varepsilon_s = 0.5\times10^{2-2} = 0.5\%$. The true value = sin(π/4) = 0.707107…

zero-order:

$$\sin\left(\frac{\pi}{4}\right) \cong 0.785398$$

$$\varepsilon_t = \left|\frac{0.707107 - 0.785398}{0.707107}\right| 100\% = 11.1\%$$

first-order:

$$\sin\left(\frac{\pi}{4}\right) \cong 0.785398 - \frac{(\pi/4)^3}{6} = 0.704653$$

$$\varepsilon_t = \left|\frac{0.707107 - 0.704653}{0.707107}\right| 100\% = 0.347\%$$

$$\varepsilon_a = \left|\frac{0.704653 - 0.785398}{0.704653}\right| 100\% = 11.46\%$$

second-order:

$$\sin\left(\frac{\pi}{4}\right) \cong 0.704653 + \frac{(\pi/4)^5}{120} = 0.707143$$

$$\varepsilon_t = \left|\frac{0.707107 - 0.707143}{0.707107}\right| 100\% = 0.0051\%$$

$$\varepsilon_a = \left|\frac{0.707143 - 0.704653}{0.707143}\right| 100\% = 0.352\%$$

Because $\varepsilon_a < 0.5\%$, we can terminate the computation.

4.6 The true value is $f(2) = 102$.

zero order:

$$f(2) = f(1) = -62 \qquad \varepsilon_t = \left|\frac{102 - (-62)}{102}\right| 100\% = 160.8\%$$

first order:

$$f'(1) = 75(1)^2 - 12(1) + 7 = 70$$

$$f(2) = -62 + 70(1) = 8 \qquad \varepsilon_t = \left|\frac{102 - 8}{102}\right| 100\% = 92.1\%$$

second order:

$$f''(1) = 150(1) - 12 = 138$$

$$f(2) = 8 + \frac{138}{2}(1)^2 = 77 \qquad \varepsilon_t = \left|\frac{102 - 77}{102}\right| 100\% = 24.5\%$$

third order:

$$f^{(3)}(1) = 150$$

$$f(2) = 77 + \frac{150}{6}(1)^3 = 102 \qquad \varepsilon_t = \left|\frac{102 - 102}{102}\right| 100\% = 0.0\%$$

Because we are working with a third-order polynomial, the error is zero. This is due to the fact that cubics have zero fourth and higher derivatives.

4.7 The true value is ln(3) = 1.098612

zero order:

$$f(3) = f(1) = 0 \qquad \varepsilon_t = \left|\frac{1.098612 - 0}{1.098612}\right| 100\% = 100\%$$

first order:

$$f'(x) = \frac{1}{x} \qquad f'(1) = 1$$

$$f(3) = 0 + 1(2) = 2 \qquad \varepsilon_t = \left|\frac{1.098612 - 2}{1.098612}\right| 100\% = 82.05\%$$

second order:

$$f''(x) = -\frac{1}{x^2} \qquad f''(1) = -1$$

$$f(3) = 2 - 1\frac{2^2}{2} = 0 \qquad \varepsilon_t = \left|\frac{1.098612 - 0}{1.098612}\right| 100\% = 100\%$$

third order:

$$f^{(3)}(x) = \frac{2}{x^3} \qquad f''(1) = 2$$

$$f(3) = 0 + 2\frac{2^3}{6} = 2.66667 \qquad \varepsilon_t = \left|\frac{1.098612 - 2.66667}{1.098612}\right| 100\% = 142.7\%$$

fourth order:

$$f^{(4)}(x) = -\frac{6}{x^4} \qquad f^{(4)}(1) = -6$$

$$f(3) = 2.66666 - 6\frac{2^4}{24} = -1.33333 \qquad \varepsilon_t = \left|\frac{1.098612 - (-1.33333)}{1.098612}\right| 100\% = 221.4\%$$

The series is diverging. A smaller step size is required to obtain convergence.

4.8 The first derivative of the function at $x = 2$ can be evaluated as

$$f'(2) = 75(2)^2 - 12(2) + 7 = 283$$

The points needed to form the finite divided differences can be computed as

$x_{i-1} = 1.75$	$f(x_{i-1}) = 39.85938$
$x_i = 2.0$	$f(x_i) = 102$
$x_{i+1} = 2.25$	$f(x_{i+1}) = 182.1406$

forward:

$$f'(2) = \frac{182.1406 - 102}{0.25} = 320.5625 \qquad |E_t| = |283 - 320.5625| = 37.5625$$

backward:

$$f'(2) = \frac{102 - 39.85938}{0.25} = 248.5625 \qquad |E_t| = |283 - 248.5625| = 34.4375$$

centered:

$$f'(2) = \frac{182.1406 - 39.85938}{0.5} = 284.5625 \quad E_t = 283 - 284.5625 = -1.5625$$

Both the forward and backward differences should have errors approximately equal to

$$|E_t| \approx \frac{f''(x_i)}{2} h$$

The second derivative can be evaluated as

$$f''(2) = 150(2) - 12 = 288$$

Therefore,

$$|E_t| \approx \frac{288}{2} 0.25 = 36$$

which is similar in magnitude to the computed errors.

For the central difference,

$$E_t \approx -\frac{f^{(3)}(x_i)}{6}h^2$$

The third derivative of the function is 150 and

$$E_t \approx -\frac{150}{6}(0.25)^2 = -1.5625$$

which is exact. This occurs because the underlying function is a cubic equation that has zero fourth and higher derivatives.

4.9 The second derivative of the function at $x = 2$ can be evaluated as

$$f'(2) = 150(2) - 12 = 288$$

For $h = 0.2$,

$$f''(2) = \frac{164.56 - 2(102) + 50.96}{(0.2)^2} = 288$$

For $h = 0.1$,

$$f''(2) = \frac{131.765 - 2(102) + 75.115}{(0.1)^2} = 288$$

Both are exact because the errors are a function of fourth and higher derivatives which are zero for a 3rd-order polynomial.

4.10 Use $\varepsilon_s = 0.5\times10^{2-2} = 0.5\%$. The true value = 1/(1 – 0.1) = 1.11111…

zero-order:

$$\frac{1}{1-0.1} \cong 1$$

$$\varepsilon_t = \left|\frac{1.11111-1}{1.11111}\right|100\% = 10\%$$

first-order:

$$\frac{1}{1-0.1} \cong 1 + 0.1 = 1.1$$

$$\varepsilon_t = \left|\frac{1.11111 - 1.1}{1.11111}\right| 100\% = 1\%$$

$$\varepsilon_a = \left|\frac{1.1 - 1}{1.1}\right| 100\% = 9.0909\%$$

second-order:

$$\frac{1}{1-0.1} \cong 1 + 0.1 + 0.01 = 1.11$$

$$\varepsilon_t = \left|\frac{1.11111 - 1.11}{1.11111}\right| 100\% = 0.1\%$$

$$\varepsilon_a = \left|\frac{1.11 - 1.1}{1.11}\right| 100\% = 0.9009\%$$

third-order:

$$\frac{1}{1-0.1} \cong 1 + 0.1 + 0.01 + 0.001 = 1.111$$

$$\varepsilon_t = \left|\frac{1.11111 - 1.111}{1.11111}\right| 100\% = 0.01\%$$

$$\varepsilon_a = \left|\frac{1.111 - 1.11}{1.111}\right| 100\% = 0.090009\%$$

The approximate error has fallen below 0.5% so the computation can be terminated.

4.11 Here are the function and its derivatives

$$f(x) = x - 1 - \frac{1}{2}\sin x$$

$$f'(x) = 1 - \frac{1}{2}\cos x$$

$$f''(x) = \frac{1}{2}\sin x$$

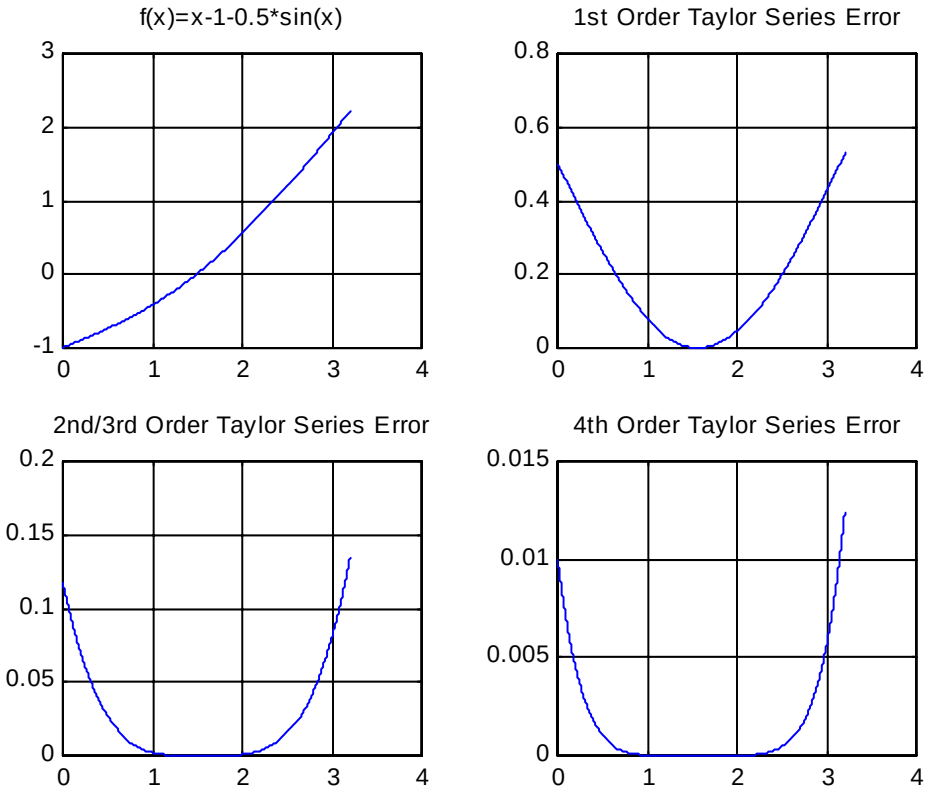
$$f^{(3)}(x) = \frac{1}{2}\cos x$$

$$f^{(4)}(x) = -\frac{1}{2}\sin x$$

Using the Taylor Series expansion, we obtain the following 1st, 2nd, 3rd, and 4th order Taylor Series functions shown below in the MATLAB program–`f1`, `f2`, and `f4`. Note the 2nd and 3rd order Taylor Series functions are the same.

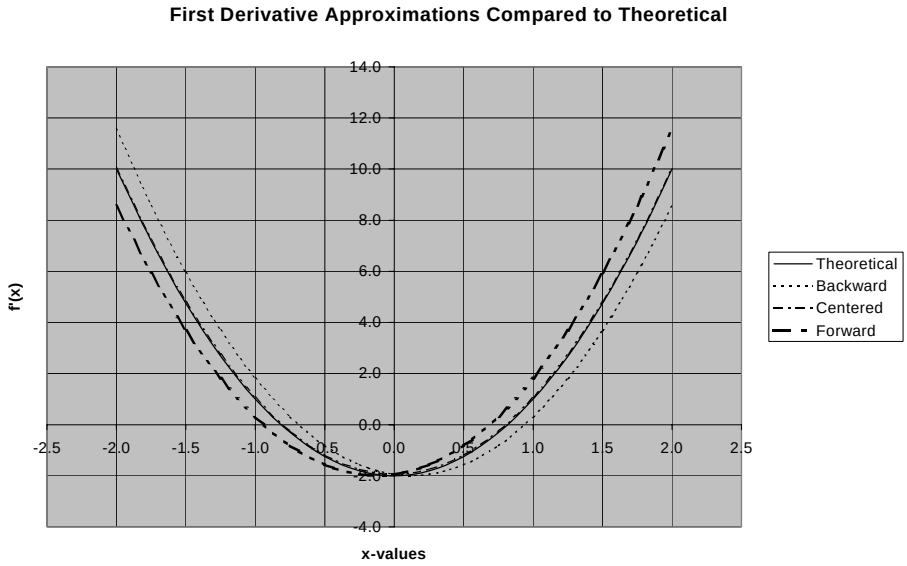
From the plots below, we see that the answer is the 4th Order Taylor Series expansion.

```
x=0:0.001:3.2;
f=x-1-0.5*sin(x);
subplot(2,2,1);
plot(x,f);grid;title('f(x)=x-1-0.5*sin(x)');hold on

f1=x-1.5;
e1=abs(f-f1);   %Calculates the absolute value of the
difference/error
subplot(2,2,2);
plot(x,e1);grid;title('1st Order Taylor Series Error');

f2=x-1.5+0.25.*((x-0.5*pi).^2);
e2=abs(f-f2);
subplot(2,2,3);
plot(x,e2);grid;title('2nd/3rd Order Taylor Series Error');

f4=x-1.5+0.25.*((x-0.5*pi).^2)-(1/48)*((x-0.5*pi).^4);
e4=abs(f4-f);
subplot(2,2,4);
plot(x,e4);grid;title('4th Order Taylor Series Error');hold off
```



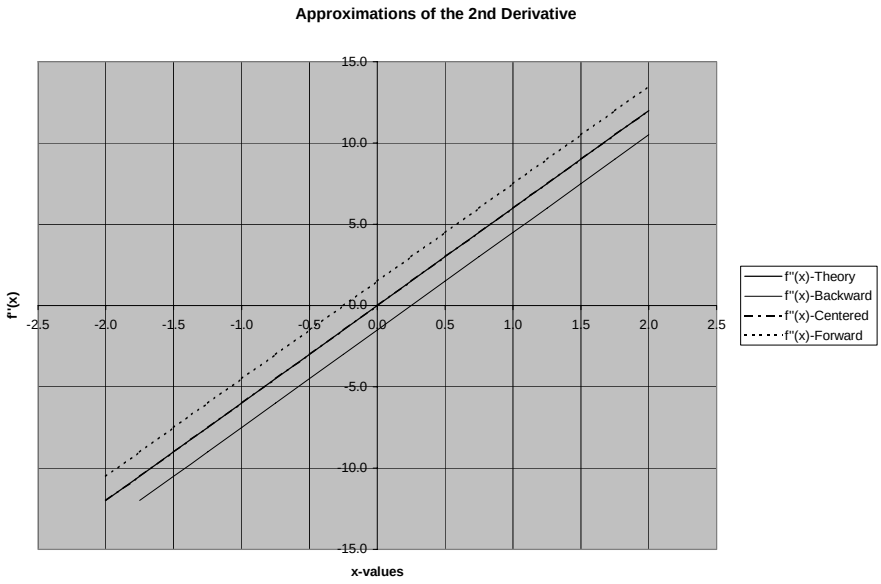
4.12

x	f(x)	f(x-1)	f(x+1)	f'(x)-Theory	f'(x)-Back	f'(x)-Cent	f'(x)-Forw
-2.000	0.000	-2.891	2.141	10.000	11.563	10.063	8.563
-1.750	2.141	0.000	3.625	7.188	8.563	7.250	5.938
-1.500	3.625	2.141	4.547	4.750	5.938	4.813	3.688
-1.250	4.547	3.625	5.000	2.688	3.688	2.750	1.813
-1.000	5.000	4.547	5.078	1.000	1.813	1.063	0.313
-0.750	5.078	5.000	4.875	-0.313	0.313	-0.250	-0.813
-0.500	4.875	5.078	4.484	-1.250	-0.813	-1.188	-1.563
-0.250	4.484	4.875	4.000	-1.813	-1.563	-1.750	-1.938
0.000	4.000	4.484	3.516	-2.000	-1.938	-1.938	-1.938
0.250	3.516	4.000	3.125	-1.813	-1.938	-1.750	-1.563
0.500	3.125	3.516	2.922	-1.250	-1.563	-1.188	-0.813
0.750	2.922	3.125	3.000	-0.313	-0.813	-0.250	0.313
1.000	3.000	2.922	3.453	1.000	0.313	1.063	1.813
1.250	3.453	3.000	4.375	2.688	1.813	2.750	3.688
1.500	4.375	3.453	5.859	4.750	3.688	4.813	5.938
1.750	5.859	4.375	8.000	7.188	5.938	7.250	8.563
2.000	8.000	5.859	10.891	10.000	8.563	10.063	11.563



x	f(x)	f(x-1)	f(x+1)	f(x-2)	f(x+2)	f''(x)-Theory	f''(x)-Back	f''(x)-Cent	f''(x)-Forw
-2.000	0.000	-2.891	2.141	3.625	3.625	-12.000	150.500	-12.000	-10.500
-1.750	2.141	0.000	3.625	-2.891	4.547	-10.500	-12.000	-10.500	-9.000
-1.500	3.625	2.141	4.547	0.000	5.000	-9.000	-10.500	-9.000	-7.500
-1.250	4.547	3.625	5.000	2.141	5.078	-7.500	-9.000	-7.500	-6.000
-1.000	5.000	4.547	5.078	3.625	4.875	-6.000	-7.500	-6.000	-4.500
-0.750	5.078	5.000	4.875	4.547	4.484	-4.500	-6.000	-4.500	-3.000
-0.500	4.875	5.078	4.484	5.000	4.000	-3.000	-4.500	-3.000	-1.500
-0.250	4.484	4.875	4.000	5.078	3.516	-1.500	-3.000	-1.500	0.000
0.000	4.000	4.484	3.516	4.875	3.125	0.000	-1.500	0.000	1.500
0.250	3.516	4.000	3.125	4.484	2.922	1.500	0.000	1.500	3.000

0.500	3.125	3.516	2.922	4.000	3.000	3.000	1.500	3.000	4.500
0.750	2.922	3.125	3.000	3.516	3.453	4.500	3.000	4.500	6.000
1.000	3.000	2.922	3.453	3.125	4.375	6.000	4.500	6.000	7.500
1.250	3.453	3.000	4.375	2.922	5.859	7.500	6.000	7.500	9.000
1.500	4.375	3.453	5.859	3.000	8.000	9.000	7.500	9.000	10.500
1.750	5.859	4.375	8.000	3.453	10.891	10.500	9.000	10.500	12.000
2.000	8.000	5.859	10.891	4.375	14.625	12.000	10.500	12.000	13.500



4.13

```
function eps = macheps
% determines the machine epsilon
e = 1;
while e+1>1
  e = e/2;
end
eps = 2*e;

>> macheps

ans =
  2.2204e-016

>> eps

ans =
  2.2204e-016
```

CHAPTER 5

5.1 The function to evaluate is

$$f(c_d) = \sqrt{\frac{gm}{c_d}}\tanh\left(\sqrt{\frac{gc_d}{m}}t\right) - v(t)$$

or substituting the given values

$$f(c_d) = \sqrt{\frac{9.81(80)}{c_d}}\tanh\left(\sqrt{\frac{9.81c_d}{80}}4\right) - 36$$

The first iteration is

$$x_r = \frac{0.1 + 0.2}{2} = 0.15$$

$$f(0.1)\,f(0.15) = 0.860291(-0.204516) = -0.175944$$

Therefore, the root is in the first interval and the upper guess is redefined as $x_u = 0.15$. The second iteration is

$$x_r = \frac{0.1 + 0.15}{2} = 0.125$$

$$\varepsilon_a = \left|\frac{0.125 - 0.15}{0.125}\right|100\% = 20\%$$

$$f(0.1)\,f(0.125) = 0.860291(0.318407) = 0.273923$$

Therefore, the root is in the second interval and the lower guess is redefined as $x_u = 0.125$. The remainder of the iterations are displayed in the following table:

i	x_l	$f(x_l)$	x_u	$f(x_u)$	x_r	$f(x_r)$	$\lvert\varepsilon_a\rvert$
1	0.1	0.86029	0.2	–1.19738	0.15	–0.20452	
2	0.1	0.86029	0.15	–0.20452	0.125	0.31841	20.00%
3	0.125	0.31841	0.15	–0.20452	0.1375	0.05464	9.09%
4	0.1375	0.05464	0.15	–0.20452	0.14375	–0.07551	4.35%
5	0.1375	0.05464	0.14375	–0.07551	0.140625	–0.01058	2.22%
6	0.1375	0.05464	0.140625	–0.01058	0.1390625	0.02199	1.12%

Thus, after six iterations, we obtain a root estimate of **0.1390625** with an approximate error of 1.12%.

5.2

```
function root = bisectnew(func,xl,xu,Ead)
% bisectnew(xl,xu,es,maxit):
```

```
%   uses bisection method to find the root of a function
%   with a fixed number of iterations to attain
%   a prespecified tolerance
% input:
%   func = name of function
%   xl, xu = lower and upper guesses
%   Ead = (optional) desired tolerance (default = 0.000001)
% output:
%   root = real root

if func(xl)*func(xu)>0  %if guesses do not bracket a sign change
  error('no bracket')    %display an error message and terminate
end
% if necessary, assign default values
if nargin<4, Ead = 0.000001; end   %if Ead blank set to 0.000001
% bisection
xr = xl;
% compute n and round up to next highest integer
n = round(1 + log2((xu - xl)/Ead) + 0.5);
for i = 1:n
  xrold = xr;
  xr = (xl + xu)/2;
  if xr ~= 0, ea = abs((xr - xrold)/xr) * 100; end
  test = func(xl)*func(xr);
  if test < 0
    xu = xr;
  elseif test > 0
    xl = xr;
  else
    ea = 0;
  end
end
root = xr;
```

The following is a MATLAB session that uses the function to solve Prob. 5.1 with $E_{a,d}$ = 0.0001.

```
>> fcd = inline('sqrt(9.81*80/cd)*tanh(sqrt(9.81*cd/80)*4)-36','cd')

fcd =
     Inline function:
     fcd(cd) = sqrt(9.81*80/cd)*tanh(sqrt(9.81*cd/80)*4)-36

>> format long
>> bisectnew(fcd,0.1,0.2,0.0001)

ans =
   0.14008789062500
```

5.3 The function to evaluate is

$$f(c_d) = \sqrt{\frac{9.81(80)}{c_d}}\tanh\left(\sqrt{\frac{9.81c_d}{80}}4\right) - 36$$

The first iteration is

$$x_r = 0.2 - \frac{-1.19738(0.1 - 0.2)}{0.86029 - (-1.19738)} = 0.141809$$

$$f(0.1)\,f(0.141809) = 0.860291(-0.03521) = -0.030292$$

Therefore, the root is in the first interval and the upper guess is redefined as $x_u = 0.141809$. The second iteration is

$$x_r = 0.141809 - \frac{-0.03521(0.1 - 0.141809)}{0.86029 - (-0.03521)} = 0.140165$$

$$\varepsilon_a = \left|\frac{0.140165 - 0.141809}{0.140165}\right| 100\% = 1.17\%$$

Therefore, after only two iterations we obtain a root estimate of 0.140165 with an approximate error of 1.17% which is below the stopping criterion of 2%.

5.4

```
function root = falsepos(func,xl,xu,es,maxit)
% falsepos(xl,xu,es,maxit):
%   uses the false position method to find the root
%   of the function func
% input:
%   func = name of function
%   xl, xu = lower and upper guesses
%   es = (optional) stopping criterion (%) (default = 0.001)
%   maxit = (optional) maximum allowable iterations (default = 50)
% output:
%   root = real root
if func(xl)*func(xu)>0  %if guesses do not bracket a sign change
  error('no bracket')    %display an error message and terminate
end
% default values
if nargin<5, maxit=50; end
if nargin<4, es=0.001; end
% false position
iter = 0;
xr = xl;
while (1)
  xrold = xr;
  xr = xu - func(xu)*(xl - xu)/(func(xl) - func(xu));
  iter = iter + 1;
  if xr ~= 0, ea = abs((xr - xrold)/xr) * 100; end
  test = func(xl)*func(xr);
  if test < 0
    xu = xr;
  elseif test > 0
    xl = xr;
  else
    ea = 0;
```

```
  end
  if ea <= es | iter >= maxit, break, end
end
root = xr;
```

The following is a MATLAB session that uses the function to solve Prob. 5.1:

```
>> fcd = inline('sqrt(9.81*80/cd)*tanh(sqrt(9.81*cd/80)*4)-36','cd')

fcd =
     Inline function:
     fcd(cd) = sqrt(9.81*80/cd)*tanh(sqrt(9.81*cd/80)*4)-36

>> format long
>> falsepos(fcd,0.1,0.2,2)

ans =
   0.14016503741282
```

5.5 Solve for the reactions:

$R_1 = 265$ lbs. $R_2 = 285$ lbs.

Write beam equations:

$0<x<3$

$$M + (16.667x^2)\frac{x}{3} - 265x = 0$$

$$(1) \quad M = 265 - 5.55x^3$$

$3<x<6$

$$M + 100(x-3)(\frac{x-3}{2}) + 150(x - \frac{2}{3}(3)) - 265x = 0$$

$$(2) \quad M = -50x^2 + 415x - 150$$

$6<x<10$

$$M = 150(x - \frac{2}{3}(3)) + 300(x - 4.5) - 265x$$

$$(3) \quad M = -185x + 1650$$

$10<x<12$

$$M + 100(12 - x) = 0$$

$$(4) \quad M = 100x - 1200$$

Combining Equations:

Because the curve crosses the axis between 6 and 10, use (3).

(3) $M = -185x + 1650$

Set $x_L = 6; x_U = 10$

$M(x_L) = 540$
$M(x_U) = -200$ $\quad x_r = \frac{x_L + x_U}{2} = 8$
$M(x_R) = 170 \rightarrow$ *replaces* x_L

$M(x_L) = 170$
$M(x_U) = -200$ $\quad x_r = \frac{8+10}{2} = 9$
$M(x_R) = -15 \rightarrow$ *replaces* x_U

$M(x_L) = 170$
$M(x_U) = -15$ $\quad x_r = \frac{8+9}{2} = 8.5$
$M(x_R) = 77.5 \rightarrow$ *replaces* x_L

$M(x_L) = 77.5$
$M(x_U) = -15$ $\quad x_r = \frac{8.5+9}{2} = 8.75$
$M(x_R) = 31.25 \rightarrow$ *replaces* x_L

$M(x_L) = 31.25$
$M(x_U) = -15$ $\quad x_r = \frac{8.75+9}{2} = 8.875$
$M(x_R) = 8.125 \rightarrow$ *replaces* x_L

$M(x_L) = 8.125$
$M(x_U) = -15$ $\quad x_r = \frac{8.875+9}{2} = 8.9375$
$M(x_R) = -3.4375 \rightarrow$ *replaces* x_U

$M(x_L) = 8.125$
$M(x_U) = -3.4375$ $\quad x_r = \frac{8.875+8.9375}{2} = 8.90625$
$M(x_R) = 2.34375 \rightarrow$ *replaces* x_L

$M(x_L) = 2.34375$
$M(x_U) = -3.4375$ $\quad x_r = \frac{8.90625+8.9375}{2} = 8.921875$
$M(x_R) = -0.546875 \rightarrow$ *replaces* x_U

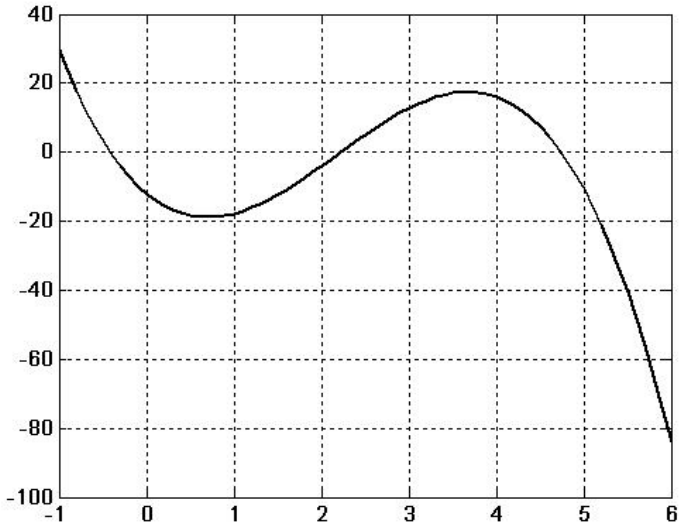
$M(x_L) = 2.34375$
$M(x_U) = -0.546875$ $\quad x_r = \frac{8.90625+8.921875}{2} = 8.9140625$

$M(x_R) = 0.8984$ **Therefore,** $x = 8.91$ *feet*

5.6 (a) The graph can be generated with MATLAB

```
>> x=[-1:0.1:6];
```

```
>> f=-12-21*x+18*x.^2-2.75*x.^3;
>> plot(x,f)
>> grid
```



This plot indicates that roots are located at about –0.4, 2.25 and 4.7.

(b) Using bisection, the first iteration is

$$x_r = \frac{-1+0}{2} = -0.5$$

$$f(-1)\,f(-0.5) = 29.75(3.34375) = 99.47656$$

Therefore, the root is in the second interval and the lower guess is redefined as $x_l = -0.5$. The second iteration is

$$x_r = \frac{-0.5+0}{2} = -0.25$$

$$\varepsilon_a = \left|\frac{-0.25-(-0.5)}{-0.25}\right| 100\% = 100\%$$

$$f(-0.5)\,f(-0.25) = 3.34375(-5.5820313) = -18.66492$$

Therefore, the root is in the first interval and the upper guess is redefined as $x_u = -0.25$. The remainder of the iterations are displayed in the following table:

i	x_l	$f(x_l)$	x_u	$f(x_u)$	x_r	$f(x_r)$	$\lvert\varepsilon_a\rvert$
1	–1	29.75	0	–12	–0.5	3.34375	
2	–0.5	3.34375	0	–12	–0.25	–5.5820313	100.00%
3	–0.5	3.34375	–0.25	–5.5820313	–0.375	–1.4487305	33.33%

4	–0.5	3.34375	–0.375	–1.4487305	–0.4375	0.8630981	14.29%
5	–0.4375	0.863098	–0.375	–1.4487305	–0.40625	–0.3136673	7.69%
6	–0.4375	0.863098	–0.40625	–0.3136673	–0.421875	0.2694712	3.70%
7	–0.42188	0.269471	–0.40625	–0.3136673	–0.414063	–0.0234052	1.89%
8	–0.42188	0.269471	–0.41406	–0.0234052	–0.417969	0.1227057	0.93%

Thus, after eight iterations, we obtain a root estimate of **–0.417969** with an approximate error of 0.93%, which is below the stopping criterion of 1%.

(c) Using false position, the first iteration is

$$x_r = 0 - \frac{-12(-1-0)}{29.75-(-12)} = -0.287425$$

$$f(-1)\,f(-0.287425) = 29.75(-4.4117349) = -131.2491$$

Therefore, the root is in the first interval and the upper guess is redefined as $x_u = -0.287425$. The second iteration is

$$x_r = -0.287425 - \frac{-4.4117349(-1-(-0.287425))}{29.75-(-4.4117349)} = -0.3794489$$

$$\varepsilon_a = \left|\frac{-0.3794489-(-0.2874251)}{-0.3794489}\right| 100\% = 24.25\%$$

$$f(-1)\,f(-0.3794489) = 29.75(-1.2896639) = -38.3675$$

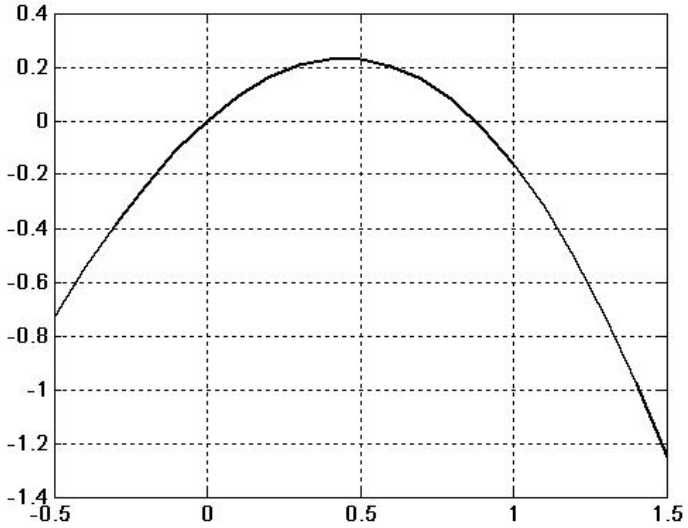
Therefore, the root is in the first interval and the upper guess is redefined as $x_u = -0.379449$. The remainder of the iterations are displayed in the following table:

i	x_l	$f(x_l)$	x_u	$f(x_u)$	x_r	$f(x_r)$	$\lvert\varepsilon_a\rvert$
1	–1	29.75	0	–12	–0.287425	–4.4117349	
2	–1	29.75	–0.28743	–4.4117349	–0.379449	–1.2896639	24.25%
3	–1	29.75	–0.37945	–1.2896639	–0.405232	–0.3512929	6.36%
4	–1	29.75	–0.40523	–0.3512929	–0.412173	–0.0938358	1.68%
5	–1	29.75	–0.41217	–0.0938358	–0.414022	–0.0249338	0.45%

Therefore, after five iterations we obtain a root estimate of **–0.414022** with an approximate error of 0.45%, which is below the stopping criterion of 1%.

5.7 A graph of the function can be generated with MATLAB

```
>> x=[-0.5:0.1:1.5];
>> f=sin(x)-x.^2;
>> plot(x,f)
>> grid
```



This plot indicates that a nontrivial root (i.e., nonzero) is located at about 0.85.

Using bisection, the first iteration is

$$x_r = \frac{0.5+1}{2} = 0.75$$

$$f(0.5)\,f(0.75) = 0.229426(0.1191388) = 0.027333$$

Therefore, the root is in the second interval and the lower guess is redefined as $x_l = 0.75$. The second iteration is

$$x_r = \frac{0.75+1}{2} = 0.875$$

$$\varepsilon_a = \left|\frac{0.875-0.75}{0.875}\right|100\% = 14.29\%$$

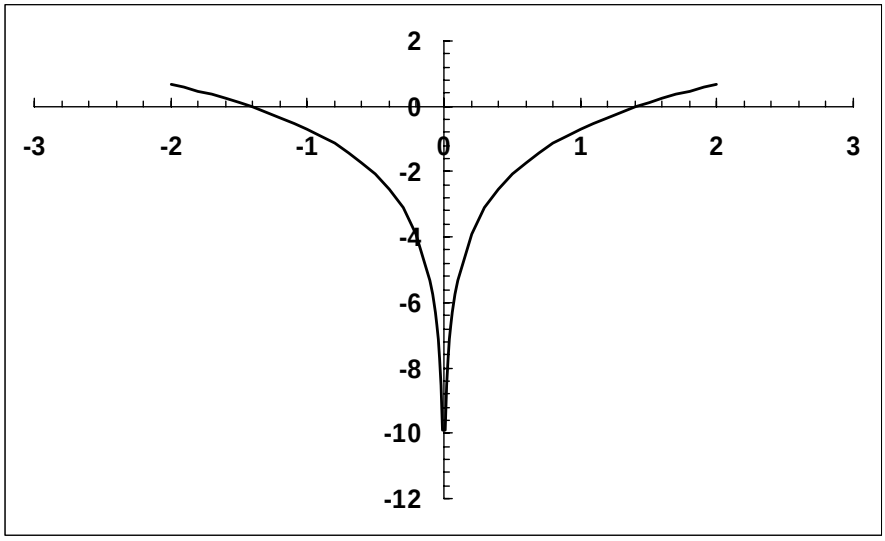
$$f(0.75)\,f(0.875) = 0.119139(0.0019185) = 0.000229$$

Because the product is positive, the root is in the second interval and the lower guess is redefined as $x_l = 0.875$. The remainder of the iterations are displayed in the following table:

i	x_l	$f(x_l)$	x_u	$f(x_u)$	x_r	$f(x_r)$	$\lvert\varepsilon_a\rvert$
1	0.5	0.229426	1	–0.158529	0.75	0.1191388	
2	0.75	0.119139	1	–0.158529	0.875	0.0019185	14.29%
3	0.875	0.001919	1	–0.158529	0.9375	–0.0728251	6.67%
4	0.875	0.001919	0.9375	–0.0728251	0.90625	–0.0340924	3.45%
5	0.875	0.001919	0.90625	–0.0340924	0.890625	–0.0157479	1.75%

Therefore, after five iterations we obtain a root estimate of **0.890625** with an approximate error of 1.75%, which is below the stopping criterion of 2%.

5.8 **(a)** A graph of the function indicates a positive real root at approximately $x = 1.4$.



(b) Using bisection, the first iteration is

$$x_r = \frac{0.5 + 2}{2} = 1.25$$

$$f(0.5)\,f(1.25) = -2.08629(-0.2537129) = 0.52932$$

Therefore, the root is in the second interval and the lower guess is redefined as $x_l = 1.25$. The second iteration is

$$x_r = \frac{1.25 + 2}{2} = 1.625$$

$$\varepsilon_a = \left|\frac{1.625 - 1.25}{1.625}\right| 100\% = 23.08\%$$

$$f(1.25)\,f(1.625) = -0.253713(0.2710156) = -0.06876$$

Therefore, the root is in the first interval and the upper guess is redefined as $x_u = 1.625$. The remainder of the iterations are displayed in the following table:

i	x_l	$f(x_l)$	x_u	$f(x_u)$	x_r	$f(x_r)$	$\lvert\varepsilon_a\rvert$
1	0.5	–2.08629	2	0.6862944	1.25	–0.2537129	
2	1.25	–0.25371	2	0.6862944	1.625	0.2710156	23.08%
3	1.25	–0.25371	1.625	0.2710156	1.4375	0.025811	13.04%

Thus, after three iterations, we obtain a root estimate of **1.4375** with an approximate error of 13.04%.

(c) Using false position, the first iteration is

$$x_r = 2 - \frac{0.6862944(0.5-2)}{-2.086294-0.6862944} = 1.628707$$

$$f(0.5)\,f(1.628707) = -2.086294(0.2755734) = -0.574927$$

Therefore, the root is in the first interval and the upper guess is redefined as x_u = 1.628707. The second iteration is

$$x_r = 0.2755734 - \frac{1.4970143(0.5-1.628707)}{-2.086294-0.2755734} = 1.4970143$$

$$\varepsilon_a = \left|\frac{1.4970143-1.6287074}{1.4970143}\right| 100\% = 8.8\%$$

$$f(0.5)\,f(1.4970143) = -2.086294(0.1069453) = -0.223119$$

Therefore, the root is in the first interval and the upper guess is redefined as x_u = 1.497014. The remainder of the iterations are displayed in the following table:

i	x_l	$f(x_l)$	x_u	$f(x_u)$	x_r	$f(x_r)$	$\lvert\varepsilon_a\rvert$
1	0.5	–2.08629	2	0.6862944	1.6287074	0.2755734	
2	0.5	–2.08629	1.628707	0.2755734	1.4970143	0.1069453	8.80%
3	0.5	–2.08629	1.497014	0.1069453	1.4483985	0.040917	3.36%

Therefore, after three iterations we obtain a root estimate of **1.4483985** with an approximate error of 3.36%.

5.9 (a) Equation (5.6) can be used to determine the number of iterations

$$n = 1 + \log_2\left(\frac{\Delta x^0}{E_{a,d}}\right) = 1 + \log_2\left(\frac{35}{0.05}\right) = 10.45121$$

which can be rounded up to 11 iterations.

(b) Here is an M-file that evaluates the temperature in ^{o}C using 11 iterations of bisection based on a given value of the oxygen saturation concentration in freshwater:

```
function TC = TempEval(osf)
% function to evaluate the temperature in degrees C based
% on the oxygen saturation concentration in freshwater (osf).
xl = 0 + 273.15;
xu = 35 + 273.15;
if fTa(xl,osf)*fTa(xu,osf)>0  %if guesses do not bracket
  error('no bracket')   %display an error message and terminate
```

```
end
xr = xl;
for i = 1:11
  xrold = xr;
  xr = (xl + xu)/2;
  if xr ~= 0, ea = abs((xr - xrold)/xr) * 100; end
  test = fTa(xl,osf)*fTa(xr,osf);
  if test < 0
    xu = xr;
  elseif test > 0
    xl = xr;
  else
    ea = 0;
  end
end
TC = xr - 273.15;
end

function f = fTa(Ta, osf)
f = -139.34411 + 1.575701e5/Ta - 6.642308e7/Ta^2;
f = f + 1.2438e10/Ta^3 - 8.621949e11/Ta^4;
f = f - log(osf);
```

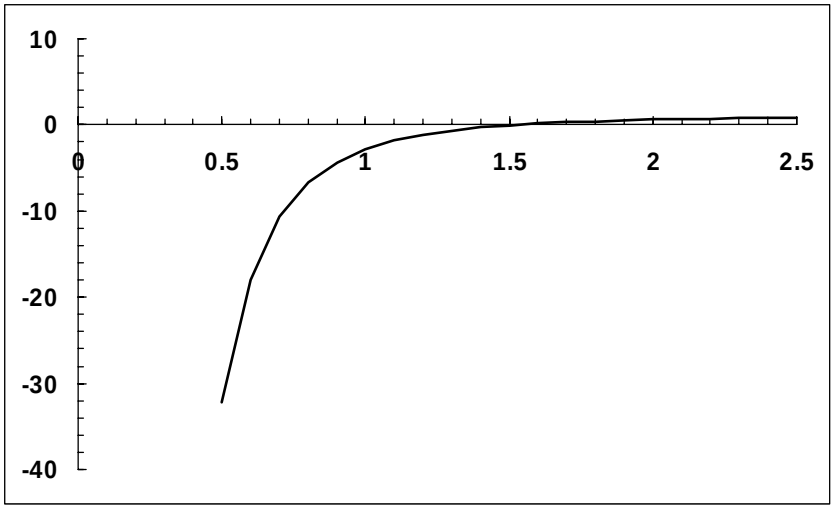
The function can be used to evaluate the test cases:

```
>> TempEval(8)

ans =
   26.7798

>> TempEval(10)

ans =
   15.3979

>> TempEval(14)

ans =
    1.5552
```

5.10 (a) The function to be evaluated is

$$f(y) = 1 - \frac{400}{9.81(3y + y^2/2)^3}(3 + y)$$

A graph of the function indicates a positive real root at approximately 1.5.



(b) Using bisection, the first iteration is

$$x_r = \frac{0.5 + 2.5}{2} = 1.5$$

$$f(0.5)\, f(1.5) = -32.2582(-0.030946) = 0.998263$$

Therefore, the root is in the second interval and the lower guess is redefined as $x_l = 1.5$. The second iteration is

$$x_r = \frac{1.5 + 2.5}{2} = 2$$

$$\varepsilon_a = \left|\frac{2 - 1.5}{2}\right| 100\% = 25\%$$

$$f(1.5)\, f(2) = -0.030946(0.601809) = -0.018624$$

Therefore, the root is in the first interval and the upper guess is redefined as $x_u = 2$. The remainder of the iterations are displayed in the following table:

i	x_l	$f(x_l)$	x_u	$f(x_u)$	x_r	$f(x_r)$	$\lvert\varepsilon_a\rvert$
1	0.5	–32.2582	2.5	0.813032	1.5	–0.030946	
2	1.5	–0.03095	2.5	0.813032	2	0.601809	25.00%
3	1.5	–0.03095	2	0.601809	1.75	0.378909	14.29%
4	1.5	–0.03095	1.75	0.378909	1.625	0.206927	7.69%
5	1.5	–0.03095	1.625	0.206927	1.5625	0.097956	4.00%
6	1.5	–0.03095	1.5625	0.097956	1.53125	0.036261	2.04%
7	1.5	–0.03095	1.53125	0.036261	1.515625	0.003383	1.03%
8	1.5	–0.03095	1.515625	0.003383	1.5078125	–0.013595	0.52%

After eight iterations, we obtain a root estimate of **1.5078125** with an approximate error of 0.52%.

(c) Using false position, the first iteration is

$$x_r = 2.5 - \frac{0.81303(0.5 - 2.5)}{-32.2582 - 0.81303} = 2.45083$$

$$f(0.5)\,f(2.45083) = -32.25821(0.79987) = -25.80248$$

Therefore, the root is in the first interval and the upper guess is redefined as $x_u = 2.45083$. The second iteration is

$$x_r = 2.45083 - \frac{0.79987(0.5 - 2.45083)}{-32.25821 - 0.79987} = 2.40363$$

$$\varepsilon_a = \left|\frac{2.40363 - 2.45083}{2.40363}\right| 100\% = 1.96\%$$

$$f(0.5)\,f(2.40363) = -32.2582(0.78612) = -25.35893$$

The root is in the first interval and the upper guess is redefined as $x_u = 2.40363$. The remainder of the iterations are displayed in the following table:

i	x_l	$f(x_l)$	x_u	$f(x_u)$	x_r	$f(x_r)$	$\lvert\varepsilon_a\rvert$
1	0.5	–32.2582	2.50000	0.81303	2.45083	0.79987	
2	0.5	–32.2582	2.45083	0.79987	2.40363	0.78612	1.96%
3	0.5	–32.2582	2.40363	0.78612	2.35834	0.77179	1.92%
4	0.5	–32.2582	2.35834	0.77179	2.31492	0.75689	1.88%
5	0.5	–32.2582	2.31492	0.75689	2.27331	0.74145	1.83%
6	0.5	–32.2582	2.27331	0.74145	2.23347	0.72547	1.78%
7	0.5	–32.2582	2.23347	0.72547	2.19534	0.70900	1.74%
8	0.5	–32.2582	2.19534	0.70900	2.15888	0.69206	1.69%
9	0.5	–32.2582	2.15888	0.69206	2.12404	0.67469	1.64%
10	0.5	–32.2582	2.12404	0.67469	2.09077	0.65693	1.59%

After ten iterations we obtain a root estimate of **2.09077** with an approximate error of 1.59%. Thus, after ten iterations, the false position method is converging at a very slow pace and is still far from the root in the vicinity of 1.5 that we detected graphically.

Discussion: This is a classic example of a case where false position performs poorly and is inferior to bisection. Insight into these results can be gained by examining the plot that was developed in part **(a)**. This function violates the premise upon which false position was based–that is, if $f(x_u)$ is much closer to zero than $f(x_l)$, then the root is closer to x_u than to x_l (recall Fig. 5.8). Because of the shape of the present function, the opposite is true.

CHAPTER 6

6.1 The function can be set up for fixed-point iteration by solving it for x

$$x_{i+1} = \sin\left(\sqrt{x_i}\right)$$

Using an initial guess of $x_0 = 0.5$, the first iteration yields

$$x_1 = \sin\left(\sqrt{0.5}\right) = 0.649637$$

$$\left|\varepsilon_a\right| = \left|\frac{0.649637 - 0.5}{0.649637}\right| \times 100\% = 23\%$$

Second iteration:

$$x_2 = \sin\left(\sqrt{0.649637}\right) = 0.721524$$

$$\left|\varepsilon_a\right| = \left|\frac{0.721524 - 0.649637}{0.721524}\right| \times 100\% = 9.96\%$$

The process can be continued as tabulated below:

iteration	x_i	$\lvert\varepsilon_a\rvert$
0	0.500000	
1	0.649637	23.0339%
2	0.721524	9.9632%
3	0.750901	3.9123%
4	0.762097	1.4691%
5	0.766248	0.5418%
6	0.767772	0.1984%
7	0.768329	0.0725%
8	0.768532	0.0265%
9	0.768606	0.0097%

Thus, after nine iterations, the root is estimated to be 0.768606 with an approximate error of 0.0097%.

6.2 (a) The function can be set up for fixed-point iteration by solving it for x in two different ways. First, it can be solved for the linear x,

$$x_{i+1} = \frac{0.9x_i^2 - 2.5}{1.7}$$

Using an initial guess of 5, the first iteration yields

$$x_1 = \frac{0.9(5)^2 - 2.5}{1.7} = 11.76$$

$$|\varepsilon_a| = \left|\frac{11.76 - 5}{11.76}\right| \times 100\% = 57.5\%$$

Second iteration:

$$x_1 = \frac{0.9(11.76)^2 - 2.5}{1.7} = 71.8$$

$$|\varepsilon_a| = \left|\frac{71.8 - 11.76}{71.8}\right| \times 100\% = 83.6\%$$

Clearly, this solution is diverging.

An alternative is to solve for the second-order *x*,

$$x_{i+1} = \sqrt{\frac{1.7x_i + 2.5}{0.9}}$$

Using an initial guess of 5, the first iteration yields

$$x_{i+1} = \sqrt{\frac{1.7(5) + 2.5}{0.9}} = 3.496$$

$$|\varepsilon_a| = \left|\frac{3.496 - 5}{3.496}\right| \times 100\% = 43.0\%$$

Second iteration:

$$x_{i+1} = \sqrt{\frac{1.7(3.496) + 2.5}{0.9}} = 3.0629$$

$$|\varepsilon_a| = \left|\frac{3.0629 - 3.496}{3.0629}\right| \times 100\% = 14.14\%$$

This version is converging. All the iterations can be tabulated as

iteration	x_i	$\|\varepsilon_a\|$
0	5.000000	
1	3.496029	43.0194%
2	3.062905	14.1410%
3	2.926306	4.6680%
4	2.881882	1.5415%
5	2.867287	0.5090%

6	2.862475	0.1681%
7	2.860887	0.0555%
8	2.860363	0.0183%
9	2.860190	0.0061%

Thus, after 9 iterations, the root estimate is 2.860190 with an approximate error of 0.0061%. The result can be checked by substituting it back into the original function,

$$f(2.860190) = -0.9(2.860190)^2 + 1.7(2.860190) + 2.5 = -0.000294$$

(b) The formula for Newton-Raphson is

$$x_{i+1} = x_i - \frac{-0.9x_i^2 + 1.7x_i + 2.5}{-1.8x_i + 1.7}$$

Using an initial guess of 5, the first iteration yields

$$x_{i+1} = 5 - \frac{-0.9(5)^2 + 1.7(5) + 2.5}{-1.8(5) + 1.7} = 3.424658$$

$$|\varepsilon_a| = \left|\frac{3.424658 - 5}{3.424658}\right| \times 100\% = 46.0\%$$

Second iteration:

$$x_{i+1} = 3.424658 - \frac{-0.9(3.424658)^2 + 1.7(3.424658) + 2.5}{-1.8(3.424658) + 1.7} = 2.924357$$

$$|\varepsilon_a| = \left|\frac{2.924357 - 3.424658}{2.924357}\right| \times 100\% = 17.1\%$$

The process can be continued as tabulated below:

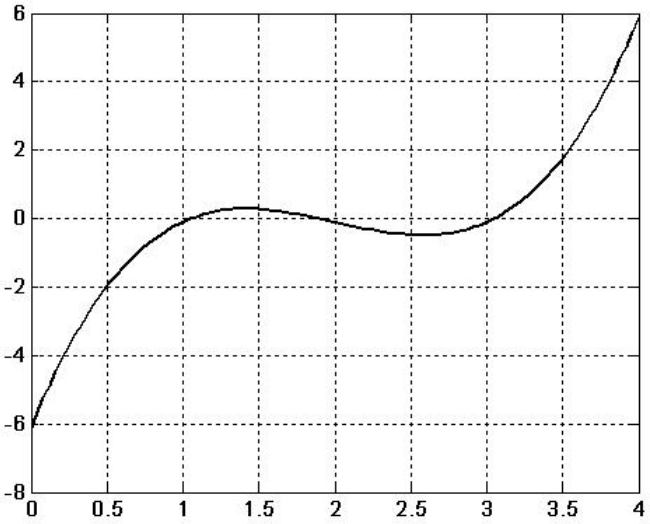
iteration	x_i	$f(x_i)$	$f'(x_i)$	$\lvert\varepsilon_a\rvert$
0	5	–11.5	–7.3	
1	3.424658	–2.23353	–4.46438	46.0000%
2	2.924357	–0.22527	–3.56384	17.1081%
3	2.861147	–0.00360	–3.45006	2.2093%
4	2.860105	–9.8E–07	–3.44819	0.0364%
5	2.860104	–7.2E–14	–3.44819	0.0000%

After 5 iterations, the root estimate is **2.860104** with an approximate error of 0.0000%. The result can be checked by substituting it back into the original function,

$$f(2.860104) = -0.9(2.860104)^2 + 1.7(2.860104) + 2.5 = -7.2 \times 10^{-14}$$

6.3 (a)

```
>> x = linspace(0,4);
>> y = x.^3-6*x.^2+11*x-6.1;
>> plot(x,y)
>> grid
```



Estimates are approximately 1.05, 1.9 and 3.05.

(b) The formula for Newton-Raphson is

$$x_{i+1} = x_i - \frac{x_i^3 - 6x_i^2 + 11x_i - 6.1}{3x_i^2 - 12x_i + 11}$$

Using an initial guess of 3.5, the first iteration yields

$$x_1 = 3.5 - \frac{(3.5)^3 - 6(3.5)^2 + 11(3.5) - 6.1}{3(3.5)^2 - 12(3.5) + 11} = 3.191304$$

$$|\varepsilon_a| = \left|\frac{3.191304 - 3.5}{3.191304}\right| \times 100\% = 9.673\%$$

Second iteration:

$$x_2 = 3.191304 - \frac{(3.191304)^3 - 6(3.191304)^2 + 11(3.191304) - 6.1}{3(3.191304)^2 - 12(3.191304) + 11} = 3.068699$$

$$|\varepsilon_a| = \left|\frac{3.068699 - 3.191304}{3.068699}\right| \times 100\% = 3.995\%$$

Third iteration:

$$x_3 = 3.068699 - \frac{(3.068699)^3 - 6(3.068699)^2 + 11(3.068699) - 6.1}{3(3.068699)^2 - 12(3.068699) + 11} = 3.047317$$

$$|\varepsilon_a| = \left|\frac{3.047317 - 3.068699}{3.047317}\right| \times 100\% = 0.702\%$$

(c) For the secant method, the first iteration:

$x_{-1} = 2.5$ $\quad f(x_{-1}) = -0.475$
$x_0 = 3.5$ $\quad f(x_0) = 1.775$

$$x_1 = 3.5 - \frac{1.775(2.5 - 3.5)}{-0.475 - 1.775} = 2.711111$$

$$|\varepsilon_a| = \left|\frac{2.711111 - 3.5}{2.711111}\right| \times 100\% = 29.098\%$$

Second iteration:

$x_0 = 3.5$ $\quad f(x_0) = 1.775$
$x_1 = 2.711111$ $\quad f(x_1) = -0.45152$

$$x_2 = 2.711111 - \frac{-0.45152(3.5 - 2.711111)}{1.775 - (-0.45152)} = 2.871091$$

$$|\varepsilon_a| = \left|\frac{2.871091 - 2.711111}{2.871091}\right| \times 100\% = 5.572\%$$

Third iteration:

$x_1 = 2.711111$ $\quad f(x_1) = -0.45152$
$x_2 = 2.871091$ $\quad f(x_2) = -0.31011$

$$x_3 = 2.871091 - \frac{-0.31011(2.711111 - 2.871091)}{-0.45152 - (-0.31011)} = 3.221923$$

$$|\varepsilon_a| = \left|\frac{3.221923 - 2.871091}{3.221923}\right| \times 100\% = 10.889\%$$

(d) For the modified secant method, the first iteration:

$x_0 = 3.5$ $\quad f(x_0) = 1.775$
$x_0 + \delta x_0 = 3.57$ $\quad f(x_0 + \delta x_0) = 2.199893$

$$x_1 = 3.5 - \frac{0.02(3.5)1.775}{2.199893 - 1.775} = 3.207573$$

$$|\varepsilon_a| = \left|\frac{3.207573 - 3.5}{3.207573}\right| \times 100\% = 9.117\%$$

Second iteration:

$x_1 = 3.207573$ $\quad f(x_1) = 0.453351$
$x_1 + \delta x_1 = 3.271725$ $\quad f(x_1 + \delta x_1) = 0.685016$

$$x_2 = 3.207573 - \frac{0.02(3.207573)0.453351}{0.685016 - 0.453351} = 3.082034$$

$$|\varepsilon_a| = \left|\frac{3.082034 - 3.207573}{3.082034}\right| \times 100\% = 4.073\%$$

Third iteration:

$x_2 = 3.082034$ $\quad f(x_2) = 0.084809$
$x_2 + \delta x_2 = 3.143675$ $\quad f(x_2 + \delta x_2) = 0.252242$

$$x_3 = 3.082034 - \frac{0.02(3.082034)0.084809}{0.252242 - 0.084809} = 3.050812$$

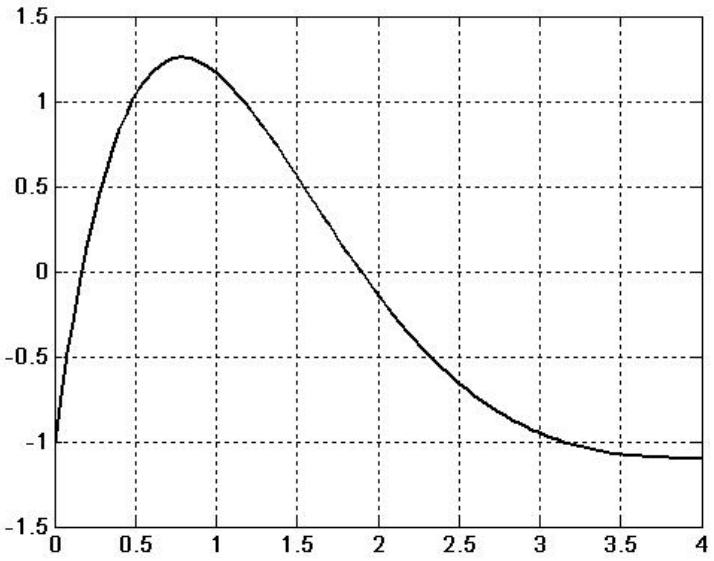
$$|\varepsilon_a| = \left|\frac{3.050812 - 3.082034}{3.050812}\right| \times 100\% = 1.023\%$$

(e)

```
>> a = [1 -6 11 -6.1]

a =
    1.0000   -6.0000   11.0000   -6.1000

>> roots(a)

ans =
    3.0467
    1.8990
    1.0544
```

6.4 (a)

```
>> x = linspace(0,4);
>> y = 7*sin(x).*exp(-x)-1;
>> plot(x,y)
>> grid
```



The lowest positive root seems to be at approximately 0.2.

(b) The formula for Newton-Raphson is

$$x_{i+1} = x_i - \frac{7\sin(x_i)e^{-x_i} - 1}{7e^{-x_i}(\cos(x_i) - \sin(x_i))}$$

Using an initial guess of 3.5, the first iteration yields

$$x_1 = 0.3 - \frac{7\sin(0.3)e^{-0.3} - 1}{7e^{-0.3}(\cos(0.3) - \sin(0.3))} = 0.3 - \frac{0.532487}{3.421627} = 0.144376$$

$$|\varepsilon_a| = \left|\frac{0.144376 - 0.3}{0.144376}\right| \times 100\% = 107.8\%$$

Second iteration:

$$x_2 = 0.144376 - \frac{7\sin(0.144376)e^{-0.144376} - 1}{7e^{-0.144376}(\cos(0.144376) - \sin(0.144376))} = 0.144376 - \frac{-0.12827}{5.124168} = 0.169409$$

$$|\varepsilon_a| = \left|\frac{0.169409 - 0.144376}{0.169409}\right| \times 100\% = 14.776\%$$

Third iteration:

$$x_1 = 0.169409 - \frac{7\sin(0.169409)e^{-0.169409} - 1}{7e^{-0.169409}(\cos(0.169409) - \sin(0.169409))} = 0.169409 - \frac{-0.00372}{4.828278} = 0.170179$$

$$|\varepsilon_a| = \left|\frac{0.170179 - 0.169409}{0.170179}\right| \times 100\% = 0.453\%$$

(c) For the secant method, the first iteration:

$x_{-1} = 0.4$ $\quad f(x_{-1}) = 0.827244$
$x_0 = 0.3$ $\quad f(x_0) = 0.532487$

$$x_1 = 0.3 - \frac{0.532487(0.4 - 0.3)}{0.827244 - 0.532487} = 0.119347$$

$$|\varepsilon_a| = \left|\frac{0.119347 - 0.3}{0.119347}\right| \times 100\% = 151.4\%$$

Second iteration:

$x_0 = 0.3$ $\quad f(x_0) = 0.532487$
$x_1 = 0.119347$ $\quad f(x_1) = -0.26032$

$$x_2 = 0.119347 - \frac{-0.26032(0.3 - 0.119347)}{0.532487 - (-0.26032)} = 0.178664$$

$$|\varepsilon_a| = \left|\frac{0.178664 - 0.119347}{0.178664}\right| \times 100\% = 33.2\%$$

Third iteration:

$x_1 = 0.119347$ $\quad f(x_1) = -0.26032$
$x_2 = 0.178664$ $\quad f(x_2) = 0.04047$

$$x_3 = 0.178664 - \frac{0.04047(0.119347 - 0.178664)}{-0.26032 - 0.04047} = 0.170683$$

$$|\varepsilon_a| = \left|\frac{0.170683 - 0.178664}{0.170683}\right| \times 100\% = 4.68\%$$

(d) For the modified secant method, the first iteration:

$x_0 = 0.3$ $\quad f(x_0) = 0.532487$
$x_0 + \delta x_0 = 0.303$ $\quad f(x_0 + \delta x_0) = 0.542708$

$$x_1 = 0.3 - \frac{0.01(0.3)0.532487}{0.542708 - 0.532487} = 0.143698$$

$$|\varepsilon_a| = \left|\frac{0.143698 - 0.3}{0.143698}\right| \times 100\% = 108.8\%$$

Second iteration:

$x_1 = 0.143698$ $\quad f(x_1) = -0.13175$
$x_1 + \delta x_1 = 0.145135$ $\quad f(x_1 + \delta x_1) = -0.12439$

$$x_2 = 0.143698 - \frac{0.02(0.143698)(-0.13175)}{-0.12439 - (-0.13175)} = 0.169412$$

$$|\varepsilon_a| = \left|\frac{0.169412 - 0.143698}{0.169412}\right| \times 100\% = 15.18\%$$

Third iteration:

$x_2 = 0.169412$ $\quad f(x_2) = -0.00371$
$x_2 + \delta x_2 = 0.171106$ $\quad f(x_2 + \delta x_2) = 0.004456$

$$x_3 = 0.169412 - \frac{0.02(0.169412)(-0.00371)}{0.004456 - (-0.00371)} = 0.170181$$

$$|\varepsilon_a| = \left|\frac{0.170181 - 0.169412}{0.170181}\right| \times 100\% = 0.452\%$$

6.5 **(a)** The formula for Newton-Raphson is

$$x_{i+1} = x_i - \frac{x_i^5 - 16.05x_i^4 + 88.75x_i^3 - 192.0375x_i^2 + 116.35x_i + 31.6875}{5x_i^4 - 64.2x_i^3 + 266.25x_i^2 - 384.075x_i + 116.35}$$

Using an initial guess of 0.5825, the first iteration yields

$$x_1 = 0.5825 - \frac{50.06217}{-29.1466} = 2.300098$$

$$|\varepsilon_a| = \left|\frac{2.300098 - 0.5825}{2.300098}\right| \times 100\% = 74.675\%$$

Second iteration

$$x_1 = 2.300098 - \frac{-21.546}{0.245468} = 90.07506$$

$$|\varepsilon_a| = \left|\frac{90.07506 - 2.300098}{90.07506}\right| \times 100\% = 97.446\%$$

Thus, the result seems to be diverging. However, the computation eventually settles down and converges (at a very slow rate) on a root at $x = 6.5$. The iterations can be summarized as

iteration	x_i	$f(x_i)$	$f'(x_i)$	$\lvert\varepsilon_a\rvert$
0	0.582500	50.06217	−29.1466	
1	2.300098	−21.546	0.245468	74.675%
2	90.07506	4.94E+09	2.84E+08	97.446%
3	72.71520	1.62E+09	1.16E+08	23.874%
4	58.83059	5.3E+08	47720880	23.601%
5	47.72701	1.74E+08	19552115	23.265%
6	38.84927	56852563	8012160	22.852%
7	31.75349	18616305	3284098	22.346%
8	26.08487	6093455	1346654	21.731%
9	21.55998	1993247	552546.3	20.987%
10	17.95260	651370.2	226941	20.094%
11	15.08238	212524.6	93356.59	19.030%
12	12.80590	69164.94	38502.41	17.777%
13	11.00952	22415.54	15946.36	16.317%
14	9.603832	7213.396	6652.03	14.637%
15	8.519442	2292.246	2810.851	12.728%
16	7.703943	710.9841	1217.675	10.585%
17	7.120057	209.2913	556.1668	8.201%
18	6.743746	54.06896	286.406	5.580%
19	6.554962	9.644695	187.9363	2.880%
20	6.503643	0.597806	164.8912	0.789%
21	6.500017	0.00285	163.32	0.056%
22	6.5	6.58E−08	163.3125	0.000%

(b) For the modified secant method, the first iteration:

$x_0 = 0.5825$ $\qquad f(x_0) = 50.06217$
$x_0 + \delta x_0 = 0.611625$ $\qquad f(x_0 + \delta x_0) = 49.15724$

$$x_1 = 0.5825 - \frac{0.05(0.5825)50.06217}{49.15724 - 50.06217} = 2.193735$$

$$\left|\varepsilon_a\right| = \left|\frac{2.193735 - 0.5825}{2.193735}\right| \times 100\% = 73.447\%$$

Second iteration:

$x_1 = 2.193735$ $\qquad f(x_1) = -21.1969$
$x_1 + \delta x_1 = 2.303422$ $\qquad f(x_1 + \delta x_1) = -21.5448$

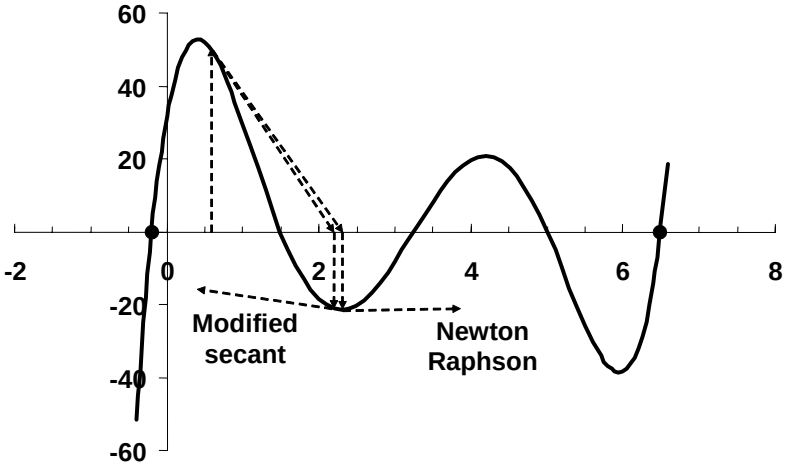
$$x_2 = 2.193735 - \frac{0.05(2.193735)(-21.1969)}{-21.5448 - (-21.1969)} = -4.48891$$

$$|\varepsilon_a| = \left|\frac{-4.48891 - 2.193735}{-4.48891}\right| \times 100\% = 148.87\%$$

Again, the result seems to be diverging. However, the computation eventually settles down and converges on a root at $x = -0.2$. The iterations can be summarized as

iteration	x_i	$x_i+\delta x_i$	$f(x_i)$	$f(x_i+\delta x_i)$	$\lvert\varepsilon_a\rvert$
0	0.5825	0.611625	50.06217	49.15724	
1	2.193735	2.303422	–21.1969	–21.5448	73.447%
2	–4.48891	–4.71336	–20727.5	–24323.6	148.870%
3	–3.19524	–3.355	–7201.94	–8330.4	40.487%
4	–2.17563	–2.28441	–2452.72	–2793.57	46.865%
5	–1.39285	–1.46249	–808.398	–906.957	56.200%
6	–0.82163	–0.86271	–250.462	–277.968	69.524%
7	–0.44756	–0.46994	–67.4718	–75.4163	83.579%
8	–0.25751	–0.27038	–12.5942	–15.6518	73.806%
9	–0.20447	–0.2147	–0.91903	–3.05726	25.936%
10	–0.20008	–0.21008	–0.01613	–2.08575	2.196%
11	–0.2	–0.21	–0.0002	–2.0686	0.039%
12	–0.2	–0.21	–2.4E–06	–2.06839	0.000%

Explanation of results: The results are explained by looking at a plot of the function. The guess of 0.5825 is located at a point where the function is relatively flat. Therefore, the first iteration results in a prediction of 2.3 for Newton-Raphson and 2.193 for the secant method. At these points the function is very flat and hence, the Newton-Raphson results in a very high value (90.075), whereas the modified false position goes in the opposite direction to a negative value (-4.49). Thereafter, the methods slowly converge on the nearest roots.



6.6

```
function root = secant(func,xrold,xr,es,maxit)
% secant(func,xrold,xr,es,maxit):
%   uses secant method to find the root of a function
% input:
%   func = name of function
%   xrold, xr = initial guesses
%   es = (optional) stopping criterion (%)
```

```
%   maxit = (optional) maximum allowable iterations
% output:
%   root = real root

% if necessary, assign default values
if nargin<5, maxit=50; end     %if maxit blank set to 50
if nargin<4, es=0.001; end     %if es blank set to 0.001

% Secant method
iter = 0;
while (1)
  xrn = xr - func(xr)*(xrold - xr)/(func(xrold) - func(xr));
  iter = iter + 1;
  if xrn ~= 0, ea = abs((xrn - xr)/xrn) * 100; end
  if ea <= es | iter >= maxit, break, end
  xrold = xr;
  xr = xrn;
end
root = xrn;
```

Test by solving Prob. 6.3:

```
>> secant(inline('x^3-6*x^2+11*x-6.1'),2.5,3.5)

ans =
    3.0467
```

6.7

```
function root = modsec(func,xr,delta,es,maxit)
% secant(func,xrold,xr,es,maxit):
%   uses the modified secant method
%   to find the root of a function
% input:
%   func = name of function
%   xr = initial guess
%   delta = perturbation fraction
%   es = (optional) stopping criterion (%)
%   maxit = (optional) maximum allowable iterations
% output:
%   root = real root

% if necessary, assign default values
if nargin<5, maxit=50; end     %if maxit blank set to 50
if nargin<4, es=0.001; end     %if es blank set to 0.001
if nargin<3, delta=1E-5; end   %if delta blank set to 0.00001

% Secant method
iter = 0;
while (1)
  xrold = xr;
  xr = xr - delta*xr*func(xr)/(func(xr+delta*xr)-func(xr));
  iter = iter + 1;
  if xr ~= 0, ea = abs((xr - xrold)/xr) * 100; end
  if ea <= es | iter >= maxit, break, end
end
root = xr;
```

Test by solving Prob. 6.3:

```
>> modsec(inline('x^3-6*x^2+11*x-6.1'),3.5,0.02)

ans =
    3.0467
```

6.8 The equation to be differentiated is

$$f(m) = \sqrt{\frac{gm}{c_d}} \tanh\left(\sqrt{\frac{gc_d}{m}} t\right) - v$$

Note that

$$\frac{d \tanh u}{dx} = \operatorname{sech}^2 u \frac{du}{dx}$$

Therefore, the derivative can be evaluated as

$$\frac{df(m)}{dm} = \sqrt{\frac{gm}{c_d}} \operatorname{sech}^2\left(\sqrt{\frac{gc_d}{m}} t\right)\left(-\frac{1}{2}\sqrt{\frac{m}{c_d g}}\right) t \frac{c_d g}{m^2} + \tanh\left(\sqrt{\frac{gc_d}{m}} t\right) \frac{1}{2}\sqrt{\frac{c_d}{gm}} \frac{g}{c_d}$$

The two terms can be reordered

$$\frac{df(m)}{dm} = \frac{1}{2}\sqrt{\frac{c_d}{gm}} \frac{g}{c_d} \tanh\left(\sqrt{\frac{gc_d}{m}} t\right) - \frac{1}{2}\sqrt{\frac{gm}{c_d}} \sqrt{\frac{m}{c_d g}} \frac{c_d g}{m^2} t \operatorname{sech}^2\left(\sqrt{\frac{gc_d}{m}} t\right)$$

The terms premultiplying the tanh and sech can be simplified to yield the final result

$$\frac{df(m)}{dm} = \frac{1}{2}\sqrt{\frac{g}{mc_d}} \tanh\left(\sqrt{\frac{gc_d}{m}} t\right) - \frac{g}{2m} t \operatorname{sech}^2\left(\sqrt{\frac{gc_d}{m}} t\right)$$

6.9 **(a)** The formula for Newton-Raphson is

$$x_{i+1} = x_i - \frac{-2 + 6x_i - 4x_i^2 + 0.5x_i^3}{6 - 8x_i + 1.5x_i^2}$$

Using an initial guess of 4.5, the iterations proceed as

iteration	x_i	$f(x_i)$	$f'(x_i)$	$\lvert\varepsilon_a\rvert$
0	4.5	−10.4375	0.375	
1	32.333330	12911.57	1315.5	86.082%
2	22.518380	3814.08	586.469	43.586%
3	16.014910	1121.912	262.5968	40.609%
4	11.742540	326.4795	118.8906	36.384%

5	8.996489	92.30526	55.43331	30.524%
6	7.331330	24.01802	27.97196	22.713%
7	6.472684	4.842169	17.06199	13.266%
8	6.188886	0.448386	13.94237	4.586%
9	6.156726	0.005448	13.6041	0.522%
10	6.156325	8.39E–07	13.59991	0.007%

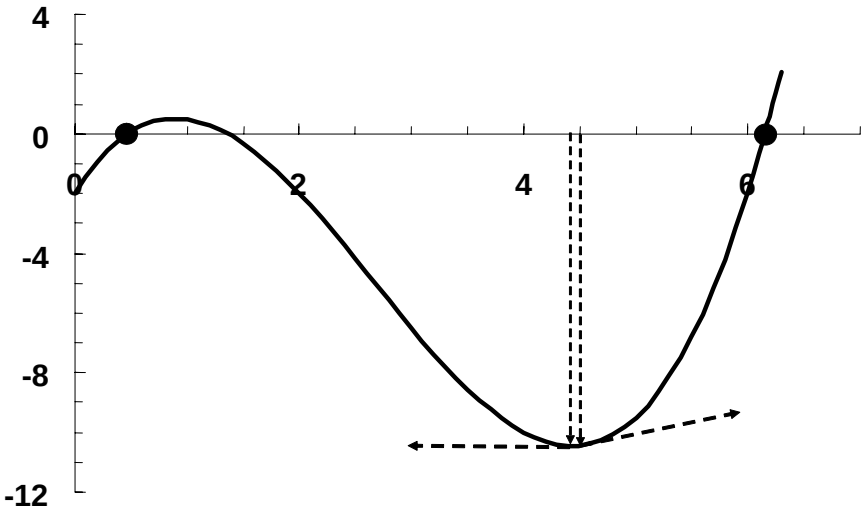
Thus, after an initial jump, the computation eventually settles down and converges on a root at $x = 6.156325$.

(b) Using an initial guess of 4.43, the iterations proceed as

iteration	x_i	$f(x_i)$	$f'(x_i)$	$\lvert\varepsilon_a\rvert$
0	4.43	-10.4504	-0.00265	
1	-3939.13	-3.1E+10	23306693	100.112%
2	-2625.2	-9.1E+09	10358532	50.051%
3	-1749.25	-2.7E+09	4603793	50.076%
4	-1165.28	-8E+08	2046132	50.114%
5	-775.964	-2.4E+08	909393.5	50.171%
.				
.				
.				
21	0.325261	-0.45441	3.556607	105.549%
22	0.453025	-0.05629	2.683645	28.203%
23	0.474	-0.00146	2.545015	4.425%
24	0.474572	-1.1E-06	2.541252	0.121%
25	0.474572	-5.9E-13	2.541249	0.000%

This time the solution jumps to an extremely large negative value The computation eventually converges at a very slow rate on a root at $x = 0.474572$.

Explanation of results: The results are explained by looking at a plot of the function. Both guesses are in a region where the function is relatively flat. Because the two guesses are on opposite sides of a minimum, both are sent to different regions that are far from the initial guesses. Thereafter, the methods slowly converge on the nearest roots.



6.10 The function to be evaluated is

$$x = \sqrt{a}$$

This equation can be squared and expressed as a roots problem,

$$f(x) = x^2 - a$$

The derivative of this function is

$$f'(x) = 2x$$

These functions can be substituted into the Newton-Raphson equation (Eq. 6.6),

$$x_{i+1} = x_i - \frac{x_i^2 - a}{2x_i}$$

which can be expressed as

$$x_{i+1} = \frac{x_i + a / x_i}{2}$$

6.11 (a) The formula for Newton-Raphson is

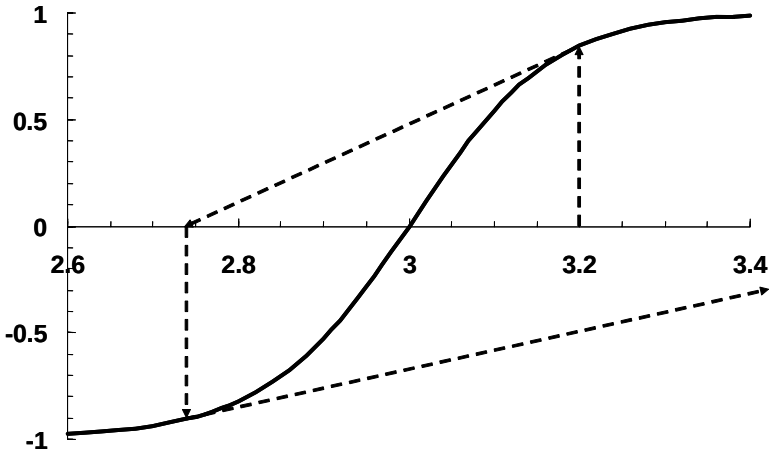
$$x_{i+1} = x_i - \frac{\tanh\left(x_i^2 - 9\right)}{2x_i \operatorname{sech}^2\left(x_i^2 - 9\right)}$$

Using an initial guess of 3.2, the iterations proceed as

iteration	x_i	$f(x_i)$	$f'(x_i)$	$\lvert\varepsilon_a\rvert$
0	3.2	0.845456	1.825311	
1	2.736816	–0.906910	0.971640	16.924%

2	3.670197	0.999738	0.003844	25.431%
3	–256.413			101.431%

(b) The solution diverges from its real root of $x = 3$. Due to the concavity of the slope, the next iteration will always diverge. The following graph illustrates how the divergence evolves.



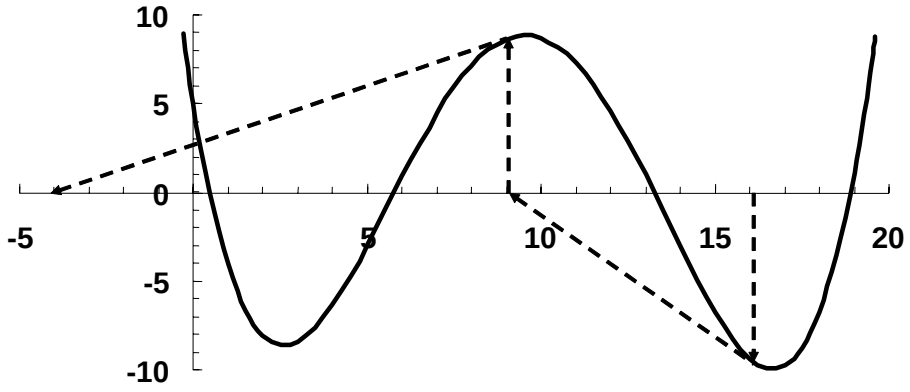
6.12 The formula for Newton-Raphson is

$$x_{i+1} = x_i - \frac{0.0074x_i^4 - 0.284x_i^3 + 3.355x_i^2 - 12.183x_i + 5}{0.0296x_i^3 - 0.852x_i^2 + 6.71x_i - 12.1832}$$

Using an initial guess of 16.15, the iterations proceed as

iteration	x_i	$f(x_i)$	$f'(x_i)$	$\lvert\varepsilon_a\rvert$
0	16.15	–9.57445	–1.35368	
1	9.077102	8.678763	0.662596	77.920%
2	–4.02101	128.6318	–54.864	325.742%
3	–1.67645	36.24995	–25.966	139.852%
4	–0.2804	8.686147	–14.1321	497.887%
5	0.334244	1.292213	–10.0343	183.890%
6	0.463023	0.050416	–9.25584	27.813%
7	0.46847	8.81E–05	–9.22351	1.163%
8	0.46848	2.7E–10	–9.22345	0.002%

As depicted below, the iterations involve regions of the curve that have flat slopes. Hence, the solution is cast far from the roots in the vicinity of the original guess.

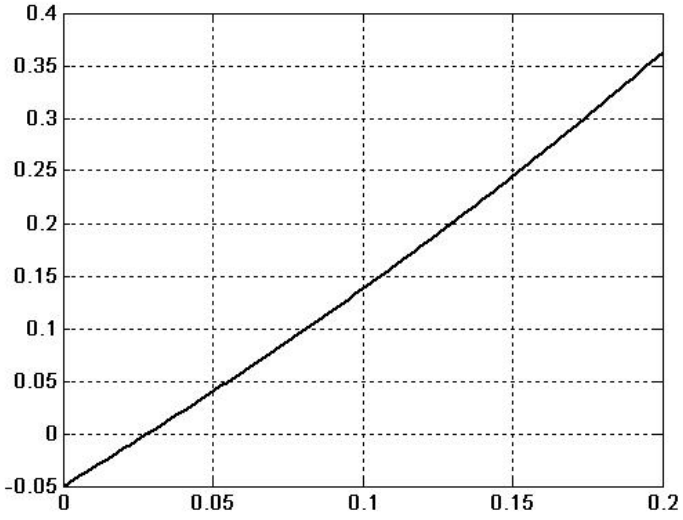


6.13 The solution involves determining the root of

$$f(x) = \frac{x}{1-x}\sqrt{\frac{6}{2+x}} - 0.05$$

MATLAB can be used to develop a plot that indicates that a root occurs in the vicinity of $x = 0.03$.

```
>> f = inline('x./(1-x).*sqrt(6./(2+x))-0.05')

f =
     Inline function:
     f(x) = x./(1-x).*sqrt(6./(2+x))-0.05

>> x = linspace(0,.2);
>> y = f(x);
>> plot(x,y)
```



The `fzero` function can then be used to find the root

```
>> format long
>> fzero(f,0.03)

ans =
   0.02824944114847
```

6.14 The coefficient, *a* and *b*, can be evaluated as

```
>> format long
>> R = 0.518;pc = 4600;Tc = 191;
>> a = 0.427*R^2*Tc^2.5/pc

a =
  12.55778319740302

>> b = 0.0866*R*Tc/pc

b =
   0.00186261539130
```

The solution, therefore, involves determining the root of

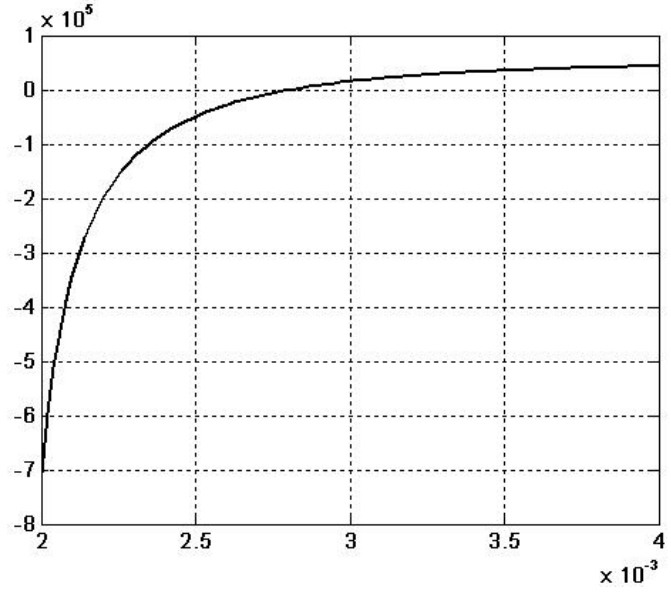
$$f(v) = 65{,}000 - \frac{0.518(233.15)}{v - 0.0018626} + \frac{12.557783}{v(v + 0.0018626)\sqrt{233.15}}$$

MATLAB can be used to generate a plot of the function and to solve for the root. One way to do this is to develop an M-file for the function,

```
function y = fvol(v)
R = 0.518;pc = 4600;Tc = 191;
a = 0.427*R^2*Tc^2.5/pc;
b = 0.0866*R*Tc/pc;
T = 273.15-40;p = 65000;
y = p - R*T./(v-b)+a./(v.*(v+b)*sqrt(T));
```

This function is saved as `fvol.m`. It can then be used to generate a plot

```
>> v = linspace(0.002,0.004);
>> fv = fvol(v);
>> plot(v,fv)
>> grid
```



Thus, a root is located at about 0.0028. The `fzero` function can be used to refine this estimate,

```
>> vroot = fzero('fvol',0.0028)

vroot =
   0.00280840865703
```

The mass of methane contained in the tank can be computed as

$$\text{mass} = \frac{V}{v} = \frac{3}{0.0028084} = 1068.317\ \text{m}^3$$

6.15 The function to be evaluated is

$$f(h) = V - \left[r^2 \cos^{-1}\left(\frac{r-h}{r}\right) - (r-h)\sqrt{2rh-h^2} \right] L$$

To use MATLAB to obtain a solution, the function can be written as an M-file

```
function y = fh(h,r,L,V)
y = V - (r^2*acos((r-h)/r)-(r-h)*sqrt(2*r*h-h^2))*L;
```

The `fzero` function can be used to determine the root as

```
>> format long
>> r = 2;L = 5;V = 8;
>> h = fzero('fh',0.5,[],r,L,V)

h =
   0.74001521805594
```

6.16 (a) The function to be evaluated is

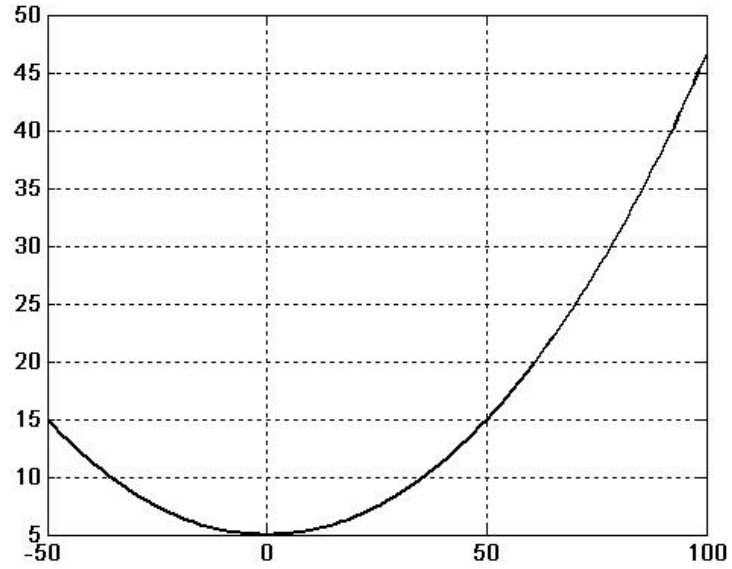
$$f(T_A) = 10 - \frac{T_A}{10}\cosh\left(\frac{500}{T_A}\right) + \frac{T_A}{10}$$

The solution can be obtained with the `fzero` function as

```
>> format long
>> TA = fzero(inline('10-x/10*cosh(500/x)+x/10'),1000)

TA =
    1.266324360399887e+003
```

(b) A plot of the cable can be generated as

```
>> x = linspace(-50,100);
>> w = 10;y0 = 5;
>> y = TA/w*cosh(w*x/TA) + y0 - TA/w;
>> plot(x,y),grid
```

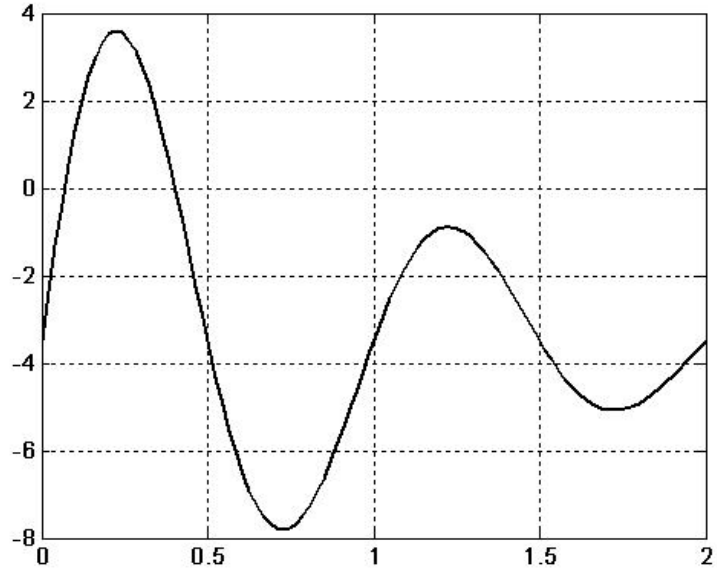


6.17 The function to be evaluated is

$$f(t) = 9e^{-t}\sin(2\pi t) - 3.5$$

A plot can be generated with MATLAB,

```
>> t = linspace(0,2);
>> y = 9*exp(-t) .* sin(2*pi*t) - 3.5;
>> plot(t,y),grid
```



Thus, there appear to be two roots at approximately 0.1 and 0.4. The `fzero` function can be used to obtain refined estimates,

```
>> t = fzero('9*exp(-x)*sin(2*pi*x)-3.5',[0 0.2])

t =
   0.06835432096851

>> t = fzero('9*exp(-x)*sin(2*pi*x)-3.5',[0.2 0.8])

t =
   0.40134369265980
```

6.18 The function to be evaluated is

$$f(\omega) = \frac{1}{Z} - \sqrt{\frac{1}{R^2} + \left(\omega C - \frac{1}{\omega L}\right)^2}$$

Substituting the parameter values yields

$$f(\omega) = 0.01 - \sqrt{\frac{1}{50625} + \left(0.6 \times 10^{-6}\omega - \frac{2}{\omega}\right)^2}$$

The `fzero` function can be used to determine the root as

```
>> fzero('0.01-sqrt(1/50625+(.6e-6*x-2./x).^2)',[1 1000])

ans =
  220.0202
```

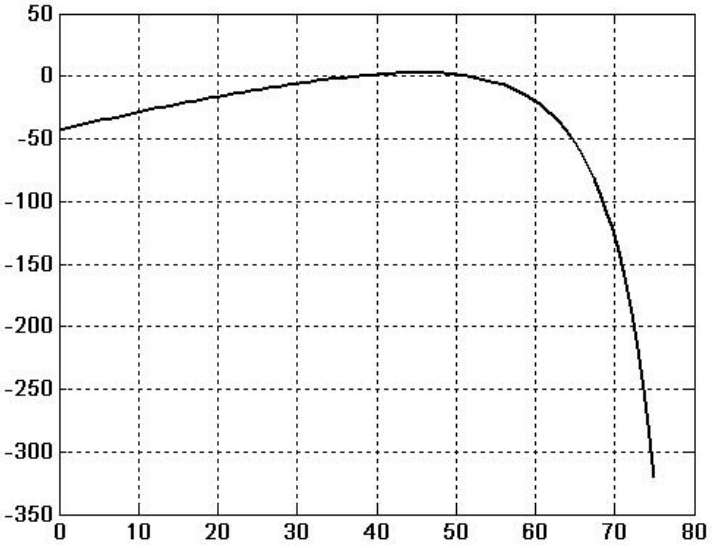
6.19 The `fzero` function can be used to determine the root as

```
>> format long
>> fzero('2*40*x^(5/2)/5+0.5*40000*x^2-95*9.8*x-95*9.8*0.43',1)

ans =
   0.16662477900186
```

6.20 If the height at which the throw leaves the right fielders arm is defined as $y = 0$, the y at 90 m will be –0.8. Therefore, the function to be evaluated is

$$f(\theta) = 0.8 + 90\tan\left(\frac{\pi}{180}\theta_0\right) - \frac{44.1}{\cos^2(\pi\theta_0/180)}$$

Note that the angle is expressed in degrees. First, MATLAB can be used to plot this function versus various angles.



Roots seem to occur at about 40^o and 50^o. These estimates can be refined with the `fzero` function,

```
>> theta = fzero('0.8+90*tan(pi*x/180)-44.1./cos(pi*x/180).^2',0)

theta =
   37.8380

>> theta = fzero('0.8+90*tan(pi*x/180)-44.1./cos(pi*x/180).^2',[40 60])

theta =
   51.6527
```

Thus, the right fielder can throw at two different angles to attain the same result.

6.21 The equation to be solved is

$$f(h) = \pi R h^2 - \left(\frac{\pi}{3}\right) h^3 - V$$

Because this equation is easy to differentiate, the Newton-Raphson is the best choice to achieve results efficiently. It can be formulated as

$$x_{i+1} = x_i - \frac{\pi R x_i^2 - \left(\frac{\pi}{3}\right) x_i^3 - V}{2\pi R x_i - \pi x_i^2}$$

or substituting the parameter values,

$$x_{i+1} = x_i - \frac{\pi(10) x_i^2 - \left(\frac{\pi}{3}\right) x_i^3 - 1000}{2\pi(10) x_i - \pi x_i^2}$$

The iterations can be summarized as

iteration	x_i	$f(x_i)$	$f'(x_i)$	$\lvert\varepsilon_a\rvert$
0	10	1094.395	314.1593	
1	6.516432	44.26917	276.0353	53.458%
2	6.356057	0.2858	272.4442	2.523%
3	6.355008	1.26E-05	272.4202	0.017%

Thus, after only three iterations, the root is determined to be 6.355008 with an approximate relative error of 0.017%.

6.22

```
>> r = [-2 6 1 -4 8];
>> a = poly(r)

a =
     1    -9   -20   204   208  -384

>> polyval(a,1)

ans =
     0

>> b = poly([-2 6])

b =
     1    -4   -12

>> [q,r] = deconv(a,b)

q =
     1    -5   -28    32
r =
     0     0     0     0     0     0
```

```
>> x = roots(q)

x =
    8.0000
   -4.0000
    1.0000

>> a = conv(q,b)

a =
     1    -9   -20   204   208  -384

>> x = roots(a)

x =
    8.0000
    6.0000
   -4.0000
   -2.0000
    1.0000

>> a = poly(x)

a =
    1.0000   -9.0000  -20.0000  204.0000  208.0000 -384.0000
```

6.23

```
>> a = [1 9 26 24];
>> r = roots(a)

r =
   -4.0000
   -3.0000
   -2.0000

>> a = [1 15 77 153 90];
>> r = roots(a)

r =
   -6.0000
   -5.0000
   -3.0000
   -1.0000
```

Therefore, the transfer function is

$$G(s) = \frac{(s+4)(s+3)(s+2)}{(s+6)(s+5)(s+3)(s+1)}$$

CHAPTER 7

7.1

```
>> 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
```

7.2 **(a)** $[A]$: 3×2 $[B]$: 3×3 $\{C\}$: 3×1 $[D]$: 2×4
$[E]$: 3×3 $[F]$: 2×3 $\lfloor G \rfloor$: 1×3

(b) square: $[B]$, $[E]$; column: $\{C\}$, row: $\lfloor G \rfloor$

(c) $a_{12} = 5$, $b_{23} = 6$, d_{32} = undefined, $e_{22} = 1$, $f_{12} = 0$, $g_{12} = 6$

(d) MATLAB can be used to perform the operations

(1) $[E]+[B]=\begin{bmatrix} 5 & 8 & 13 \\ 8 & 3 & 9 \\ 5 & 0 & 9 \end{bmatrix}$

(2) $[E]-[B]=\begin{bmatrix} 3 & -2 & 1 \\ -6 & 1 & 3 \\ -3 & 0 & -1 \end{bmatrix}$

(3) $[A]$ + $[F]$ = undefined

(4) $5[F]=\begin{bmatrix} 20 & 15 & 35 \\ 5 & 10 & 30 \\ 5 & 0 & 20 \end{bmatrix}$

(5) $[A] \times [B]$ = undefined

(6) $[B]\times[A]=\begin{bmatrix} 54 & 68 \\ 36 & 45 \\ 24 & 29 \end{bmatrix}$

(7) $[G] \times [C] = 56$

(8) $[C]^T = \begin{bmatrix} 2 & 6 & 1 \end{bmatrix}$

(9) $[D]^T = \begin{bmatrix} 5 & 2 \\ 4 & 1 \\ 3 & 7 \\ 6 & 5 \end{bmatrix}$

(10) $I \times [B] = \begin{bmatrix} 4 & 3 & 7 \\ 1 & 2 & 6 \\ 1 & 0 & 4 \end{bmatrix}$

7.3 The terms can be collected to give

$$\begin{bmatrix} -7 & 3 & 0 \\ 0 & 4 & 7 \\ -4 & 3 & -7 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 10 \\ -30 \\ 40 \end{Bmatrix}$$

Here is the MATLAB session:

```
>> A = [-7 3 0;0 4 7;-4 3 -7];
>> b = [10;-30;40];
>> x = A\b

x =
   -1.0811
    0.8108
   -4.7490

>> AT = A'

AT =

    -7     0    -4
     3     4     3
     0     7    -7

>> AI = inv(A)

AI =
   -0.1892    0.0811    0.0811
   -0.1081    0.1892    0.1892
    0.0618    0.0347   -0.1081
```

7.4

$$[X]\times[Y] = \begin{bmatrix} 23 & -8 \\ 55 & 56 \\ -17 & 24 \end{bmatrix}$$

$$[X]\times[Z] = \begin{bmatrix} 12 & 8 \\ -30 & 52 \\ -23 & 2 \end{bmatrix}$$

$$[Y]\times[Z] = \begin{bmatrix} 4 & 8 \\ -47 & 34 \end{bmatrix}$$

$$[Z]\times[Y] = \begin{bmatrix} 6 & 16 \\ -20 & 32 \end{bmatrix}$$

7.5 Terms can be combined to yield

$$2kx_1 - kx_2 = m_1 g$$

$$-kx_1 + 2kx_2 - kx_3 = m_2 g$$

$$-kx_2 + kx_3 = m_3 g$$

Substituting the parameter values

$$\begin{bmatrix} 20 & -10 & 0 \\ -10 & 20 & -10 \\ 0 & -10 & 10 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 19.62 \\ 29.43 \\ 24.525 \end{Bmatrix}$$

A MATLAB session can be used to obtain the solution for the displacements

```
>> K=[20 -10 0;-10 20 -10;0 -10 10];
>> m=[2;3;2.5];
>> mg=m*9.81;
>> x=K\mg

x =
    7.3575
   12.7530
   15.2055
```

7.6 The mass balances can be written as

$$(Q_{15} + Q_{12})c_1 - Q_{31}c_3 = Q_{01}c_{01}$$

$$-Q_{12}c_1 + (Q_{23} + Q_{24} + Q_{25})c_2 = 0$$

$$-Q_{23}c_2 + (Q_{31} + Q_{34})c_3 = Q_{03}c_{03}$$

$$-Q_{24}c_2 - Q_{34}c_3 + Q_{44}c_4 - Q_{54}c_5 = 0$$

$$-Q_{15}c_1 - Q_{25}c_2 + (Q_{54} + Q_{55})c_5 = 0$$

The parameters can be substituted and the result written in matrix form as

$$\begin{bmatrix} 6 & 0 & -1 & 0 & 0 \\ -3 & 3 & 0 & 0 & 0 \\ 0 & -1 & 9 & 0 & 0 \\ 0 & -1 & -8 & 11 & -2 \\ -3 & -1 & 0 & 0 & 4 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{Bmatrix} = \begin{Bmatrix} 50 \\ 0 \\ 160 \\ 0 \\ 0 \end{Bmatrix}$$

MATLAB can then be used to solve for the concentrations

```
>> Q = [6 0 -1 0 0;
-3 3 0 0 0;
0 -1 9 0 0;
0 -1 -8 11 -2;
-3 -1 0 0 4];
>> Qc = [50;0;160;0;0];
```

```
>> c = Q\Qc

c =
   11.5094
   11.5094
   19.0566
   16.9983
   11.5094
```

7.7 The problem can be written in matrix form as

$$\begin{bmatrix} 0.866 & 0 & -0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0.866 & 0 & 0 & 0 \\ -0.866 & -1 & 0 & -1 & 0 & 0 \\ -0.5 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & -0.866 & 0 & 0 & -1 \end{bmatrix} \begin{Bmatrix} F_1 \\ F_2 \\ F_3 \\ H_2 \\ V_2 \\ V_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ -1000 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix}$$

MATLAB can then be used to solve for the forces and reactions,

```
>> A = [0.866 0 -0.5 0 0 0;
0.5 0 0.866 0 0 0;
-0.866 -1 0 -1 0 0;
-0.5 0 0 0 -1 0;
0 1 0.5 0 0 0;
0 0 -0.866 0 0 -1]
>> b = [0 -1000 0 0 0 0]';
>> F = A\b

F =
 -500.0220
  433.0191
 -866.0381
   -0.0000
  250.0110
  749.9890
```

Therefore,

$F_1 = -500$ $F_2 = 433$ $F_3 = -866$
$H_2 = 0$ $V_2 = 250$ $V_3 = 750$

7.8 The problem can be written in matrix form as

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 10 & -10 & 0 & -15 & -5 \\ 5 & -10 & 0 & -20 & 0 & 0 \end{bmatrix} \begin{Bmatrix} i_{12} \\ i_{52} \\ i_{32} \\ i_{65} \\ i_{54} \\ i_{43} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 200 \end{Bmatrix}$$

MATLAB can then be used to solve for the currents,

```
>> A = [1 1 1 0 0 0 ;
0 -1 0 1 -1 0;
0 0 -1 0 0 1;
0 0 0 0 1 -1;
0 10 -10 0 -15 -5;
5 -10 0 -20 0 0];
>> b = [0 0 0 0 0 200]';
>> i = A\b

i =
    6.1538
   -4.6154
   -1.5385
   -6.1538
   -1.5385
   -1.5385
```

7.9

```
>> k1 = 10;k2 = 40;k3 = 40;k4 = 10;
>> m1 = 1;m2 = 1;m3 = 1;
>> km = [(1/m1)*(k2+k1), -(k2/m1),0;
-(k2/m2), (1/m2)*(k2+k3), -(k3/m2);
0, -(k3/m3),(1/m3)*(k3+k4)];
>> x = [0.05;0.04;0.03];
>> kmx = km*x

kmx =
    0.9000
    0.0000
   -0.1000
```

Therefore, $\ddot{x}_1 = -0.9$, $\ddot{x}_2 = 0$, and $\ddot{x}_3 = 0.1$ m/s^2.

CHAPTER 8

8.1 The flop counts for the tridiagonal algorithm in Fig. 8.6 can be summarized as

	Mult/Div	Add/Subtr	Total
Forward elimination	3(n – 1)	2(n – 1)	5(n – 1)
Back substitution	2n – 1	n – 1	3n – 2
Total	5n – 4	3n – 3	**8n – 7**

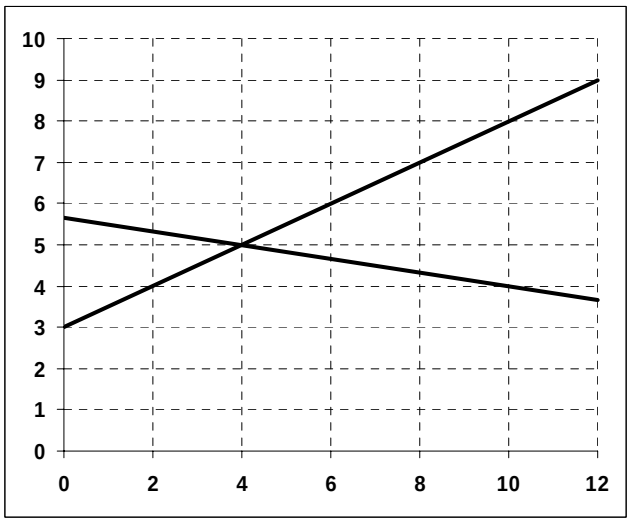
Thus, as n increases, the effort is much, much less than for a full matrix solved with Gauss elimination which is proportional to n^3.

8.2 The equations can be expressed in a format that is compatible with graphing x_2 versus x_1:

$$x_2 = 0.5x_1 + 3$$

$$x_2 = -\frac{1}{6}x_1 + \frac{34}{6}$$

which can be plotted as



Thus, the solution is $x_1 = 4$, $x_2 = 5$. The solution can be checked by substituting it back into the equations to give

$$4(4) - 8(5) = 16 - 40 = -24$$

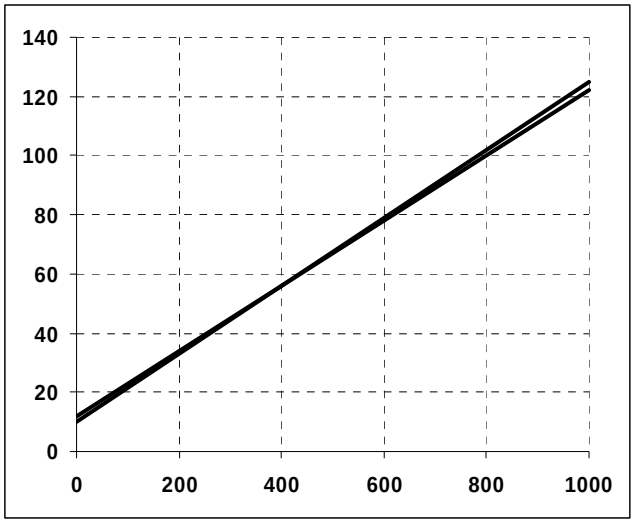
$$4 + 6(5) = 4 + 30 = 34$$

8.3 (a) The equations can be expressed in a format that is compatible with graphing x_2 versus x_1:

$$x_2 = 0.11x_1 + 12$$

$$x_2 = 0.114943x_1 + 10$$

which can be plotted as



Thus, the solution is approximately $x_1 = 400$, $x_2 = 60$. The solution can be checked by substituting it back into the equations to give

$$-1.1(400) + 10(60) = 160 \approx 120$$

$$-2(400) + 17.4(60) = 244 \approx 174$$

Therefore, the graphical solution is not very good.

(b) Because the lines have very similar slopes, you would expect that the system would be ill-conditioned

(c) The determinant can be computed as

$$\begin{vmatrix} -1.1 & 10 \\ -2 & 17.4 \end{vmatrix} = -1.1(17.2) - 10(-2) = -19.14 + 20 = 0.86$$

This result is relatively low suggesting that the solution is ill-conditioned.

8.4 (a) The determinant can be evaluated as

$$D = 0\begin{bmatrix} 2 & -1 \\ -2 & 0 \end{bmatrix} - (-3)\begin{bmatrix} 1 & -1 \\ 5 & 0 \end{bmatrix} + 7\begin{bmatrix} 1 & 2 \\ 5 & -2 \end{bmatrix}$$

$$D = 0(-2) + 3(5) + 7(-12) = -69$$

(b) Cramer's rule

$$x_1 = \frac{\begin{vmatrix} 2 & -3 & 7 \\ 3 & 2 & -1 \\ 2 & -2 & 0 \end{vmatrix}}{-69} = \frac{-68}{-69} = 0.9855$$

$$x_2 = \frac{\begin{vmatrix} 0 & 2 & 7 \\ 1 & 3 & -1 \\ 5 & 2 & 0 \end{vmatrix}}{-69} = \frac{-101}{-69} = 1.4638$$

$$x_3 = \frac{\begin{vmatrix} 0 & -3 & 2 \\ 1 & 2 & 3 \\ 5 & -2 & 2 \end{vmatrix}}{-69} = \frac{-63}{-69} = 0.9130$$

(c) Pivoting is necessary, so switch the first and third rows,

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

Multiply pivot row 1 by 1/5 and subtract the result from the second row to eliminate the a_{21} term.

$$\begin{aligned} 5x_1 - 2x_2 &= 2 \\ 2.4x_2 - x_3 &= 2.6 \\ -3x_2 + 7x_3 &= 2 \end{aligned}$$

Pivoting is necessary so switch the second and third row,

$$\begin{aligned} 5x_1 - 2x_2 &= 2 \\ -3x_2 + 7x_3 &= 2 \\ 2.4x_2 - x_3 &= 2.6 \end{aligned}$$

Multiply pivot row 2 by 2.4/(–3) and subtract the result from the third row to eliminate the a_{32} term.

$$\begin{aligned} 5x_1 - 2x_2 &= 2 \\ -3x_2 + 7x_3 &= 2 \\ 4.6x_3 &= 4.2 \end{aligned}$$

The solution can then be obtained by back substitution

$$x_3 = \frac{4.2}{4.6} = 0.913043$$

$$x_2 = \frac{2 - 7(0.913043)}{-3} = 1.463768$$

$$x_1 = \frac{2 + 2(1.463768)}{5} = 0.985507$$

(d)

$$-3(1.463768) + 7(0.913043) = 2$$

$$0.985507 + 2(1.463768) - (0.913043) = 3$$

$$5(0.985507) - 2(1.463768) = 2$$

8.5 Prob. 8.3:

```
>> A=[-1.1 10;-2 17.4];
>> det(A)

ans =
    0.8600
```

Prob. 8.4:

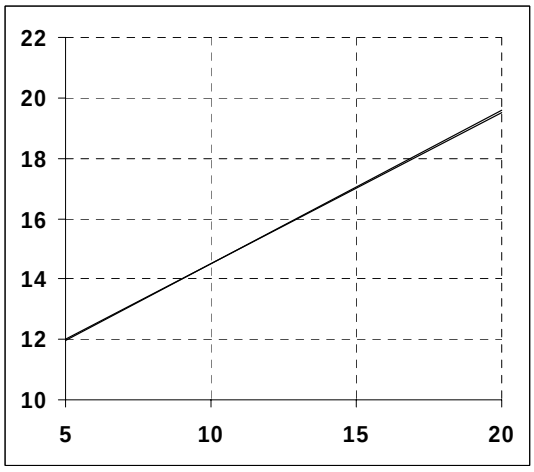
```
>> A=[0 -3 7;1 2 -1;5 -2 0];
>> det(A)

ans =
   -69
```

8.6 **(a)** The equations can be expressed in a format that is compatible with graphing x_2 versus x_1:

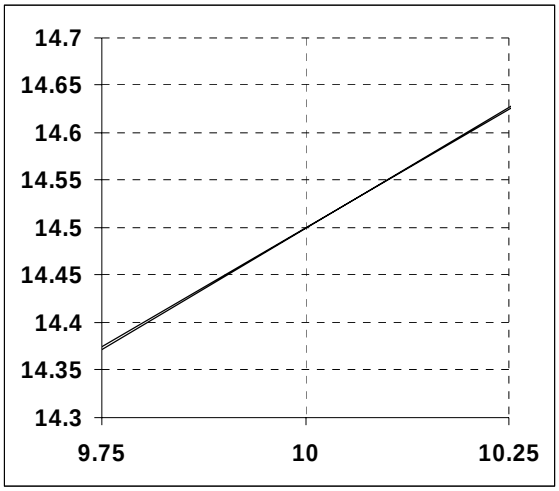
$$x_2 = 0.5x_1 + 9.5$$

$$x_2 = 0.51x_1 + 9.4$$

The resulting plot indicates that the intersection of the lines is difficult to detect:



Only when the plot is zoomed is it at all possible to discern that solution seems to lie at about x_1 = 14.5 and x_2 = 10.



(b) The determinant can be computed as

$$\begin{vmatrix} 0.5 & -1 \\ 1.02 & -2 \end{vmatrix} = 0.5(-2) - (-1)(1.02) = 0.02$$

which is close to zero.

(c) Because the lines have very similar slopes and the determinant is so small, you would expect that the system would be ill-conditioned

(d) Multiply the first equation by 1.02/0.5 and subtract the result from the second equation to eliminate the x_1 term from the second equation,

$$0.5x_1 - x_2 = -9.5$$

$$0.04x_2 = 0.58$$

The second equation can be solved for

$$x_2 = \frac{0.58}{0.04} = 14.5$$

This result can be substituted into the first equation which can be solved for

$$x_1 = \frac{-9.5 + 14.5}{0.5} = 10$$

(e) Multiply the first equation by 1.02/0.52 and subtract the result from the second equation to eliminate the x_1 term from the second equation,

$$0.52x_1 \quad - x_2 = -9.5$$

$$-0.03846x_2 = -0.16538$$

The second equation can be solved for

$$x_2 = \frac{-0.16538}{-0.03846} = 4.3$$

This result can be substituted into the first equation which can be solved for

$$x_1 = \frac{-9.5 + 4.3}{0.52} = -10$$

Interpretation: The fact that a slight change in one of the coefficients results in a radically different solution illustrates that this system is very ill-conditioned.

8.7 **(a)** Multiply the first equation by –3/10 and subtract the result from the second equation to eliminate the x_1 term from the second equation. Then, multiply the first equation by 1/10 and subtract the result from the third equation to eliminate the x_1 term from the third equation.

$$\begin{aligned} 10x_1 + 2x_2 \quad & - x_3 = 27 \\ -5.4x_2 & + 1.7x_3 = -53.4 \\ 0.8x_2 & + 5.1x_3 = -24.2 \end{aligned}$$

Multiply the second equation by 0.8/(–5.4) and subtract the result from the third equation to eliminate the x_2 term from the third equation,

$$\begin{aligned} 10x_1 + 2x_2 \quad & - x_3 = 27 \\ -5.4x_2 & + 1.7x_3 = -53.4 \\ 5.351852x_3 & = -32.11111 \end{aligned}$$

Back substitution can then be used to determine the unknowns

$$x_3 = \frac{-32.11111}{5.351852} = -6$$

$$x_2 = \frac{(-53.4 - 1.7(-6))}{-5.4} = 8$$

$$x_1 = \frac{(27 - 6 - 2(8))}{10} = 0.5$$

(b) Check:

$$10(0.5) + 2(8) - (-6) = 27$$

$$-3(0.5) - 6(8) + 2(-6) = -61.5$$

$$0.5 + 8 + 5(-6) = -21.5$$

8.8 **(a)** Pivoting is necessary, so switch the first and third rows,

$$-8x_1 + x_2 - 2x_3 = -20$$

$$-3x_1 - x_2 + 7x_3 = -34$$

$$2x_1 - 6x_2 - x_3 = -38$$

Multiply the first equation by –3/(–8) and subtract the result from the second equation to eliminate the a_{21} term from the second equation. Then, multiply the first equation by 2/(–8) and subtract the result from the third equation to eliminate the a_{31} term from the third equation.

$$-8x_1 + x_2 - 2x_3 = -20$$

$$-1.375x_2 + 7.75x_3 = -26.5$$

$$-5.75x_2 - 1.5x_3 = -43$$

Pivoting is necessary so switch the second and third row,

$$-8x_1 + x_2 - 2x_3 = -20$$

$$-5.75x_2 - 1.5x_3 = -43$$

$$-1.375x_2 + 7.75x_3 = -26.5$$

Multiply pivot row 2 by –1.375/(–5.75) and subtract the result from the third row to eliminate the a_{32} term.

$$-8x_1 + x_2 - 2x_3 = -20$$

$$-5.75x_2 - 1.5x_3 = -43$$

$$8.108696x_3 = -16.21739$$

The solution can then be obtained by back substitution

$$x_3 = \frac{-16.21739}{8.108696} = -2$$

$$x_2 = \frac{-43 + 1.5(-2)}{-5.75} = 8$$

$$x_1 = \frac{-20 + 2(-2) - 1(8)}{-8} = 4$$

(b) Check:

$$2(4) - 6(8) - (-2) = -38$$

$$-3(4) - (8) + 7(-2) = -34$$

$$-8(4) + (8) - 2(-2) = -20$$

8.9 Multiply the first equation by –0.4/0.8 and subtract the result from the second equation to eliminate the x_1 term from the second equation.

$$\begin{bmatrix} 0.8 & -0.4 & \\ & 0.6 & -0.4 \\ & -0.4 & 0.8 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 41 \\ 45.5 \\ 105 \end{Bmatrix}$$

Multiply pivot row 2 by –0.4/0.6 and subtract the result from the third row to eliminate the x_2 term.

$$\begin{bmatrix} 0.8 & -0.4 & \\ & 0.6 & -0.4 \\ & & 0.533333 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 41 \\ 45.5 \\ 135.3333 \end{Bmatrix}$$

The solution can then be obtained by back substitution

$$x_3 = \frac{135.3333}{0.533333} = 253.75$$

$$x_2 = \frac{45.5 - (-0.4)253.75}{0.6} = 245$$

$$x_1 = \frac{41 - (-0.4)245}{0.8} = 173.75$$

(b) Check:

$$0.8(173.75) - 0.4(245) = 41$$

$$-0.4(173.75) + 0.8(245) - 0.4(253.75) = 25$$

$$-0.4(245) + 0.8(253.75) = 105$$

8.10 The mass balances can be written as

$$Q_{21}c_2 + 400 = Q_{12}c_1 + Q_{13}c_1$$

$$Q_{12}c_1 = Q_{21}c_2 + Q_{23}c_2$$

$$Q_{13}c_1 + Q_{23}c_2 = Q_{33}c_3 + 200$$

or collecting terms

$$\begin{aligned}(Q_{12} + Q_{13})c_1 \quad & - Q_{21}c_2 \quad & = 400 \\ - Q_{12}c_1 + (Q_{21} + Q_{23})c_2 \quad & & = 0 \\ - Q_{13}c_1 \quad & - Q_{23}c_2 + Q_{33}c_3 & = 200\end{aligned}$$

Substituting the values for the flows and expressing in matrix form

$$\begin{bmatrix} 120 & -20 & 0 \\ -80 & 80 & 0 \\ -40 & -60 & 120 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ 0 \\ 200 \end{Bmatrix}$$

A solution can be obtained with MATLAB as

```
>> A = [120 -20 0;-80 80 0;-40 -60 120];
>> b = [400 0 200]';
>> c = a\b

c =
    4.0000
    4.0000
    5.0000
```

8.11 Equations for the amount of sand, fine gravel and coarse gravel can be written as

$$0.32x_1 + 0.25x_2 + 0.35x_3 = 6000$$

$$0.30x_1 + 0.40x_2 + 0.15x_3 = 5000$$

$$0.38x_1 + 0.35x_2 + 0.50x_3 = 8000$$

where x_i = the amount of gravel taken from pit *i*. MATLAB can be used to solve this system of equations for

```
>> A=[0.32 0.25 0.35;0.3 0.4 0.15;0.38 0.35 0.5];
>> b=[6000;5000;8000];
>> x=A\b

x =
  1.0e+003 *
    7.0000
    4.4000
    7.6000
```

Therefore, we take 7000, 4400 and 7600 m^3 from pits 1, 2 and 3 respectively.

8.12 Substituting the parameter values the heat-balance equations can be written for the four nodes as

$$
\begin{aligned}
-40 + 2.2T_1 - T_2 &= 4 \\
-T_1 + 2.2T_2 - T_3 &= 4 \\
-T_2 + 2.2T_3 - T_4 &= 4 \\
-T_3 + 2.2T_4 - 200 &= 4
\end{aligned}
$$

Collecting terms and expressing in matrix form

$$
\begin{bmatrix} 2.2 & -1 & 0 & 0 \\ -1 & 2.2 & -1 & 0 \\ 0 & -1 & 2.2 & -1 \\ 0 & 0 & -1 & 2.2 \end{bmatrix} \begin{Bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{Bmatrix} = \begin{Bmatrix} 44 \\ 4 \\ 4 \\ 204 \end{Bmatrix}
$$

The solution can be obtained with MATLAB as

```
>> A=[2.2 -1 0 0;-1 2.2 -1 0;0 -1 2.2 -1;0 0 -1 2.2]
>> b=[44 4 4 204]'
>> T=A\b

T =
   50.7866
   67.7306
   94.2206
  135.5548
```

CHAPTER 9

9.1 The flop counts for *LU* decomposition can be determined in a similar fashion as was done for Gauss elimination. The major difference is that the elimination is only implemented for the left-hand side coefficients. Thus, for every iteration of the inner loop, there are n multiplications/divisions and $n - 1$ addition/subtractions. The computations can be summarized as

Outer Loop k	Inner Loop i	Addition/Subtraction flops	Multiplication/Division flops
1	2, n	$(n-1)(n-1)$	$(n-1)n$
2	3, n	$(n-2)(n-2)$	$(n-2)(n-1)$
.	.		
.	.		
.	.		
k	$k+1$, n	$(n-k)(n-k)$	$(n-k)(n+1-k)$
.	.		
.	.		
.	.		
$n-1$	n, n	(1)(1)	(1)(2)

Therefore, the total addition/subtraction flops for elimination can be computed as

$$\sum_{k=1}^{n-1}(n-k)(n-k)=\sum_{k=1}^{n-1}\left[n^2-2nk+k^2\right]$$

Applying some of the relationships from Eq. (8.14) yields

$$\sum_{k=1}^{n-1}\left[n^2-2nk+k^2\right]=\frac{n^3}{3}-\frac{n^2}{2}+\frac{n}{6}$$

A similar analysis for the multiplication/division flops yields

$$\sum_{k=1}^{n-1}(n-k)(n+1-k)=\frac{n^3}{3}-\frac{n}{3}$$

$$\left[n^3+O(n^2)\right]-\left[n^3+O(n)\right]+\left[\frac{1}{3}n^3+O(n^2)\right]=\frac{n^3}{3}+O(n^2)$$

Summing these results gives

$$\frac{2n^3}{3}-\frac{n^2}{2}-\frac{n}{6}$$

For forward substitution, the numbers of multiplications and subtractions are the same and equal to

$$\sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} = \frac{n^2}{2} - \frac{n}{2}$$

Back substitution is the same as for Gauss elimination: $n^2/2 - n/2$ subtractions and $n^2/2 + n/2$ multiplications/divisions. The entire number of flops can be summarized as

	Mult/Div	Add/Subtr	Total
Forward elimination	$\frac{n^3}{3} - \frac{n}{3}$	$\frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6}$	$\frac{2n^3}{3} - \frac{n^2}{2} - \frac{n}{6}$
Forward substitution	$\frac{n^2}{2} - \frac{n}{2}$	$\frac{n^2}{2} - \frac{n}{2}$	$n^2 - n$
Back substitution	$\frac{n^2}{2} + \frac{n}{2}$	$\frac{n^2}{2} - \frac{n}{2}$	n^2
Total	$\frac{n^3}{3} + n^2 - \frac{n}{3}$	$\frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6}$	$\frac{2n^3}{3} + \frac{3n^2}{2} - \frac{7n}{6}$

The total number of flops is identical to that obtained with standard Gauss elimination.

9.2 Equation (9.6) is

$$[L]\{[U]\{x\} - \{d\}\} = [A]\{x\} - \{b\} \tag{9.6}$$

Matrix multiplication is distributive, so the left-hand side can be rewritten as

$$[L][U]\{x\} - [L]\{d\} = [A]\{x\} - \{b\}$$

Equating the terms that are multiplied by $\{x\}$ yields,

$$[L][U]\{x\} = [A]\{x\}$$

and, therefore, Eq. (9.7) follows

$$[L][U] = [A] \tag{9.7}$$

Equating the constant terms yields Eq. (9.8)

$$[L]\{d\} = \{b\} \tag{9.8}$$

9.3 The matrix to be evaluated is

$$\begin{bmatrix} 10 & 2 & -1 \\ -3 & -6 & 2 \\ 1 & 1 & 5 \end{bmatrix}$$

Multiply the first row by $f_{21} = -3/10 = -0.3$ and subtract the result from the second row to eliminate the a_{21} term. Then, multiply the first row by $f_{31} = 1/10 = 0.1$ and subtract the result from the third row to eliminate the a_{31} term. The result is

$$\begin{bmatrix} 10 & 2 & -1 \\ 0 & -5.4 & 1.7 \\ 0 & 0.8 & 5.1 \end{bmatrix}$$

Multiply the second row by $f_{32} = 0.8/(-5.4) = -0.148148$ and subtract the result from the third row to eliminate the a_{32} term.

$$\begin{bmatrix} 10 & 2 & -1 \\ 0 & -5.4 & 1.7 \\ 0 & 0 & 5.351852 \end{bmatrix}$$

Therefore, the *LU* decomposition is

$$[L]\{U\} = \begin{bmatrix} 1 & 0 & 0 \\ -0.3 & 1 & 0 \\ 0.1 & -0.148148 & 1 \end{bmatrix} \begin{bmatrix} 10 & 2 & -1 \\ 0 & -5.4 & 1.7 \\ 0 & 0 & 5.351852 \end{bmatrix}$$

Multiplying [*L*] and [*U*] yields the original matrix as verified by the following MATLAB session,

```
>> L = [1 0 0;-0.3 1 0;0.1 -0.148148 1];
>> U = [10 2 -1;0 -5.4 1.7;0 0 5.351852];
>> A = L*U

A =
   10.0000    2.0000   -1.0000
   -3.0000   -6.0000    2.0000
    1.0000    1.0000    5.0000
```

9.4 The *LU* decomposition can be computed as

$$[L]\{U\} = \begin{bmatrix} 1 & 0 & 0 \\ -0.3 & 1 & 0 \\ 0.1 & -0.148148 & 1 \end{bmatrix} \begin{bmatrix} 10 & 2 & -1 \\ 0 & -5.4 & 1.7 \\ 0 & 0 & 5.351852 \end{bmatrix}$$

Forward substitution:

$$\{d\} = \begin{bmatrix} 1 & 0 & 0 \\ -0.3 & 1 & 0 \\ 0.1 & -0.148148 & 1 \end{bmatrix} \begin{Bmatrix} 27 \\ -61.5 \\ -21.5 \end{Bmatrix}$$

$$d_1 = 27$$

$$d_2 = -61.5 + 0.3(27) = -53.4$$

$$d_3 = -21.5 - 0.1(27) - (-0.148148)(-53.4) = -32.11111$$

Back substitution:

$$\{x\} = \begin{bmatrix} 10 & 2 & -1 \\ 0 & -5.4 & 1.7 \\ 0 & 0 & 5.351852 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 27 \\ -53.5 \\ -32.11111 \end{Bmatrix}$$

$$x_3 = \frac{-32.11111}{5.351852} = -6$$

$$x_2 = \frac{-53.4 - 1.7(-6)}{-5.4} = 8$$

$$x_1 = \frac{27 - 2(8) - (-1)(-6)}{10} = 0.5$$

For the alternative right-hand-side vector, forward substitution is implemented as

$$\{d\} = \begin{bmatrix} 1 & 0 & 0 \\ -0.3 & 1 & 0 \\ 0.1 & -0.148148 & 1 \end{bmatrix} \begin{Bmatrix} 12 \\ 18 \\ -6 \end{Bmatrix}$$

$$d_1 = 12$$

$$d_2 = 18 + 0.3(12) = 21.6$$

$$d_3 = -6 - 0.1(12) - (-0.148148)(18) = -4$$

Back substitution:

$$\{x\} = \begin{bmatrix} 10 & 2 & -1 \\ 0 & -5.4 & 1.7 \\ 0 & 0 & 5.351852 \end{bmatrix} \begin{Bmatrix} 12 \\ 21.6 \\ -4 \end{Bmatrix}$$

$$x_3 = \frac{-4}{5.351852} = -0.747405$$

$$x_2 = \frac{21.6 - 1.7(-0.747405)}{-5.4} = -4.235294$$

$$x_1 = \frac{12 - 2(-4.235294) - (-1)(-0.747405)}{10} = 1.972318$$

9.5 The system can be written in matrix form as

$$[A] = \begin{bmatrix} 2 & -6 & -1 \\ -3 & -1 & 7 \\ -8 & 1 & -2 \end{bmatrix} \qquad \{b\} = \begin{Bmatrix} -38 \\ -34 \\ -20 \end{Bmatrix}$$

Partial pivot:

$$[A]=\begin{bmatrix}-8 & 1 & -2\\ -3 & -1 & 7\\ 2 & -6 & -1\end{bmatrix} \qquad \{b\}=\begin{Bmatrix}-20\\ -34\\ -38\end{Bmatrix}$$

Forward eliminate

$f_{21} = -3/(-8) = 0.375$ $\qquad$ $f_{31} = 2/(-8) = -0.25$

$$[A]=\begin{bmatrix}-8 & 1 & -2\\ 0 & -1.375 & 7.75\\ 0 & -5.75 & -1.5\end{bmatrix}$$

Pivot again

$$[A]=\begin{bmatrix}-8 & 1 & -2\\ 0 & -5.75 & -1.5\\ 0 & -1.375 & 7.75\end{bmatrix} \qquad \{b\}=\begin{Bmatrix}-20\\ -38\\ -34\end{Bmatrix}$$

$f_{21} = -0.25$ $\qquad$ $f_{31} = 0.375$

Forward eliminate

$f_{32} = -1.375/(-5.75) = 0.23913$

$$[A]=\begin{bmatrix}-8 & 1 & -2\\ 0 & -5.75 & -1.5\\ 0 & 0 & 8.108696\end{bmatrix}$$

Therefore, the LU decomposition is

$$[L]\{U\}=\begin{bmatrix}1 & 0 & 0\\ -0.25 & 1 & 0\\ 0.375 & 0.23913 & 1\end{bmatrix}\begin{bmatrix}-8 & 1 & -2\\ 0 & -5.75 & -1.5\\ 0 & 0 & 8.108696\end{bmatrix}$$

Forward elimination

$$\{d\}=\begin{bmatrix}1 & 0 & 0\\ -0.25 & 1 & 0\\ 0.375 & 0.23913 & 1\end{bmatrix}\begin{Bmatrix}-20\\ -38\\ -34\end{Bmatrix}$$

$d_1 = -20$

$d_2 = -38 - (-0.25)(-20) = -43$

$d_3 = -34 - 0.375(-20) - 0.23913(-43) = -16.21739$

Back substitution:

$$\begin{bmatrix} -8 & 1 & -2 \\ 0 & -5.75 & -1.5 \\ 0 & 0 & 8.108696 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} -20 \\ -43 \\ -16.21739 \end{Bmatrix}$$

$$x_3 = \frac{-16.21739}{8.108696} = -2$$

$$x_2 = \frac{-43-(-1.5)(-2)}{-5.75} = 8$$

$$x_1 = \frac{-20-1(8)-(-2)(-2)}{-8} = 4$$

9.6 Here is an M-file to generate the *LU* decomposition without pivoting

```
function [L, U] = LUNaive(A)
% LUNaive(A):
%   LU decomposition without pivoting.
% input:
%   A = coefficient matrix
% output:
%   L = lower triangular matrix
%   U = upper triangular matrix

[m,n] = size(A);
if m~=n, error('Matrix A must be square'); end
L = eye(n);
U = A;
% forward elimination
for k = 1:n-1
  for i = k+1:n
    L(i,k) = U(i,k)/U(k,k);
    U(i,k) = 0;
    U(i,k+1:n) = U(i,k+1:n)-L(i,k)*U(k,k+1:n);
  end
end
```

Test with Prob. 9.3

```
>> A = [10 2 -1;-3 -6 2;1 1 5];
>> [L,U] = LUnaive(A)

L =
    1.0000         0         0
   -0.3000    1.0000         0
    0.1000   -0.1481    1.0000

U =
   10.0000    2.0000   -1.0000
         0   -5.4000    1.7000
         0         0    5.3519
```

Verification that $[L][U] = [A]$.

```
>> L*U

ans =
   10.0000    2.0000   -1.0000
   -3.0000   -6.0000    2.0000
    1.0000    1.0000    5.0000
```

Check using the `lu` function,

```
>> [L,U]=lu(A)

L =
    1.0000         0         0
   -0.3000    1.0000         0
    0.1000   -0.1481    1.0000

U =
   10.0000    2.0000   -1.0000
         0   -5.4000    1.7000
         0         0    5.3519
```

9.7 The result of Example 9.4 can be substituted into Eq. (9.14) to give

$$[A] = [U]^T [U] = \begin{bmatrix} 2.44949 & & \\ 6.123724 & 4.1833 & \\ 22.45366 & 20.9165 & 6.110101 \end{bmatrix} \begin{bmatrix} 2.44949 & 6.123724 & 22.45366 \\ & 4.1833 & 20.9165 \\ & & 6.110101 \end{bmatrix}$$

The multiplication can be implemented as in

$$a_{11} = 2.44949^2 = 6.000001$$

$$a_{12} = 6.123724 \times 2.44949 = 15$$

$$a_{13} = 22.45366 \times 2.44949 = 55.00002$$

$$a_{21} = 2.44949 \times 6.123724 = 15$$

$$a_{22} = 6.123724^2 + 4.1833^2 = 54.99999$$

$$a_{22} = 22.45366 \times 6.123724^2 + 20.9165 \times 4.1833 = 225$$

$$a_{31} = 2.44949 \times 22.45366 = 55.00002$$

$$a_{32} = 6.123724 \times 22.45366 + 4.1833 \times 20.9165 = 225$$

$$a_{33} = 22.45366^2 + 20.9165^2 + 6.110101^2 = 979.0002$$

9.8 **(a)** For the first row ($i = 1$), Eq. (9.15) is employed to compute

$$u_{11} = \sqrt{a_{11}} = \sqrt{8} = 2.828427$$

Then, Eq. (9.16) can be used to determine

$$u_{12} = \frac{a_{12}}{u_{11}} = \frac{20}{2.828427} = 7.071068$$

$$u_{13} = \frac{a_{13}}{u_{11}} = \frac{15}{2.828427} = 5.303301$$

For the second row ($i = 2$),

$$u_{22} = \sqrt{a_{22} - u_{12}^2} = \sqrt{80 - (7.071068)^2} = 5.477226$$

$$u_{23} = \frac{a_{23} - u_{12}u_{13}}{u_{22}} = \frac{50 - 7.071068(5.303301)}{5.477226} = 2.282177$$

For the third row ($i = 3$),

$$u_{33} = \sqrt{a_{33} - u_{13}^2 - u_{23}^2} = \sqrt{60 - (5.303301)^2 - (2.282177)^2} = 5.163978$$

Thus, the Cholesky decomposition yields

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

The validity of this decomposition can be verified by substituting it and its transpose into Eq. (9.14) to see if their product yields the original matrix [*A*]. This is left for an exercise.

(b)

```
>> A = [8 20 15;20 80 50;15 50 60];
>> U = chol(A)

U =
    2.8284    7.0711    5.3033
         0    5.4772    2.2822
         0         0    5.1640
```

(c) The solution can be obtained by hand or by MATLAB. Using MATLAB:

```
>> b = [50;250;100];
>> d=U'\b

d =
```

```
   17.6777
   22.8218
   -8.8756

>> x=U\d

x =
   -2.7344
    4.8828
   -1.7187
```

9.9 Here is an M-file to generate the Cholesky decomposition without pivoting

```
function U = cholesky(A)
% cholesky(A):
%   cholesky decomposition without pivoting.
% input:
%   A = coefficient matrix
% output:
%   U = upper triangular matrix
[m,n] = size(A);
if m~=n, error('Matrix A must be square'); end
for i = 1:n
  s = 0;
  for k = 1:i-1
    s = s + U(k, i) ^ 2;
  end
  U(i, i) = sqrt(A(i, i) - s);
  for j = i + 1:n
    s = 0;
    for k = 1:i-1
      s = s + U(k, i) * U(k, j);
    end
    U(i, j) = (A(i, j) - s) / U(i, i);
  end
end
```

Test with Prob. 9.8

```
>> A = [8 20 15;20 80 50;15 50 60];
>> cholesky(A)

ans =
    2.8284    7.0711    5.3033
         0    5.4772    2.2822
         0         0    5.1640
```

Check with the `chol` function

```
>> U = chol(A)

U =
    2.8284    7.0711    5.3033
         0    5.4772    2.2822
         0         0    5.1640
```

CHAPTER 10

10.1 First, compute the *LU* decomposition The matrix to be evaluated is

$$\begin{bmatrix} 10 & 2 & -1 \\ -3 & -6 & 2 \\ 1 & 1 & 5 \end{bmatrix}$$

Multiply the first row by $f_{21} = -3/10 = -0.3$ and subtract the result from the second row to eliminate the a_{21} term. Then, multiply the first row by $f_{31} = 1/10 = 0.1$ and subtract the result from the third row to eliminate the a_{31} term. The result is

$$\begin{bmatrix} 10 & 2 & -1 \\ 0 & -5.4 & 1.7 \\ 0 & 0.8 & 5.1 \end{bmatrix}$$

Multiply the second row by $f_{32} = 0.8/(-5.4) = -0.148148$ and subtract the result from the third row to eliminate the a_{32} term.

$$\begin{bmatrix} 10 & 2 & -1 \\ 0 & -5.4 & 1.7 \\ 0 & 0 & 5.351852 \end{bmatrix}$$

Therefore, the *LU* decomposition is

$$[L]\{U\} = \begin{bmatrix} 1 & 0 & 0 \\ -0.3 & 1 & 0 \\ 0.1 & -0.148148 & 1 \end{bmatrix} \begin{bmatrix} 10 & 2 & -1 \\ 0 & -5.4 & 1.7 \\ 0 & 0 & 5.351852 \end{bmatrix}$$

The first column of the matrix inverse can be determined by performing the forward-substitution solution procedure with a unit vector (with 1 in the first row) as the right-hand-side vector. Thus, the lower-triangular system, can be set up as,

$$\begin{bmatrix} 1 & 0 & 0 \\ -0.3 & 1 & 0 \\ 0.1 & -0.148148 & 1 \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix}$$

and solved with forward substitution for $\{d\}^T = \lfloor 1 \quad 0.3 \quad -0.055556 \rfloor$. This vector can then be used as the right-hand side of the upper triangular system,

$$\begin{bmatrix} 10 & 2 & -1 \\ 0 & -5.4 & 1.7 \\ 0 & 0 & 5.351852 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 0.3 \\ -0.055556 \end{Bmatrix}$$

which can be solved by back substitution for the first column of the matrix inverse,

$$[A]^{-1} = \begin{bmatrix} 0.110727 & 0 & 0 \\ -0.058824 & 0 & 0 \\ -0.010381 & 0 & 0 \end{bmatrix}$$

To determine the second column, Eq. (9.8) is formulated as

$$\begin{bmatrix} 1 & 0 & 0 \\ -0.3 & 1 & 0 \\ 0.1 & -0.148148 & 1 \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 1 \\ 0 \end{Bmatrix}$$

This can be solved with forward substitution for $\{d\}^T = \lfloor 0 \;\; 1 \;\; 0.148148 \rfloor$, and the results are used with $[U]$ to determine $\{x\}$ by back substitution to generate the second column of the matrix inverse,

$$[A]^{-1} = \begin{bmatrix} 0.110727 & 0.038062 & 0 \\ -0.058824 & -0.176471 & 0 \\ -0.010381 & 0.027682 & 0 \end{bmatrix}$$

Finally, the same procedures can be implemented with $\{b\}^T = \lfloor 0 \;\; 0 \;\; 1 \rfloor$ to solve for $\{d\}^T = \lfloor 0 \;\; 0 \;\; 1 \rfloor$, and the results are used with $[U]$ to determine $\{x\}$ by back substitution to generate the third column of the matrix inverse,

$$[A]^{-1} = \begin{bmatrix} 0.110727 & 0.038062 & 0.00692 \\ -0.058824 & -0.176471 & 0.058824 \\ -0.010381 & 0.027682 & 0.186851 \end{bmatrix}$$

This result can be checked by multiplying it times the original matrix to give the identity matrix. The following MATLAB session can be used to implement this check,

```
>> A = [10 2 -1;-3 -6 2;1 1 5];
>> AI = [0.110727 0.038062 0.00692;
-0.058824 -0.176471 0.058824;
-0.010381 0.027682 0.186851];
>> A*AI

ans =
    1.0000   -0.0000   -0.0000
    0.0000    1.0000   -0.0000
   -0.0000    0.0000    1.0000
```

10.2 The system can be written in matrix form as

$$[A] = \begin{bmatrix} -8 & 1 & -2 \\ 2 & -6 & -1 \\ -3 & -1 & 7 \end{bmatrix} \qquad \{b\} = \begin{Bmatrix} -38 \\ -34 \\ -20 \end{Bmatrix}$$

Forward eliminate

$f_{21} = 2/(-8) = -0.25$ $\qquad$ $f_{31} = -3/(-8) = 0.375$

$$[A]=\begin{bmatrix} -8 & 1 & -2 \\ 0 & -5.75 & -1.5 \\ 0 & -1.375 & 7.75 \end{bmatrix}$$

Forward eliminate

f_{32} = –1.375/(–5.75) = 0.23913

$$[A]=\begin{bmatrix} -8 & 1 & -2 \\ 0 & -5.75 & -1.5 \\ 0 & 0 & 8.108696 \end{bmatrix}$$

Therefore, the *LU* decomposition is

$$[L]\{U\}=\begin{bmatrix} 1 & 0 & 0 \\ -0.25 & 1 & 0 \\ 0.375 & 0.23913 & 1 \end{bmatrix}\begin{bmatrix} -8 & 1 & -2 \\ 0 & -5.75 & -1.5 \\ 0 & 0 & 8.108696 \end{bmatrix}$$

The first column of the matrix inverse can be determined by performing the forward-substitution solution procedure with a unit vector (with 1 in the first row) as the right-hand-side vector. Thus, the lower-triangular system, can be set up as,

$$\begin{bmatrix} 1 & 0 & 0 \\ -0.25 & 1 & 0 \\ 0.375 & 0.23913 & 1 \end{bmatrix}\begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix}=\begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix}$$

and solved with forward substitution for $\{d\}^T = \lfloor 1 \quad 0.25 \quad -0.434783 \rfloor$. This vector can then be used as the right-hand side of the upper triangular system,

$$\begin{bmatrix} -8 & 1 & -2 \\ 0 & -5.75 & -1.5 \\ 0 & 0 & 8.108696 \end{bmatrix}\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}=\begin{Bmatrix} 1 \\ 0.25 \\ -0.434783 \end{Bmatrix}$$

which can be solved by back substitution for the first column of the matrix inverse,

$$[A]^{-1}=\begin{bmatrix} -0.115282 & 0 & 0 \\ -0.029491 & 0 & 0 \\ -0.053619 & 0 & 0 \end{bmatrix}$$

To determine the second column, Eq. (9.8) is formulated as

$$\begin{bmatrix} 1 & 0 & 0 \\ -0.25 & 1 & 0 \\ 0.375 & 0.23913 & 1 \end{bmatrix}\begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix}=\begin{Bmatrix} 0 \\ 1 \\ 0 \end{Bmatrix}$$

This can be solved with forward substitution for $\{d\}^T = \lfloor 0 \quad 1 \quad -0.23913 \rfloor$, and the results are used with [*U*] to determine $\{x\}$ by back substitution to generate the second column of the matrix inverse,

$$[A]^{-1} = \begin{bmatrix} -0.115282 & -0.013405 & 0 \\ -0.029491 & -0.16622 & 0 \\ -0.053619 & -0.029491 & 0 \end{bmatrix}$$

Finally, the same procedures can be implemented with $\{b\}^T = \lfloor 0 \;\; 0 \;\; 1 \rfloor$ to solve for $\{d\}^T = \lfloor 0 \;\; 0 \;\; 1 \rfloor$, and the results are used with $[U]$ to determine $\{x\}$ by back substitution to generate the third column of the matrix inverse,

$$[A]^{-1} = \begin{bmatrix} -0.115282 & -0.013405 & -0.034853 \\ -0.029491 & -0.16622 & -0.032172 \\ -0.053619 & -0.029491 & 0.123324 \end{bmatrix}$$

10.3 The following solution is generated with MATLAB.

(a)

```
>> A = [15 -3 -1;-3 18 -6;-4 -1 12];
>> format long
>> AI = inv(A)

AI =
   0.07253886010363   0.01278065630397   0.01243523316062
   0.02072538860104   0.06079447322971   0.03212435233161
   0.02590673575130   0.00932642487047   0.09015544041451
```

(b)

```
>> b = [3800 1200 2350]';
>> format short
>> c = AI*b

c =
  320.2073
  227.2021
  321.5026
```

(c) The impact of a load to reactor 3 on the concentration of reactor 1 is specified by the element $a_{13}^{-1} = 0.0124352$. Therefore, the increase in the mass input to reactor 3 needed to induce a 10 g/m^3 rise in the concentration of reactor 1 can be computed as

$$\Delta b_3 = \frac{10}{0.0124352} = 804.1667 \frac{\text{g}}{\text{d}}$$

(d) The decrease in the concentration of the third reactor will be

$$\Delta c_3 = 0.0259067(500) + 0.009326(250) = 12.9534 + 2.3316 = 15.285 \frac{\text{g}}{\text{m}^3}$$

10.4 The mass balances can be written and the result written in matrix form as

$$\begin{bmatrix} 6 & 0 & -1 & 0 & 0 \\ -3 & 3 & 0 & 0 & 0 \\ 0 & -1 & 9 & 0 & 0 \\ 0 & -1 & -8 & 11 & -2 \\ -3 & -1 & 0 & 0 & 4 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{Bmatrix} = \begin{Bmatrix} Q_{01}c_{01} \\ 0 \\ Q_{03}c_{03} \\ 0 \\ 0 \end{Bmatrix}$$

MATLAB can then be used to determine the matrix inverse

```
>> Q = [6 0 -1 0 0;-3 3 0 0 0;0 -1 9 0 0;0 -1 -8 11 -2;-3 -1 0 0 4];
>> inv(Q)

ans =
    0.1698    0.0063    0.0189         0         0
    0.1698    0.3396    0.0189         0         0
    0.0189    0.0377    0.1132         0         0
    0.0600    0.0746    0.0875    0.0909    0.0455
    0.1698    0.0896    0.0189         0    0.2500
```

The concentration in reactor 5 can be computed using the elements of the matrix inverse as in,

$$c_5 = a_{51}^{-1}Q_{01}c_{01} + a_{53}^{-1}Q_{03}c_{03} = 0.1698(5)20 + 0.0189(8)50 = 16.981 + 7.547 = 24.528$$

10.5 The problem can be written in matrix form as

$$\begin{bmatrix} 0.866 & 0 & -0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0.866 & 0 & 0 & 0 \\ -0.866 & -1 & 0 & -1 & 0 & 0 \\ -0.5 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & -0.866 & 0 & 0 & -1 \end{bmatrix} \begin{Bmatrix} F_1 \\ F_2 \\ F_3 \\ H_2 \\ V_2 \\ V_3 \end{Bmatrix} = \begin{Bmatrix} F_{1,h} \\ F_{1,v} \\ F_{2,h} \\ F_{2,v} \\ F_{3,h} \\ F_{3,v} \end{Bmatrix}$$

MATLAB can then be used to solve for the matrix inverse,

```
>> A = [0.866 0 -0.5 0 0 0;
0.5 0 0.866 0 0 0;
-0.866 -1 0 -1 0 0;
-0.5 0 0 0 -1 0;
0 1 0.5 0 0 0;
0 0 -0.866 0 0 -1];
>> AI = inv(A)

AI =
    0.8660    0.5000         0         0         0         0
    0.2500   -0.4330         0         0    1.0000         0
   -0.5000    0.8660         0         0         0         0
   -1.0000    0.0000   -1.0000         0   -1.0000         0
   -0.4330   -0.2500         0   -1.0000         0         0
    0.4330   -0.7500         0         0         0   -1.0000
```

The forces in the members resulting from the two forces can be computed using the elements of the matrix inverse as in,

$$F_1 = a_{12}^{-1}F_{1,v} + a_{15}^{-1}F_{3,h} = 0.5(-2000) + 0(-500) = -1000 + 0 = -1000$$

$$F_2 = a_{22}^{-1}F_{1,v} + a_{25}^{-1}F_{3,h} = -0.433(-2000) + 1(-500) = 866 - 500 = 366$$

$$F_3 = a_{32}^{-1}F_{1,v} + a_{35}^{-1}F_{3,h} = 0.866(-2000) + 0(-500) = -1732 + 0 = -1732$$

10.6 The matrix can be scaled by dividing each row by the element with the largest absolute value

```
>> A = [8/(-10) 2/(-10) 1;1 1/(-9) 3/(-9);1 -1/15 6/15]

A =

   -0.8000   -0.2000    1.0000
    1.0000   -0.1111   -0.3333
    1.0000   -0.0667    0.4000
```

MATLAB can then be used to determine each of the norms,

```
>> norm(A,'fro')

ans =
    1.9920

>> norm(A,1)

ans =
    2.8000

>> norm(A,inf)

ans =
     2
```

10.7 Prob. 10.2:

```
>> A = [-8 1 -2;2 -6 -1;-3 -1 7];
>> norm(A,'fro')

ans =
    13

>> norm(A,inf)

ans =
    11
```

Prob. 10.3:

```
>> A = [15 -3 -1;-3 18 -6;-4 -1 12]
```

```
>> norm(A,'fro')

ans =
   27.6586

>> norm(A,inf)

ans =
    27
```

10.8 **(a)** Spectral norm

```
>> A = [1 4 9 16;4 9 16 25;9 16 25 36;16 25 36 49];
>> cond(A)

ans =
  8.8963e+016
```

(b) Row-sum norm

```
>> cond(A,inf)

Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 3.037487e-019.
(Type "warning off MATLAB:nearlySingularMatrix" to suppress this
warning.)
> In cond at 45

ans =
  3.2922e+018
```

10.9 **(a)** The matrix to be evaluated is

$$\begin{bmatrix} 16 & 4 & 1 \\ 4 & 2 & 1 \\ 49 & 7 & 1 \end{bmatrix}$$

The row-sum norm of this matrix is 49 + 7 + 1 = 57. The inverse is

$$\begin{bmatrix} -0.1667 & 0.1 & 0.0667 \\ 1.5 & -1.1 & -0.4 \\ -2.3333 & 2.8 & 0.5333 \end{bmatrix}$$

The row-sum norm of the inverse is $|-2.3333| + 2.8 + 0.5333 = 5.6667$. Therefore, the condition number is

Cond[A] = 57(5.6667) = 323

This can be verified with MATLAB,

```
>> A = [16 4 1;4 2 1;49 7 1];
>> cond(A,inf)
```

```
ans =
  323.0000
```

(b) Spectral norm:

```
>> A = [16 4 1;4 2 1;49 7 1];
>> cond(A)

ans =
  216.1294
```

Frobenius norm:

```
>> cond(A,'fro')

ans =
  217.4843
```

10.10 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.

10.11 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.

CHAPTER 11

11.1 (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	ε_a	maximum ε_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: $x_2 = 1.2(62) - 0.2(0) = 74.4$

$$x_3 = \frac{105 + 0.4x_2}{0.8} = \frac{105 + 0.4(62)}{0.8} = 168.45$$

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

Second iteration:

$$x_1 = \frac{41 + 0.4(62)}{0.8} = 88.45$$

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	ε_a	maximum ε_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.

11.2 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	ε_a	maximum ε_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.

11.3 The first iteration can be implemented as

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

$$x_2 = \frac{-61.5 + 3x_1 - 2x_3}{-6} = \frac{-61.5 + 3(0) - 2(0)}{-6} = 10.25$$

$$x_3 = \frac{-21.5 - x_1 - x_2}{5} = \frac{-21.5 - 0 - 0}{5} = -4.3$$

Second iteration:

$$x_1 = \frac{27 - 2(10.25) - 4.3}{10} = 0.22$$

$$x_2 = \frac{-61.5 + 3(2.7) - 2(-4.3)}{-6} = 7.466667$$

$$x_3 = \frac{-21.5 - (2.7) - 10.25}{5} = -6.89$$

The error estimates can be computed as

$$\varepsilon_{a,1} = \left|\frac{0.22 - 2.7}{0.258}\right| \times 100\% = 1127\%$$

$$\varepsilon_{a,2} = \left|\frac{7.466667 - 10.25}{7.466667}\right| \times 100\% = 37.28\%$$

$$\varepsilon_{a,3} = \left|\frac{-6.89 - (-4.3)}{-6.89}\right| \times 100\% = 37.59\%$$

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	ε_a	maximum ε_a
1	x_1	2.7	100.00%	
	x_2	10.25	100.00%	
	x_3	-4.3	100.00%	100.00%
2	x_1	0.22	1127.27%	
	x_2	7.466667	37.28%	
	x_3	-6.89	37.59%	1127.27%
3	x_1	0.517667	57.50%	
	x_2	7.843333	4.80%	
	x_3	-5.83733	18.03%	57.50%
4	x_1	0.5476	5.47%	
	x_2	8.045389	2.51%	
	x_3	-5.9722	2.26%	5.47%
5	x_1	0.493702	10.92%	
	x_2	7.985467	0.75%	

	x_3	-6.0186	0.77%	10.92%
6	x_1	0.501047	1.47%	
	x_2	7.99695	0.14%	
	x_3	-5.99583	0.38%	1.47%

Thus, after 6 iterations, the maximum error is 1.47% and we arrive at the result: x_1 = 0.501047, x_2 = 7.99695 and x_3 = –5.99583.

11.4 The first iteration can be implemented as

$$c_1 = \frac{3800 + 3c_2 + c_3}{15} = \frac{3800 + 3(0) + 0}{15} = 253.3333$$

$$c_2 = \frac{1200 + 3c_1 + 6c_3}{18} = \frac{1200 + 3(253.3333) + 6(0)}{18} = 108.8889$$

$$c_3 = \frac{2350 + 4c_1 + c_2}{12} = \frac{2350 + 4(253.3333) + 108.8889}{12} = 289.3519$$

Second iteration:

$$c_1 = \frac{3800 + 3(108.889) + 289.3519}{15} = 294.4012$$

$$c_2 = \frac{1200 + 3(294.4012) + 6(289.3519)}{18} = 212.1842$$

$$c_3 = \frac{2350 + 4(294.4012) + 212.1842}{12} = 311.6491$$

The error estimates can be computed as

$$\varepsilon_{a,1} = \left|\frac{294.4012 - 253.3333}{294.4012}\right| \times 100\% = 13.95\%$$

$$\varepsilon_{a,2} = \left|\frac{212.1842 - 108.8889}{212.1842}\right| \times 100\% = 48.68\%$$

$$\varepsilon_{a,3} = \left|\frac{311.6491 - 289.3519}{311.6491}\right| \times 100\% = 7.15\%$$

The remainder of the calculation can be summarized as

iteration	unknown	value	ε_a	maximum ε_a
1	x_1	253.3333	100.00%	
	x_2	108.8889	100.00%	
	x_3	289.3519	100.00%	100.00%

2	x_1	294.4012	13.95%	
	x_2	212.1842	48.68%	
	x_3	311.6491	7.15%	48.68%
3	x_1	316.5468	7.00%	
	x_2	223.3075	4.98%	
	x_3	319.9579	2.60%	7.00%
4	x_1	319.3254	0.87%	
	x_2	226.5402	1.43%	
	x_3	321.1535	0.37%	1.43%
5	x_1	320.0516	0.23%	
	x_2	227.0598	0.23%	
	x_3	321.4388	0.09%	0.23%

Note that after several more iterations, we arrive at the result: $x_1 = 320.2073$, $x_2 = 227.2021$ and $x_3 = 321.5026$.

11.5 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	ε_a	maximum ε_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 λ = 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	ε_a	maximum ε_a
1	x_1	2.5	3	100.00%	
	x_2	7.3333333	8.8	100.00%	
	x_3	-2.314286	-2.777143	100.00%	100.000%
2	x_1	4.2942857	4.5531429	34.11%	
	x_2	8.3139048	8.2166857	7.10%	
	x_3	-1.731984	-1.522952	82.35%	82.353%
3	x_1	3.9078237	3.7787598	20.49%	
	x_2	7.8467453	7.7727572	5.71%	
	x_3	-2.12728	-2.248146	32.26%	32.257%
4	x_1	4.0336312	4.0846055	7.49%	
	x_2	8.0695595	8.12892	4.38%	
	x_3	-1.945323	-1.884759	19.28%	19.280%
5	x_1	3.9873047	3.9678445	2.94%	
	x_2	7.9700747	7.9383056	2.40%	

	x_3	-2.022594	-2.050162	8.07%	8.068%
6	x_1	4.0048286	4.0122254	1.11%	
	x_2	8.0124354	8.0272613	1.11%	
	x_3	-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 $x_3 = -1.979007$.

11.6 As ordered, none of the sets will converge. However, if Set 1 and 3 are reordered so that they are diagonally dominant, they will converge on the solution of (1, 1, 1).

Set 1:
$$\begin{aligned} 8x + 3y + z &= 12 \\ 2x + 4y - z &= 5 \\ -6x + 7z &= 1 \end{aligned}$$

Set 3:
$$\begin{aligned} 3x + y - z &= 3 \\ x + 4y - z &= 4 \\ x + y + 5z &= 7 \end{aligned}$$

Because it is not diagonally dominant, Set 2 will not converge on the correct solution of (1, 1, 1). However, it will also not diverge. Rather, it will oscillate. The way that this occurs depends on how the equations are ordered. For example, if they can be ordered as

$$\begin{aligned} -2x + 4y - 5z &= -3 \\ 2y - z &= 1 \\ -x + 3y + 5z &= 7 \end{aligned}$$

For this case, Gauss-Seidel iterations yields

iteration	unknown	value	ε_a	maximum ε_a
1	x_1	1.5	100.00%	
	x_2	0.5	100.00%	
	x_3	1.4	100.00%	100.00%
2	x_1	-1	250.00%	
	x_2	1.2	58.33%	
	x_3	0.48	191.67%	250.00%
3	x_1	2.7	137.04%	
	x_2	0.74	62.16%	
	x_3	1.496	67.91%	137.04%
4	x_1	-0.76	455.26%	
	x_2	1.248	40.71%	
	x_3	0.4992	199.68%	455.26%
5	x_1	2.748	127.66%	
	x_2	0.7496	66.49%	
	x_3	1.49984	66.72%	127.66%
6	x_1	-0.7504	466.20%	
	x_2	1.24992	40.03%	
	x_3	0.499968	199.99%	466.20%
7	x_1	2.74992	127.29%	

	x_2	0.749984	66.66%	
	x_3	1.499994	66.67%	127.29%
8	x_1	-0.75002	466.65%	
	x_2	1.249997	40.00%	
	x_3	0.499999	200.00%	466.65%

Alternatively, they can be ordered as

$$\begin{aligned} -x + 3y + 5z &= 7 \\ 2y - z &= 1 \\ -2x + 4y - 5z &= -3 \end{aligned}$$

For this case, Gauss-Seidel iterations yields

iteration	unknown	value	ε_a	maximum ε_a
1	x_1	-7	100.00%	
	x_2	0.5	100.00%	
	x_3	3.8	100.00%	100.00%
2	x_1	13.5	151.85%	
	x_2	2.4	79.17%	
	x_3	-2.88	231.94%	231.94%
3	x_1	-14.2	195.07%	
	x_2	-0.94	355.32%	
	x_3	5.528	152.10%	355.32%
4	x_1	17.82	179.69%	
	x_2	3.264	128.80%	
	x_3	-3.9168	241.14%	241.14%
5	x_1	-16.792	206.12%	
	x_2	-1.4584	323.81%	
	x_3	6.15008	163.69%	323.81%
6	x_1	19.3752	186.67%	
	x_2	3.57504	140.79%	
	x_3	-4.29005	243.36%	243.36%
7	x_1	-17.7251	209.31%	
	x_2	-1.64502	317.32%	
	x_3	6.374029	167.31%	317.32%
8	x_1	19.93507	188.91%	
	x_2	3.687014	144.62%	
	x_3	-4.42442	244.06%	244.06%

11.7 The equations to be solved are

$$f_1(x, y) = -x^2 + x + 0.5 - y$$

$$f_2(x, y) = x^2 - y - 5xy$$

The partial derivatives can be computed and evaluated at the initial guesses

$$\frac{\partial f_{1,0}}{\partial x} = -2x + 1 = -2(1.2) + 1 = -1.4 \qquad \frac{\partial f_{1,0}}{\partial y} = -1$$

$$\frac{\partial f_{2,0}}{\partial x} = 2x - 5y = 2(1.2) - 5(1.2) = -3.6 \quad \frac{\partial f_{2,0}}{\partial y} = -1 - 5x = -1 - 5(1.2) = -7$$

They can then be used to compute the determinant of the Jacobian for the first iteration is

$$-1.4(-7) - (-1)(-3.6) = 6.2$$

The values of the functions can be evaluated at the initial guesses as

$$f_{1,0} = -1.2^2 + 1.2 + 0.5 - 1.2 = -0.94$$

$$f_{2,0} = 1.2^2 - 5(1.2)(1.2) - 1.2 = -6.96$$

These values can be substituted into Eq. (11.12) to give

$$x_1 = 1.2 - \frac{-0.94(-3.6) - (-6.96)(-1)}{6.2} = 1.26129$$

$$x_2 = 1.2 - \frac{-6.96(-1.4) - (-0.94)(-3.6)}{6.2} = 0.174194$$

The computation can be repeated until an acceptable accuracy is obtained. The results are summarized as

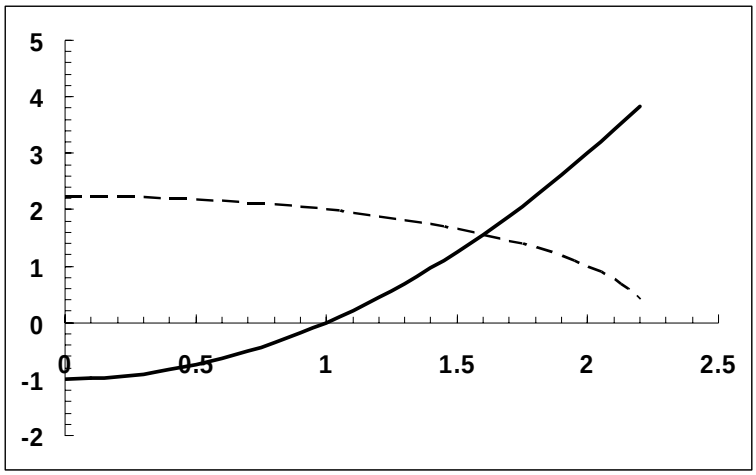
iteration	x	y	ε_{a1}	ε_{a2}
0	1.2	1.2		
1	1.26129	0.174194	4.859%	588.889%
2	1.234243	0.211619	2.191%	17.685%
3	1.233319	0.212245	0.075%	0.295%
4	1.233318	0.212245	0.000%	0.000%

11.8 (a) The equations can be set up in a form amenable to plotting as

$$y = x^2 - 1$$

$$y = \sqrt{5 - x^2}$$

These can be plotted as



Thus, a solution seems to lie at about $x = y = 1.6$.

(b) The equations can be solved in a number of different ways. For example, the first equation can be solved for x and the second solved for y. For this case, successive substitution does not work

First iteration:

$$x = \sqrt{5 - y^2} = \sqrt{5 - (1.5)^2} = 1.658312$$
$$y = (1.658312)^2 - 1 = 1.75$$

Second iteration:

$$x = \sqrt{5 - (1.75)^2} = 1.391941$$
$$y = (1.391941)^2 - 1 = 0.9375$$

Third iteration:

$$x = \sqrt{5 - (0.9375)^2} = 2.030048$$
$$y = (2.030048)^2 - 1 = 3.12094$$

Thus, the solution is moving away from the solution that lies at approximately $x = y = 1.6$.

An alternative solution involves solving the second equation for x and the first for y. For this case, successive substitution does work

First iteration:

$$x = \sqrt{y + 1} = \sqrt{1.5 + 1} = 1.581139$$
$$y = \sqrt{5 - x^2} = \sqrt{5 - (1.581139)^2} = 1.581139$$

Second iteration:

$$x = \sqrt{1.581139} = 1.606592$$
$$y = \sqrt{5 - (1.606592)^2} = 1.555269$$

Third iteration:

$$x = \sqrt{5 - (1.555269)^2} = 1.598521$$
$$y = (1.598521)^2 - 1 = 1.563564$$

After several more iterations, the calculation converges on the solution of $x = 1.600485$ and $y = 1.561553$.

(c) The equations to be solved are

$$f_1(x, y) = x^2 - y - 1$$

$$f_2(x, y) = 5 - y^2 - x^2$$

The partial derivatives can be computed and evaluated at the initial guesses

$$\frac{\partial f_{1,0}}{\partial x} = 2x \qquad \frac{\partial f_{1,0}}{\partial y} = -1$$

$$\frac{\partial f_{2,0}}{\partial x} = -2x \qquad \frac{\partial f_{2,0}}{\partial y} = -2y$$

They can then be used to compute the determinant of the Jacobian for the first iteration is

$$-1.4(-7) - (-1)(-3.6) = 6.2$$

The values of the functions can be evaluated at the initial guesses as

$$f_{1,0} = -1.2^2 + 1.2 + 0.5 - 1.2 = -0.94$$

$$f_{2,0} = 1.2^2 - 5(1.2)(1.2) - 1.2 = -6.96$$

These values can be substituted into Eq. (11.12) to give

$$x_1 = 1.2 - \frac{-0.94(-3.6) - (-6.96)(-1)}{6.2} = 1.26129$$

$$x_2 = 1.2 - \frac{-6.96(-1.4) - (-0.94)(-3.6)}{6.2} = 0.174194$$

The computation can be repeated until an acceptable accuracy is obtained. The results are summarized as

iteration	ξ	ψ	ε_{a1}	ε_{a2}
0	1.5	1.5		
1	1.604167	1.5625	6.494%	4.000%
2	1.600489	1.561553	0.230%	0.061%
3	1.600485	1.561553	0.000%	0.000%

CHAPTER 12

12.1 The data can be tabulated as

i	y	$(y_i - \bar{y})^2$
1	8.8	0.725904
2	9.4	0.063504
3	10	0.121104
4	9.8	0.021904
5	10.1	0.200704
6	9.5	0.023104
7	10.1	0.200704
8	10.4	0.559504
9	9.5	0.023104
10	9.5	0.023104
11	9.8	0.021904
12	9.2	0.204304
13	7.9	3.069504
14	8.9	0.565504
15	9.6	0.002704
16	9.4	0.063504
17	11.3	2.715904
18	10.4	0.559504
19	8.8	0.725904
20	10.2	0.300304
21	10	0.121104
22	9.4	0.063504
23	9.8	0.021904
24	10.6	0.898704
25	8.9	0.565504
Σ	**241.3**	**11.8624**

$$\bar{y} = \frac{241.3}{25} = 9.652$$

$$s_y = \sqrt{\frac{11.8624}{25-1}} = 0.703041$$

$$s_y^2 = 0.703041^2 = 0.494267$$

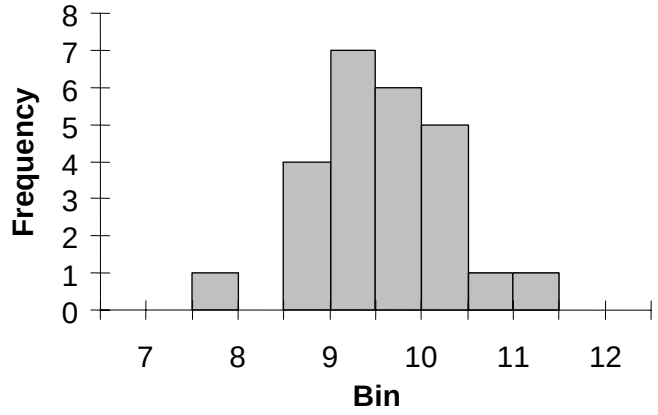
$$\text{c.v.} = \frac{0.703041}{9.652} \times 100\% = 7.28\%$$

12.2 The data can be sorted and then grouped. We assume that if a number falls on the border between bins, it is placed in the lower bin.

lower	upper	Frequency
7.5	8	1

8	8.5	0
8.5	9	4
9	9.5	7
9.5	10	6
10	10.5	5
10.5	11	1
11	11.5	1

The histogram can then be constructed as



12.3 The data can be tabulated as

i	y	$(y_i - \bar{y})^2$
1	28.65	0.390625
2	28.65	0.390625
3	27.65	0.140625
4	29.25	1.500625
5	26.55	2.175625
6	29.65	2.640625
7	28.45	0.180625
8	27.65	0.140625
9	26.65	1.890625
10	27.85	0.030625
11	28.65	0.390625
12	28.65	0.390625
13	27.65	0.140625
14	27.05	0.950625
15	28.45	0.180625
16	27.65	0.140625
17	27.35	0.455625
18	28.25	0.050625
19	31.65	13.14063
20	28.55	0.275625
21	28.35	0.105625
22	28.85	0.680625
23	26.35	2.805625
24	27.65	0.140625

25	26.85	1.380625
26	26.75	1.625625
27	27.75	0.075625
28	27.25	0.600625
Σ	**784.7**	**33.0125**

(a) $\bar{y} = \dfrac{784.7}{28} = 28.025$

(b) $s_y = \sqrt{\dfrac{33.0125}{28-1}} = 1.105751$

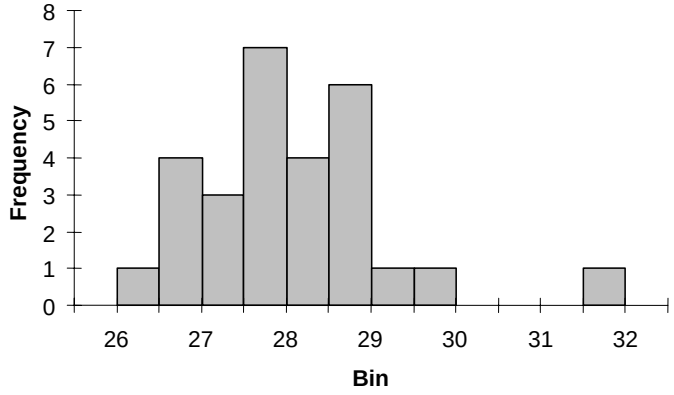
(c) $s_y^2 = 1.105751^2 = 1.222685$

(d) $\text{c.v.} = \dfrac{1.105751}{28.025} \times 100\% = 3.95\%$

(e) The data can be sorted and grouped.

Lower	Upper	Frequency
26	26.5	1
26.5	27	4
27	27.5	3
27.5	28	7
28	28.5	4
28.5	29	6
29	29.5	1
29.5	30	1
30	30.5	0
30.5	31	0
31	31.5	0
31.5	32	1

The histogram can then be constructed as



(f) 68% of the readings should fall between $\bar{y} - s_y$ and $\bar{y} + s_y$. That is, between 28.025 – 1.10575096 = 26.919249 and 28.025 + 1.10575096 = 29.130751. Twenty values fall between these bounds which is equal to 20/28 = 71.4% of the values which is not that far from 68%.

12.4 The sum of the squares of the residuals for this case can be written as

$$S_r = \sum_{i=1}^{n} (y_i - a_1 x_i)^2$$

The partial derivative of this function with respect to the single parameter a_1 can be determined as

$$\frac{\partial S_r}{\partial a_1} = -2\sum [(y_i - a_1 x_i) x_i]$$

Setting the derivative equal to zero and evaluating the summations gives

$$\sum y_i - a_1 \sum x_i$$

which can be solved for

$$a_1 = \frac{\sum y_i}{\sum x_i}$$

So the slope that minimizes the sum of the squares of the residuals for a straight line with a zero intercept is merely the ratio of the sum of the dependent variables (y) over the sum of the independent variables (x).

12.5

i	x_i	y_i	x_i^2	$x_i y_i$
1	0	9.8100	0	0
2	20000	9.7487	4.0E+08	194974
3	40000	9.6879	1.6E+09	387516
4	60000	9.6278	3.6E+09	577668
5	80000	9.5682	6.4E+09	765456
Σ	**200000**	**48.4426**	**1.2E+10**	**1925614**

$$a_1 = \frac{5(1{,}925{,}614) - 200{,}000(48.4426)}{5(1.2 \times 10^{10}) - 200{,}000^2} = -3.0225 \times 10^{-6}$$

$$a_0 = \frac{48.4426}{5} - 3.0225 \times 10^{-6} \frac{200{,}000}{5} = 9.80942$$

Therefore, the line of best fit is (using the nomenclature of the problem)

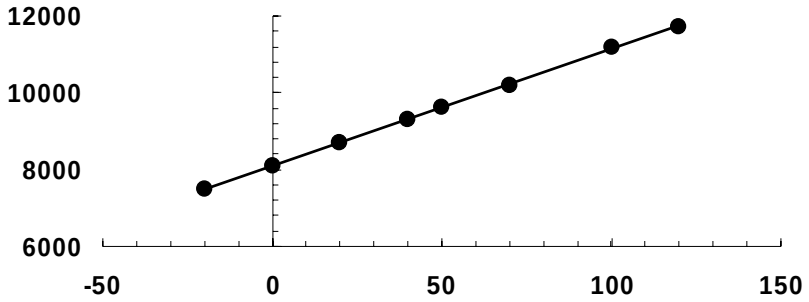
$$g = 9.80942 - 3.0225 \times 10^{-6} y$$

The value at 55,000 m can therefore be computed as

$$g = 9.80942 - 3.0225 \times 10^{-6} (55{,}000) = 9.6431825$$

12.6 Regression gives

$$p = 8100.47 + 30.3164T \qquad (r^2 = 0.999)$$



$$R = \left(\frac{p}{T}\right)\frac{V}{n}$$

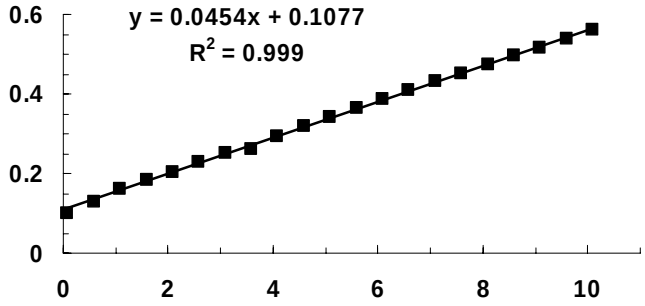
$$\frac{p}{T} = 30.3164$$

$$n = \frac{1\,\text{kg}}{28\,\text{g/mole}}$$

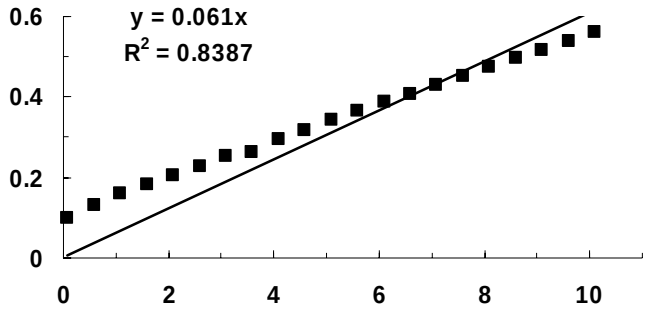
$$R = 30.3164\left(\frac{10}{10^3 / 28}\right) = 8.487$$

This is close to the standard value of 8.314 J/gmole.

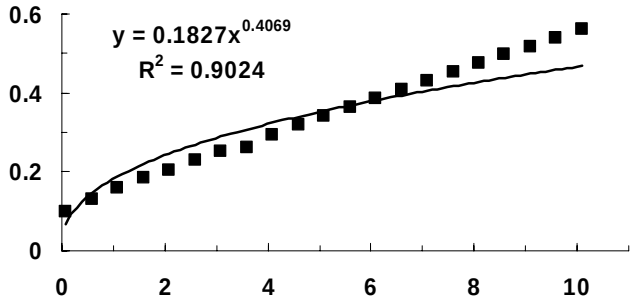
12.7 Linear regression gives



Forcing a zero intercept yields



One alternative that would force a zero intercept is a power fit



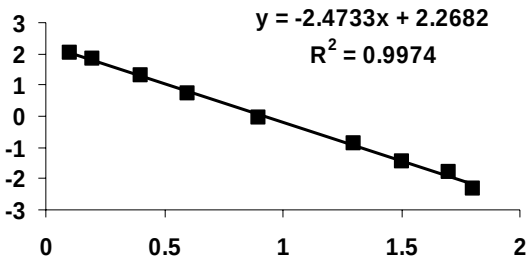
However, this seems to represent a poor compromise since it misses the linear trend in the data. An alternative approach would to assume that the physically-unrealistic non-zero intercept is an artifact of the measurement method. Therefore, if the linear slope is valid, we might try $y = 0.0454x$.

12.8 The function can be linearized by dividing it by x and taking the natural logarithm to yield

$$\ln(y/x) = \ln\alpha_4 + \beta_4 x$$

Therefore, if the model holds, a plot of ln(y/x) versus x should yield a straight line with an intercept of $\ln\alpha_4$ and an intercept of β_4.

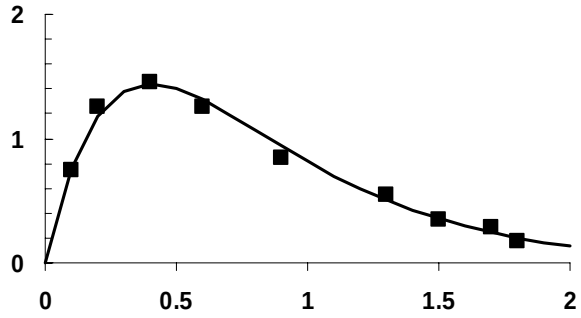
x	y	ln(y/x)
0.1	0.75	2.014903
0.2	1.25	1.832581
0.4	1.45	1.287854
0.6	1.25	0.733969
0.9	0.85	-0.05716
1.3	0.55	-0.8602
1.5	0.35	-1.45529
1.7	0.28	-1.80359
1.8	0.18	-2.30259



Therefore, $\beta_4 = -2.4733$ and $\alpha_4 = e^{2.2682} = 9.661786$, and the fit is

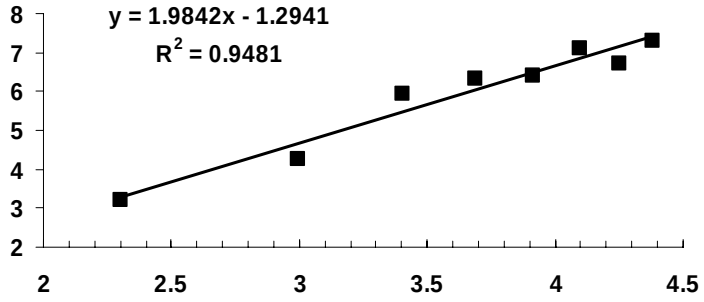
$$y = 9.661786xe^{-2.4733x}$$

This equation can be plotted together with the data:



12.9 The data can be transformed, plotted and fit with a straight line

v, m/s	F, N	ln v	ln F
10	25	2.302585	3.218876
20	70	2.995732	4.248495
30	380	3.401197	5.940171
40	550	3.688879	6.309918
50	610	3.912023	6.413459
60	1220	4.094345	7.106606
70	830	4.248495	6.721426
80	1450	4.382027	7.279319



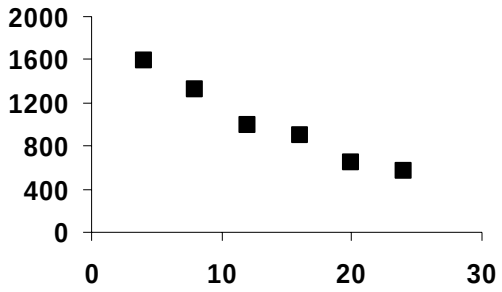
The least-squares fit is

$$\ln y = 1.9842 \ln x - 1.2941$$

The exponent is 1.9842 and the leading coefficient is $e^{-1.2941} = 0.274137$. Therefore, the result is the same as when we used common or base-10 logarithms:

$$y = 0.274137x^{1.9842}$$

12.10 (a) The data can be plotted

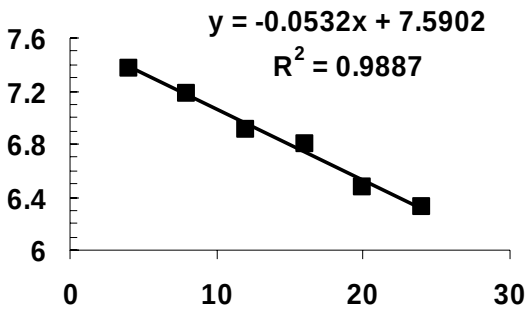


The plot indicates that the data is somewhat curvilinear. An exponential model (i.e., a semi-log plot) is the best choice to linearize the data. This conclusion is based on

- A power model does not result in a linear plot
- Bacterial decay is known to follow an exponential model
- The exponential model by definition will not produce negative values.

The exponential fit can be determined as

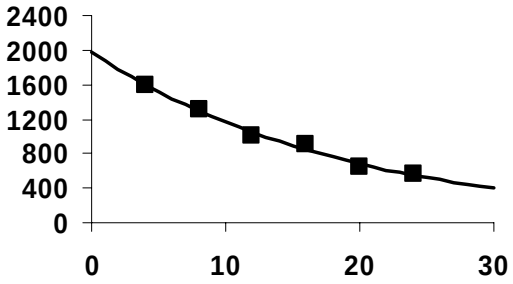
t (hrs)	*c* (CFU/100 mL)	ln *c*
4	1590	7.371489
8	1320	7.185387
12	1000	6.907755
16	900	6.802395
20	650	6.476972
24	560	6.327937



Therefore, the coefficient of the exponent (β_1) is –0.0532 and the lead coefficient (α_1) is $e^{7.5902} = 1978.63$, and the fit is

$$c = 1978.63e^{-0.0532t}$$

Consequently the concentration at $t = 0$ is 1978.63 CFU/100 ml. Here is a plot of the fit along with the original data:



(b) The time at which the concentration will reach 200 CFU/100 mL can be computed as

$$200 = 1978.63e^{-0.0532t}$$

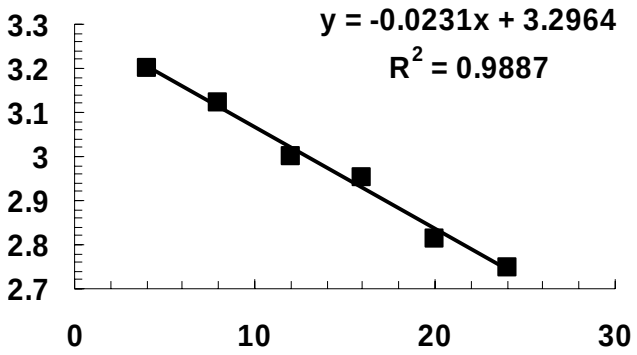
$$\ln\left(\frac{200}{1978.63}\right) = -0.0532t$$

$$t = \frac{\ln\left(\frac{200}{1978.63}\right)}{-0.0532} = 43.08\,\text{d}$$

12.11 (a) The exponential fit can be determined with the base-10 logarithm as

t (hrs)	c (CFU/100 mL)	log c
4	1590	3.201397
8	1320	3.120574
12	1000	3
16	900	2.954243

20	650	2.812913
24	560	2.748188



Therefore, the coefficient of the exponent (β_5) is –0.0231 and the lead coefficient (α_5) is $10^{3.2964} = 1978.63$, and the fit is

$$c = 1978.63(10)^{-0.0231t}$$

Consequently the concentration at $t = 0$ is 1978.63 CFU/100 ml.

(b) The time at which the concentration will reach 200 CFU/100 mL can be computed as

$$200 = 1978.63(10)^{-0.0231t}$$

$$\log_{10}\left(\frac{200}{1978.63}\right) = -0.0231t$$

$$t = \frac{\log_{10}\left(\frac{200}{1978.63}\right)}{-0.0231} = 43.08\,\text{d}$$

Thus, the results are identical to those obtained with the base-*e* model.

The relationship between β_1 and β_5 can be developed as in

$$e^{-\alpha_1 t} = 10^{-\alpha_5 t}$$

Take the natural log of this equation to yield

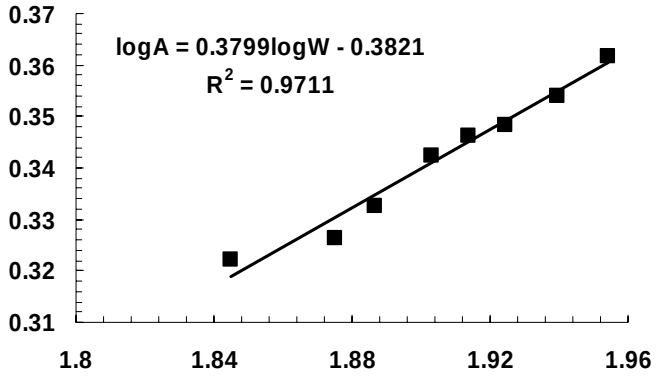
$$-\alpha_1 t = -\alpha_5 t \ln 10$$

or

$$\alpha_1 = 2.302585\alpha_5$$

12.12 The power fit can be determined as

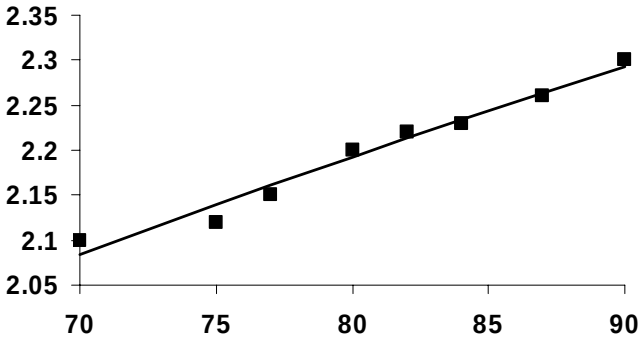
W (kg)	*A* (m^2)	log W	log A
70	2.1	1.845098	0.322219
75	2.12	1.875061	0.326336
77	2.15	1.886491	0.332438
80	2.2	1.90309	0.342423
82	2.22	1.913814	0.346353
84	2.23	1.924279	0.348305
87	2.26	1.939519	0.354108
90	2.3	1.954243	0.361728



Therefore, the power is $b = 0.3799$ and the lead coefficient is $a = 10^{-0.3821} = 0.4149$, and the fit is

$$A = 0.4149W^{0.3799}$$

Here is a plot of the fit along with the original data:

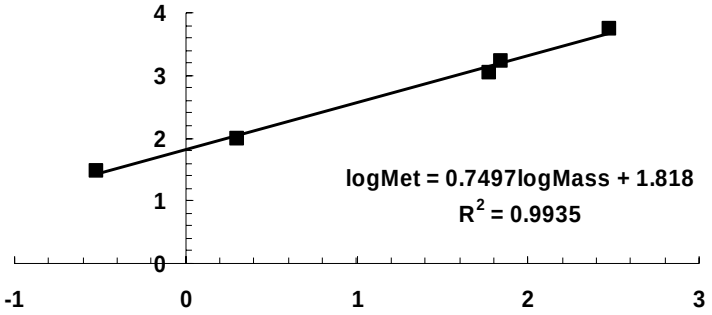


The value of the surface area for a 95-kg person can be estimated as

$$A = 0.4149(95)^{0.3799} = 2.34\,\text{m}^2$$

12.13 The power fit can be determined as

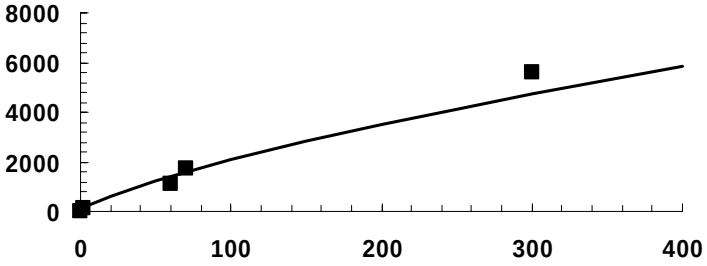
Mass (kg)	Metabolism (kCal/day)	log Mass	log Met
300	5600	2.477121	3.748188
70	1700	1.845098	3.230449
60	1100	1.778151	3.041393
2	100	0.30103	2
0.3	30	-0.52288	1.477121



Therefore, the power is $b = 0.7497$ and the lead coefficient is $a = 10^{1.818} = 65.768$, and the fit is

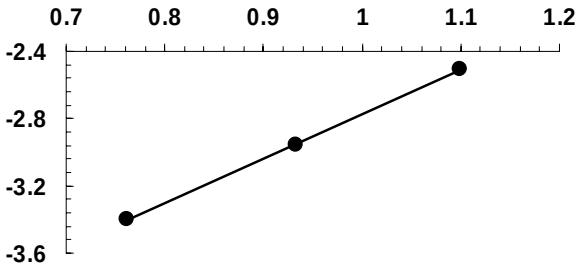
$$\text{Metabolism} = 65.768\text{Mass}^{0.7497}$$

Here is a plot of the fit along with the original data:



12.14 Linear regression of the log transformed data yields

$$\log\dot{\varepsilon} = -5.41\log B + 2.6363\log\sigma \qquad (r^2 = 0.9997)$$



Therefore,

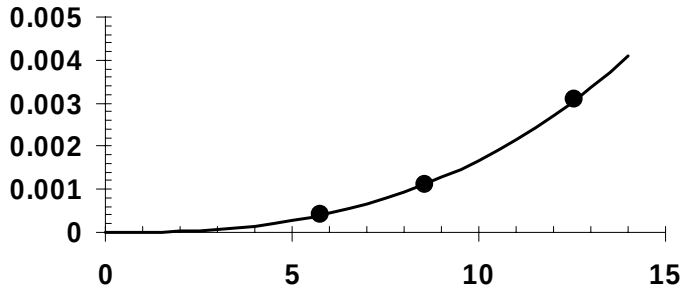
$$B = 10^{-5.41} = 3.88975 \times 10^{-6}$$

$$m = 2.6363$$

and the untransformed model is

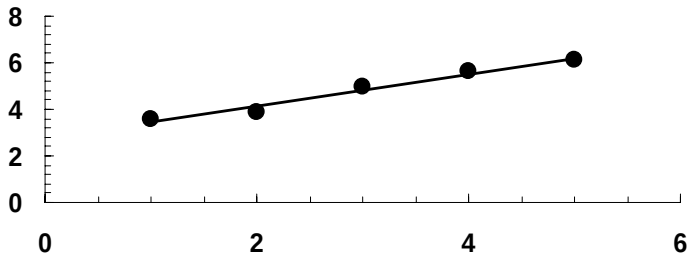
$$\dot{\varepsilon} = 3.88975 \times 10^{-6} \sigma^{2.6363}$$

A plot of the data and the model can be developed as



12.15 Linear regression of the data yields

$$\tau = 2.779 + 0.685\dot{\gamma} \qquad (r^2 = 0.977121)$$



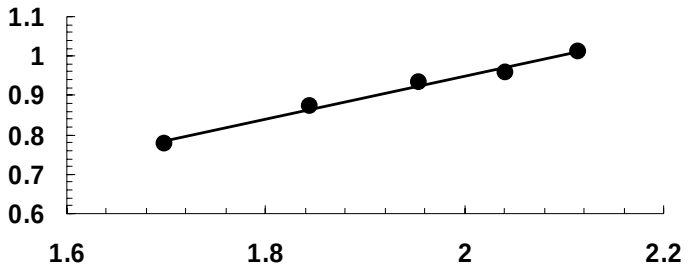
Therefore, $\mu = 0.685$ and $\tau_y = 2.779$ N/m^2.

12.16 The data can be transformed

strain	stress	log(strain)	log(stress)
50	5.99	1.69897	0.777427
70	7.45	1.845098	0.872156
90	8.56	1.954243	0.932474
110	9.09	2.041393	0.958564
130	10.25	2.113943	1.010724

Linear regression of the transformed data yields

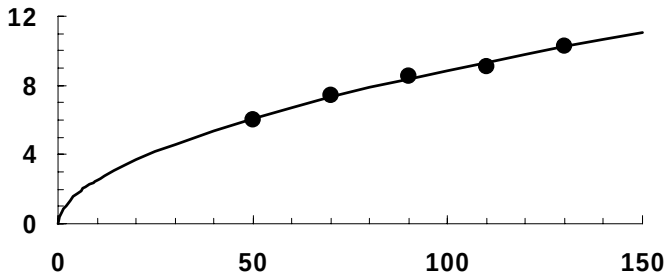
$$\log \tau = -0.13808 + 0.54298 \log \dot{\gamma} \qquad (r^2 = 0.989118)$$



Therefore, $\mu = 10^{-0.54298} = 0.72765$ and $n = 0.54298$. The power model is therefore,

$$\tau = 0.72765 \dot{\gamma}^{0.54298}$$

A plot of the power model along with the data can be created as



CHAPTER 13

13.1 The data can be tabulated and the sums computed as

i	x	y	x^2	x^3	x^4	xy	x^2y
1	10	25	100	1000	10000	250	2500
2	20	70	400	8000	160000	1400	28000
3	30	380	900	27000	810000	11400	342000
4	40	550	1600	64000	2560000	22000	880000
5	50	610	2500	125000	6250000	30500	1525000
6	60	1220	3600	216000	12960000	73200	4392000
7	70	830	4900	343000	24010000	58100	4067000
8	80	1450	6400	512000	40960000	116000	9280000
Σ	360	5135	20400	1296000	87720000	312850	20516500

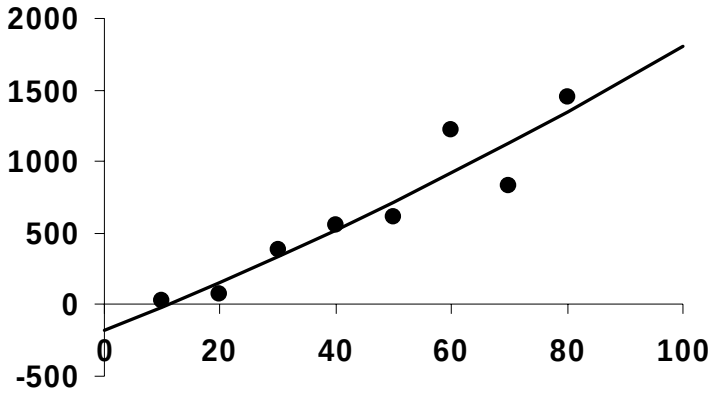
Normal equations:

$$\begin{bmatrix} 8 & 360 & 20400 \\ 360 & 20400 & 1296000 \\ 20400 & 1296000 & 87720000 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix} = \begin{Bmatrix} 5135 \\ 312850 \\ 20516500 \end{Bmatrix}$$

which can be solved for the coefficients yielding the following best-fit polynomial

$$F = -178.4821 + 16.12202v + 0.037202v^2$$

Here is the resulting fit:



The predicted values can be used to determined the sum of the squares. Note that the mean of the y values is 641.875.

i	x	y	y_{pred}	$(y_i - \bar{y})^2$	$(y - y_{pred})^2$
1	10	25	-13.5417	380535	1485
2	20	70	158.8393	327041	7892
3	30	380	338.6607	68579	1709

4	40	550	525.9226	8441	580
5	50	610	720.625	1016	12238
6	60	1220	922.7679	334229	88347
7	70	830	1132.351	35391	91416
8	80	1450	1349.375	653066	10125
Σ				1808297	213793

The coefficient of determination can be computed as

$$r^2 = \frac{1808297 - 213793}{1808297} = 0.88177$$

The model fits the trend of the data nicely, but it has the deficiency that it yields physically unrealistic negative forces at low velocities.

13.2 The sum of the squares of the residuals for this case can be written as

$$S_r = \sum_{i=1}^{n} \left(y_i - a_1 x_i - a_2 x_i^2\right)^2$$

The partial derivatives of this function with respect to the unknown parameters can be determined as

$$\frac{\partial S_r}{\partial a_1} = -2\sum \left[(y_i - a_1 x_i - a_2 x_i^2) x_i\right]$$

$$\frac{\partial S_r}{\partial a_2} = -2\sum \left[(y_i - a_1 x_i - a_2 x_i^2) x_i^2\right]$$

Setting the derivative equal to zero and evaluating the summations gives

$$\left(\sum x_i^2\right) a_1 + \left(\sum x_i^3\right) a_2 = \sum x_i y_i$$

$$\left(\sum x_i^3\right) a_1 + \left(\sum x_i^4\right) a_2 = \sum x_i^2 y_i$$

which can be solved for

$$a_1 = \frac{\sum x_i y_i \sum x_i^4 - \sum x_i^2 y_i \sum x_i^3}{\sum x_i^2 \sum x_i^4 - \left(\sum x_i^3\right)^2}$$

$$a_2 = \frac{\sum x_i^2 \sum x_i^2 y_i - \sum x_i y_i \sum x_i^3}{\sum x_i^2 \sum x_i^4 - \left(\sum x_i^3\right)^2}$$

The model can be tested for the data from Table 12.1.

x	y	x^2	x^3	x^4	xy	x^2y
10	25	100	1000	10000	250	2500
20	70	400	8000	160000	1400	28000
30	380	900	27000	810000	11400	342000
40	550	1600	64000	2560000	22000	880000
50	610	2500	125000	6250000	30500	1525000
60	1220	3600	216000	12960000	73200	4392000
70	830	4900	343000	24010000	58100	4067000
80	1450	6400	512000	40960000	116000	9280000
Σ		20400	1296000	87720000	312850	20516500

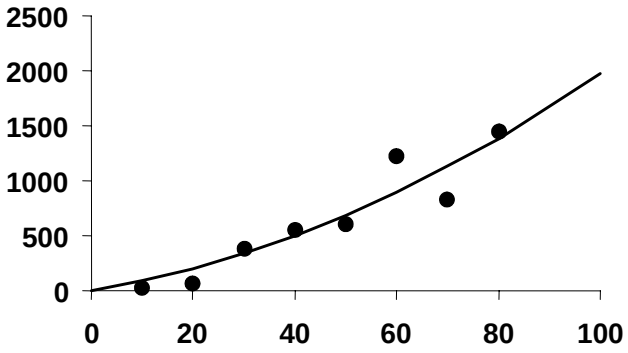
$$a_1 = \frac{312850(87720000) - 20516500(1296000)}{20400(87720000) - (1296000)^2} = 7.771024$$

$$a_2 = \frac{20400(20516500) - 312850(1296000)}{20400(87720000) - (1296000)} = 0.119075$$

Therefore, the best-fit model is

$$y = 7.771024x + 0.119075x^2$$

The fit, along with the original data can be plotted as



13.3 The data can be tabulated and the sums computed as

i	x	y	x^2	x^3	x^4	x^5	x^6	xy	x^2y	x^3y
1	3	1.6	9	27	81	243	729	4.8	14.4	43.2
2	4	3.6	16	64	256	1024	4096	14.4	57.6	230.4
3	5	4.4	25	125	625	3125	15625	22	110	550
4	7	3.4	49	343	2401	16807	117649	23.8	166.6	1166.2
5	8	2.2	64	512	4096	32768	262144	17.6	140.8	1126.4
6	9	2.8	81	729	6561	59049	531441	25.2	226.8	2041.2
7	11	3.8	121	1331	14641	161051	1771561	41.8	459.8	5057.8
8	12	4.6	144	1728	20736	248832	2985984	55.2	662.4	7948.8
Σ	59	26.4	509	4859	49397	522899	5689229	204.8	1838.4	18164

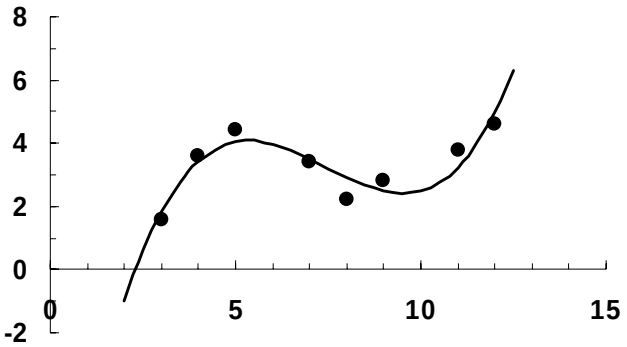
Normal equations:

$$\begin{bmatrix} 8 & 59 & 509 & 4859 \\ 59 & 509 & 4859 & 49397 \\ 509 & 4859 & 49397 & 522899 \\ 4859 & 49397 & 522899 & 5689229 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{Bmatrix} = \begin{Bmatrix} 26.4 \\ 204.8 \\ 1838.4 \\ 18164 \end{Bmatrix}$$

which can be solved for the coefficients yielding the following best-fit polynomial

$$y = -11.4887 + 7.143817x - 1.04121x^2 + 0.046676x^3$$

Here is the resulting fit:



The predicted values can be used to determined the sum of the squares. Note that the mean of the *y* values is 3.3.

i	x	y	y_{pred}	$(y_i - \bar{y})^2$	$(y - y_{pred})^2$
1	3	1.6	1.83213	2.8900	0.0539
2	4	3.6	3.41452	0.0900	0.0344
3	5	4.4	4.03471	1.2100	0.1334
4	7	3.4	3.50875	0.0100	0.0118
5	8	2.2	2.92271	1.2100	0.5223
6	9	2.8	2.4947	0.2500	0.0932
7	11	3.8	3.23302	0.2500	0.3215
8	12	4.6	4.95946	1.6900	0.1292
Σ				7.6000	1.2997

The coefficient of determination can be computed as

$$r^2 = \frac{7.6 - 1.2997}{7.6} = 0.829$$

13.4

```
function p = polyreg(x,y,m)
% polyreg(x,y,m):
%   Polynomial regression.
```

```
% input:
%   x = independent variable
%   y = dependent variable
%   m = order of polynomial
% output:
%   p = vector of coefficients

n = length(x);
if length(y)~=n, error('x and y must be same length'); end
for i = 1:m+1
  for j = 1:i
    k = i+j-2;
    s = 0;
    for l = 1:n
      s = s + x(l)^k;
    end
    A(i,j) = s;
    A(j,i) = s;
  end
  s = 0;
  for l = 1:n
    s = s + y(l)*x(l)^(i-1);
  end
  b(i) = s;
end
p = A\b';
```

Test solving Prob. 13.3:

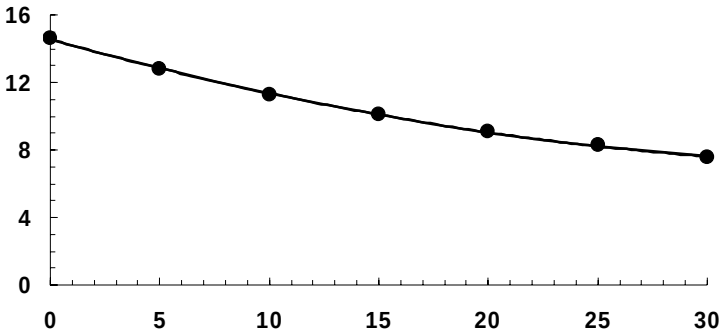
```
>> x = [3 4 5 7 8 9 11 12];
>> y = [1.6 3.6 4.4 3.4 2.2 2.8 3.8 4.6];
>> polyreg(x,y,3)

ans =

  -11.4887
    7.1438
   -1.0412
    0.0467
```

13.5 Because the data is curved, a linear regression will undoubtedly have too much error. Therefore, as a first try, fit a parabola,

```
>> format long
>> T = [0 5 10 15 20 25 30];
>> c = [14.6 12.8 11.3 10.1 9.09 8.26 7.56];
>> p = polyfit(T,c,2)

p =
   0.00439523809524  -0.36335714285714  14.55190476190477
```

Thus, the best-fit parabola would be

$$c = 14.55190476 - 0.36335714T + 0.0043952381T^2$$



We can use this equation to generate predictions corresponding to the data. When these values are rounded to the same number of significant digits the results are

T	*c*-data	*c*-pred	rounded
0	14.6	14.55190	14.6
5	12.8	12.84500	12.8
10	11.3	11.35786	11.4
15	10.1	10.09048	10.1
20	9.09	9.04286	9.04
25	8.26	8.21500	8.22
30	7.56	7.60690	7.61

Thus, although the plot looks good, discrepancies occur in the third significant digit.

We can, therefore, fit a third-order polynomial

```
>> p = polyfit(T,c,3)

p =
  -0.00006444444444   0.00729523809524  -0.39557936507936  14.60023809523810
```

Thus, the best-fit cubic would be

$$c = 14.600238095 - 0.395579365T + 0.007295238T^2 - 0.000064444T^3$$

We can use this equation to generate predictions corresponding to the data. When these values are rounded to the same number of significant digits the results are

T	*c*-data	*c*-pred	rounded
0	14.6	14.60020	14.6
5	12.8	12.79663	12.8
10	11.3	11.30949	11.3
15	10.1	10.09044	10.1
20	9.09	9.09116	9.09
25	8.26	8.26331	8.26
30	7.56	7.55855	7.56

Thus, the predictions and data agree to three significant digits.

13.6 The multiple linear regression model to evaluate is

$$o = a_0 + a_1 T + a_2 c$$

The [*Z*] and *y* matrices can be set up using MATLAB commands in a fashion similar to Example 13.4,

```
>> format long
>> t = [0 5 10 15 20 25 30];
>> T = [t t t]';
>> c = [zeros(size(x)) 10*ones(size(x)) 20*ones(size(x))]';
>> Z = [ones(size(T)) T c];
>> y = [14.6 12.8 11.3 10.1 9.09 8.26 7.56 12.9 11.3 10.1 9.03 8.17
7.46 6.85 11.4 10.3 8.96 8.08 7.35 6.73 6.2]';
```

The coefficients can be evaluated as

```
>> a = Z\y

a =
  13.52214285714286
  -0.20123809523810
  -0.10492857142857
```

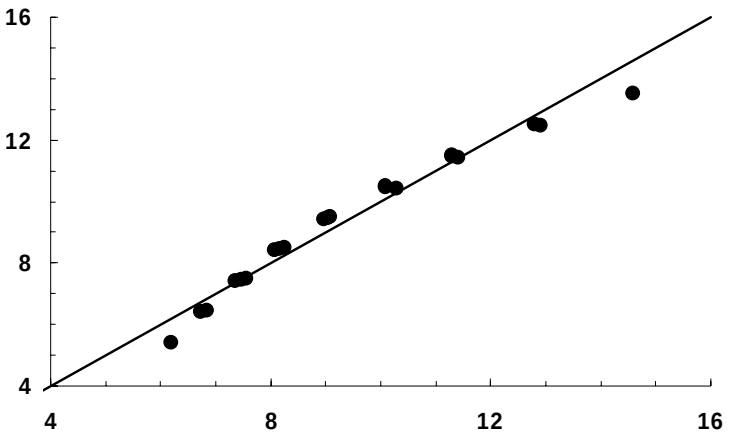
Thus, the best-fit multiple regression model is

$$o = 13.52214285714286 - 0.20123809523810T - 0.10492857142857c$$

We can evaluate the prediction at $T = 12$ and $c = 15$ and evaluate the percent relative error as

```
>> cp = a(1)+a(2)*12+a(3)*15

cp =
   9.53335714285714

>> ea = abs((9.09-cp)/9.09)*100

ea =
   4.87741631305987
```

Thus, the error is considerable. This can be seen even better by generating predictions for all the data and then generating a plot of the predictions versus the data. A one-to-one line is included to show how the predictions diverge from a perfect fit.



The cause for the discrepancy is because the dependence of oxygen concentration on the unknowns is significantly nonlinear. It should be noted that this is particularly the case for the dependency on temperature.

13.7 The multiple linear regression model to evaluate is

$$y = a_0 + a_1 T + a_2 T^2 + a_3 T^3 + a_4 c$$

The [*Z*] matrix can be set up as in

```
>> T = 0:5:30;
>> T = [T T T]';
>> c = [0 0 0  0 0 0 0 10 10 10 10 10 10 10 20 20 20 20 20 20 20]';
>> o = [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]';
>> y = [14.6 12.8 11.3 10.1 9.09 8.26 7.56 12.9 11.3 10.1 9.03 8.17
7.46 6.85 11.4 10.3 8.96 8.08 7.35 6.73 6.2]';
>> Z = [o T T.^2 T.^3 c];
```

Then, the coefficients can be generated by solving Eq.(13.10)

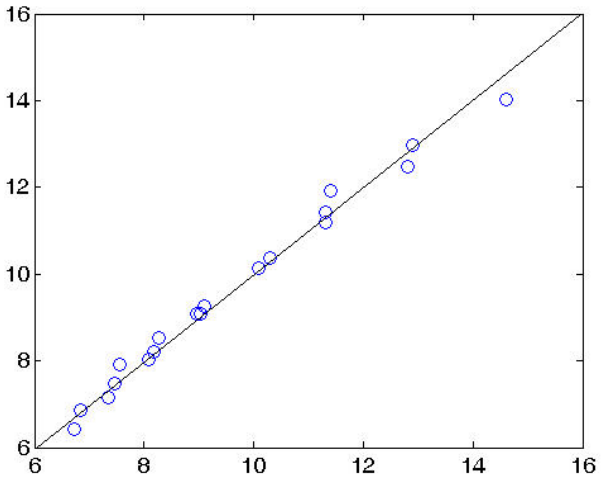
```
>> format long
>> a = (Z'*Z)\[Z'*y]

a =
  14.02714285714287
  -0.33642328042328
   0.00574444444444
  -0.00004370370370
  -0.10492857142857
```

Thus, the least-squares fit is

$$y = 14.027143 - 0.336423T + 0.00574444T^2 - 0.000043704T^3 - 0.10492857c$$

The model can then be used to predict values of oxygen at the same values as the data. These predictions can be plotted against the data to depict the goodness of fit.

```
>> yp = Z*a
>> plot(y,yp,'o')
```



Finally, the prediction can be made at $T = 12$ and $c = 15$,

```
>> a(1)+a(2)*12+a(3)*12^2+a(4)*12^3+a(5)*15
ans =
   9.16781492063485
```

which compares favorably with the true value of 9.09 mg/L.

13.8 The multiple linear regression model to evaluate is

$$y = a_0 + a_1 x_1 + a_2 x_2$$

The [*Z*] matrix can be set up as in

```
>> x1 = [0 1 1 2 2 3 3 4 4]';
>> x2 = [0 1 2 1 2 1 2 1 2]';
>> y = [15.1 17.9 12.7 25.6 20.5 35.1 29.7 45.4 40.2]';
>> o = [1 1 1 1 1 1 1 1 1]';
>> Z = [o x1 x2 y];
```

Then, the coefficients can be generated by solving Eq.(13.10)

```
>> a = (Z'*Z)\[Z'*y]

a =
   14.4609
    9.0252
   -5.7043
```

Thus, the least-squares fit is

$$y = 14.4609 + 9.0252 x_1 - 5.7043 x_2$$

The model can then be used to predict values of the unknown at the same values as the data. These predictions can be used to determine the correlation coefficient and the standard error of the estimate.

```
>> yp = Z*a

>> SSR = sum((yp - y).^2)
SSR =
    4.7397

>> SST = sum((y - mean(y)).^2)
SST =
  1.0587e+003

>> r2 = (SST - SSR)/SST
r2 =
    0.9955

>> r = sqrt(r2)
r =
    0.9978

>> syx = sqrt(SSR/(length(y)-3))
syx =
    0.8888
```

13.9 The multiple linear regression model to evaluate is

$$\log Q = \log \alpha_0 + \alpha_1 \log(D) + \alpha_2 \log(S)$$

The [*Z*] matrix can be set up as in

```
>> D = [.3 .6 .9 .3 .6 .9 .3 .6 .9]';
>> S = [.001 .001 .001 .01 .01 .01 .05 .05 .05]';
>> Q = [.04 .24 .69 .13 .82 2.38 .31 1.95 5.66]';
>> o = [1 1 1 1 1 1 1 1 1]';
>> Z = [o log10(D) log10(S)]
```

Then, the coefficients can be generated by solving Eq.(13.10)

```
>> a = (Z'*Z)\[Z'*log10(Q)]
a =
    1.5609
    2.6279
    0.5320
```

Thus, the least-squares fit is

$$\log Q = 1.5609 + 2.6279 \log(D) + 0.5320 \log(S)$$

Taking the inverse logarithm gives

$$Q = 10^{1.5609} D^{2.6279} S^{0.5320} = 36.3813 D^{2.6279} S^{0.5320}$$

13.10 The linear regression model to evaluate is

$$p(t) = Ae^{-1.5t} + Be^{-0.3t} + Ce^{-0.05t}$$

The unknowns can be entered and the [*Z*] matrix can be set up as in

```
>> p = [7 5.2 3.8 3.2 2.5 2.1 1.8 1.5 1.2 1.1]';
>> t = [0.5 1 2 3 4 5 6 7 8 9]';
>> Z = [exp(-1.5*t) exp(-0.3*t) exp(-0.05*t)];
```

Then, the coefficients can be generated by solving Eq.(13.10)

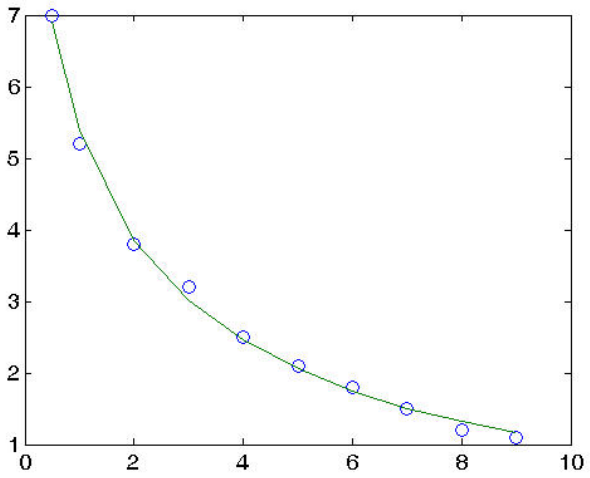
```
>> Z = [exp(-1.5*t) exp(-0.3*t) exp(-0.05*t)];
>> a = (Z'*Z)\[Z'*p]
a =
    3.7778
    4.3872
    1.3775
```

Thus, the least-squares fit is

$$p(t) = 3.7778e^{-1.5t} + 4.3872e^{-0.3t} + 1.3775e^{-0.05t}$$

The fit and the data can be plotted as

```
>> pp = Z*a
>> plot(t,p,'o',t,pp)
```



13.11 First, an M-file function must be created to compute the sum of the squares,

```
function f = fSSR(a,Im,Pm)
Pp = a(1)*Im/a(2).*exp(-Im/a(2)+1);
f = sum((Pm-Pp).^2);
```

The data can then be entered as

```
>> I = [50 80 130 200 250 350 450 550 700];
>> P = [99 177 202 248 229 219 173 142 72];
```

The minimization of the function is then implemented by

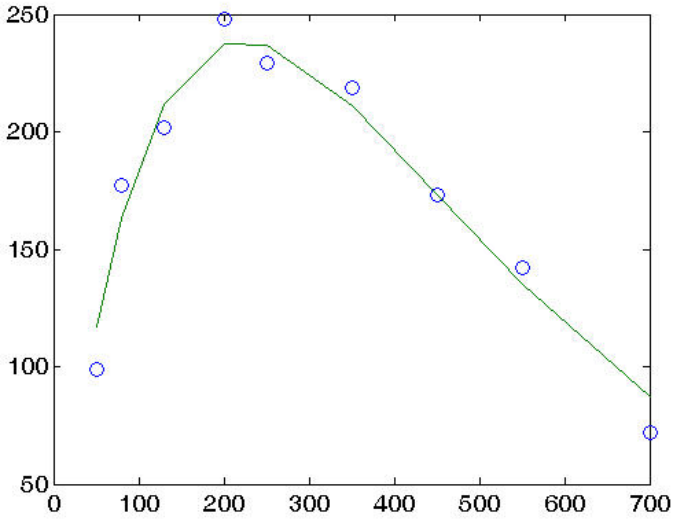
```
>> a = fminsearch(@fSSR, [200, 200], [], I, P)

a =
  238.7124  221.8239
```

The best-fit model is therefore

$$P = 238.7124 \frac{I}{221.8239} e^{-\frac{I}{221.8239}+1}$$

The fit along with the data can be displayed graphically.

```
>> Pp = a(1)*I/a(2).*exp(-I/a(2)+1);
>> plot(I,P,'o',I,Pp)
```



13.12 First, an M-file function must be created to compute the sum of the squares,

```
function f = fSSR(a,xm,ym)
yp = a(1)*xm.*exp(a(2)*xm);
f = sum((ym-yp).^2);
```

The data can then be entered as

```
>> x = [.1 .2 .4 .6 .9 1.3 1.5 1.7 1.9];
>> y = [0.75 1.25 1.45 1.25 0.85 0.55 0.35 0.28 0.18];
```

The minimization of the function is then implemented by

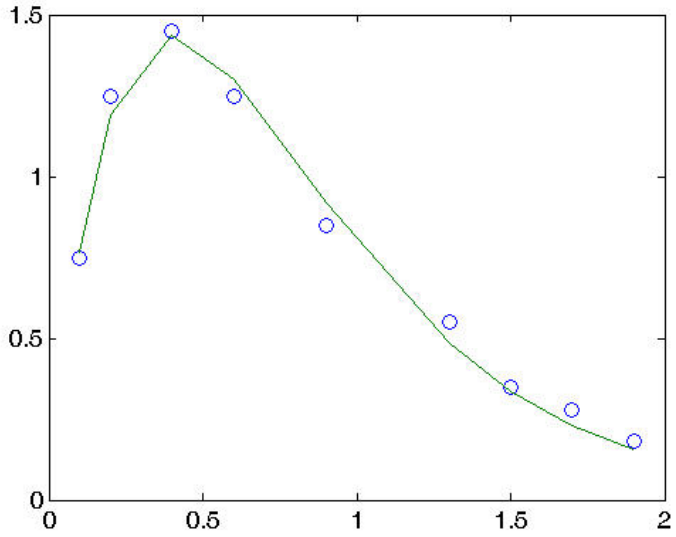
```
>> a = fminsearch(@fSSR, [1, 1], [], x, y)

a =
    9.8545   -2.5217
```

The best-fit model is therefore

$$y = 9.8545xe^{-2.5217x}$$

The fit along with the data can be displayed graphically.

```
>> yp = a(1)*x.*exp(a(2)*x);
>> plot(x,y,'o',x,yp)
```



13.13 (a) The model can be linearized by inverting it,

$$\frac{1}{v_0} = \frac{K}{k_m}\frac{1}{[S]^3} + \frac{1}{k_m}$$

If this model is valid, a plot of $1/v_0$ versus $1/[S]^3$ should yield a straight line with a slope of K/k_m and an intercept of $1/k_m$. The slope and intercept can be implemented in MATLAB using the M-file function `linregr` (Fig. 12.12),

```
>> S = [.01 .05 .1 .5 1 5 10 50 100];
>> v0 = [6.078e-11 7.595e-9 6.063e-8 5.788e-6 1.737e-5 2.423e-5
2.43e-5 2.431e-5 2.431e-5];
>> a = linregr(1./S.^3,1./v0)

a =
  1.0e+004 *
   1.64527391375701   4.13997346408367
```

These results can then be used to compute k_m and K,

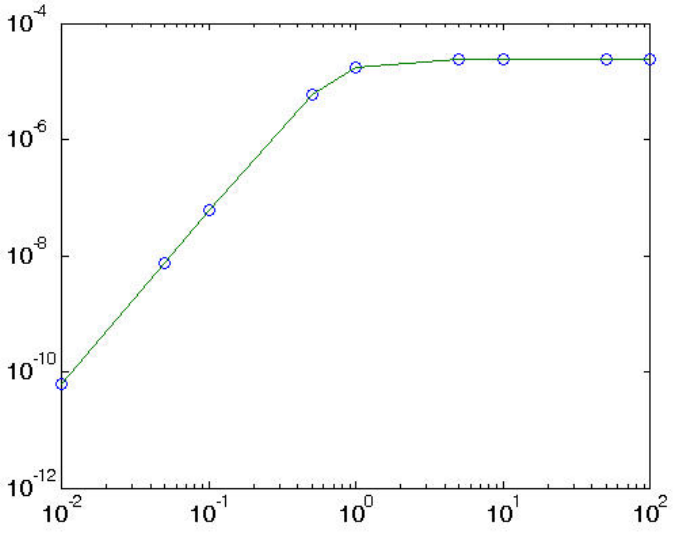
```
>> km=1/a(2)
km =
    2.415474419523452e-005

>> K=km*a(1)
K =
   0.39741170517893
```

Thus, the best-fit model is

$$v_0 = \frac{2.415474 \times 10^{-5}[S]^3}{0.39741 + [S]^3}$$

The fit along with the data can be displayed graphically. We will use a log-log plot because of the wide variation of the magnitudes of the values being displayed,

```
>> v0p = km*S.^3./(K+S.^3);
>> loglog(S,v0,'o',S,v0p)
```



(b) An M-file function must be created to compute the sum of the squares,

```
function f = fSSR(a,Sm,v0m)
v0p = a(1)*Sm.^3./(a(2)+Sm.^3);
f = sum((v0m-v0p).^2);
```

The data can then be entered as

```
>> S = [.01 .05 .1 .5 1 5 10 50 100];
>> v0 = [6.078e-11 7.595e-9 6.063e-8 5.788e-6 1.737e-5 2.423e-5
2.43e-5 2.431e-5 2.431e-5];
```

The minimization of the function is then implemented by

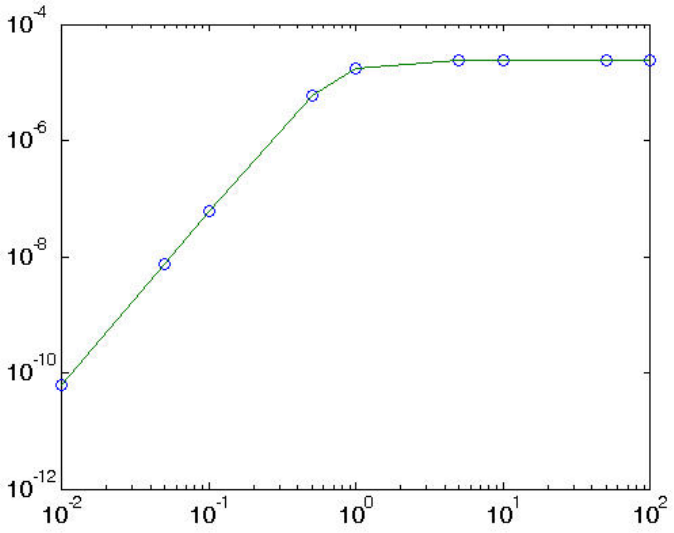
```
>> format long
>> a = fminsearch(@fSSR, [2e-5, 1], [], S, v0)

a =
   0.00002430998303   0.39976314533880
```

The best-fit model is therefore

$$v_0 = \frac{2.431 \times 10^{-5}[S]^3}{0.399763 + [S]^3}$$

The fit along with the data can be displayed graphically. We will use a log-log plot because of the wide variation of the magnitudes of the values being displayed,

```
>> v0p = a(1)*S.^3./(a(2)+S.^3);
>> loglog(S,v0,'o',S,v0p)
```



CHAPTER 14

14.1 **(a)** Newton's polynomial. Ordering of points:

$x_1 = 3 \quad f(x_1) = 6.5$
$x_2 = 4 \quad f(x_2) = 2$
$x_3 = 2.5 \quad f(x_3) = 7$
$x_4 = 5 \quad f(x_4) = 0$

Note that based purely on the distance from the unknown, the fourth point would be (2, 5). However, because it provides better balance and is located only a little bit farther from the unknown, the point at (5, 0) is chosen.

First order:

$$f_1(3.4) = 6.5 + \frac{2 - 6.5}{4 - 3}(3.4 - 3) = 6.5 + (-4.5)(3.4 - 3) = 4.7$$

Second order:

$$f_2(3.4) = 4.7 + \frac{\dfrac{7 - 2}{2.5 - 4} - (-4.5)}{2.5 - 3}(3.4 - 3)(3.4 - 4)$$

$$= 4.7 + \frac{-3.333333 - (-4.5)}{2.5 - 3}(3.4 - 3)(3.4 - 4)$$

$$= 4.7 + (-2.333333)(3.4 - 3)(3.4 - 4) = 5.259887$$

Third order:

$$f_3(3.4) = 5.259887 + \frac{\dfrac{\dfrac{0 - 7}{5 - 2.5} - (-3.333333)}{5 - 4} - (-2.333333)}{5 - 3}(3.4 - 3)(3.4 - 4)(3.4 - 2.5)$$

$$= 5.259887 + \frac{\dfrac{-2.8 - (-3.333333)}{5 - 4} - (-2.333333)}{5 - 3}(3.4 - 3)(3.4 - 4)(3.4 - 2.5)$$

$$= 5.259887 + \frac{0.5333333 - (-2.333333)}{5 - 3}(3.4 - 3)(3.4 - 4)(3.4 - 2.5) = 4.95152$$

(b) Lagrange polynomial.

First order:

$$f_1(3.4) = \frac{3.4 - 4}{3 - 4}6.5 + \frac{3.4 - 3}{4 - 3}2 = 4.7$$

Second order:

$$f_2(3.4) = \frac{(3.4-4)(3.4-2.5)}{(3-4)(3-2.5)}6.5 + \frac{(3.4-3)(3.4-2.5)}{(4-3)(4-2.5)}2 + \frac{(3.4-3)(3.4-4)}{(2.5-3)(2.5-4)}7 = 5.259887$$

Third order:

$$f_3(3.4) = \frac{(3.4-4)(3.4-2.5)(3.4-5)}{(3-4)(3-2.5)(3-5)}6.5 + \frac{(3.4-3)(3.4-2.5)(3.4-5)}{(4-3)(4-2.5)(4-5)}2$$
$$+ \frac{(3.4-3)(3.4-4)(3.4-5)}{(2.5-3)(2.5-4)(2.5-5)}7 + \frac{(3.4-3)(3.4-4)(3.4-2.5)}{(5-3)(5-4)(5-2.5)}0 = 4.95152$$

14.2 The points can be ordered so that they are close to and centered around the unknown. A divided-difference table can then be developed as

x	*f*(*x*)	First	Second	Third	Fourth
3	5.25	7.25	2	0.25	0
5	19.75	5.25	2.75	0.25	
2	4	8	1.75		
6	36	6.25			
1	4.75				

Note that the fact that the fourth divided difference is zero means that the data was generated with a third-order polynomial.

First order:

$$f_1(4) = 5.25 + 7.25(4-3) = 12.5$$

Second order:

$$f_2(4) = 12.5 + (4-3)(4-5)2 = 10.5$$

Third order:

$$f_3(4) = 10.5 + (4-3)(4-5)(4-2)0.25 = 10$$

Fourth order:

$$f_3(4) = 10.5 + (4-3)(4-5)(4-2)(4-6)0 = 10$$

14.3 Lagrange polynomial.

First order:

$$f_1(4) = \frac{4-5}{3-5}5.25 + \frac{4-3}{5-3}19.75 = 12.5$$

Second order:

$$f_2(4) = \frac{(4-5)(4-2)}{(3-5)(3-2)}5.25 + \frac{(4-3)(4-2)}{(5-3)(5-2)}19.75 + \frac{(4-3)(4-5)}{(2-3)(2-5)}4 = 10.5$$

Third order:

$$f_3(4) = \frac{(4-5)(4-2)(4-6)}{(3-5)(3-2)(3-6)}5.25 + \frac{(4-3)(4-2)(4-6)}{(5-3)(5-2)(5-6)}19.75$$

$$+ \frac{(4-3)(4-5)(4-6)}{(2-3)(2-5)(2-6)}4 + \frac{(4-3)(4-5)(4-2)}{(6-3)(6-5)(6-2)}36 = 10$$

14.4 **(a)** The points can be ordered so that they are close to and centered around the unknown. A divided-difference table can then be developed as

T, °C	c = 10 g/L	first	second	third
10	10.1	-0.214	0.0026	0.000107
15	9.03	-0.227	0.003667	
5	11.3	-0.20867		
20	8.17			

Second order:

$$f_2(4) = 10.1 - 0.214(12-10) + 0.0026(12-10)(12-15) = 9.6564$$

Third order:

$$f_3(4) = 9.6564 + 0.000107(12-10)(12-15)(12-5) = 9.65192$$

(b) First, linear interpolation can be used to generate values for T = 10 and 15 at c = 15,

$$f_1(T=10, c=15) = 10.1 + \frac{8.96-10.1}{20-10}(15-10) = 9.53$$

$$f_1(T=15, c=15) = 9.03 + \frac{8.08-9.03}{20-10}(15-10) = 8.555$$

These values can then be used to determine the result at T = 12,

$$f_1(T=12, c=15) = 9.53 + \frac{8.555-9.53}{15-10}(12-10) = 9.14$$

(c) First, quadratic interpolation can be used to generate values for T = 5, 10 and 15 at c = 15,

$$f_2(T=5, c=15) = 12.8 - 0.15(15-0) + 0.0025(15-0)(15-10) = 10.7375$$

$$f_2(T=10, c=15) = 11.3 - 0.12(15-0) + 0.0003(15-0)(15-10) = 9.5225$$

$$f_2(T=15, c=15) = 10.1 - 0.107(15-0) + 0.0006(15-0)(15-10) = 8.54$$

These values can then be used to determine the result at $T = 12$,

$$f_2(T=12, c=15) = 10.7375 - 0.243(12-5) + 0.00465(12-5)(12-10) = 9.1016$$

14.5 MATLAB can be used to generate a cubic polynomial through the first 4 points in the table,

```
>> x = [1 2 3 4];
>> fx = [3.6 1.8 1.2 0.9];
>> p = polyfit(x,fx,3)
p =
   -0.1500    1.5000   -5.2500    7.5000
```

Therefore, the roots problem to be solved is

$$1.6 = -0.15x^3 + 1.5x^2 - 5.25x + 7.5$$

or

$$f(x) = -0.15x^3 + 1.5x^2 - 5.25x + 5.9 = 0$$

Bisection can be employed the root of this polynomial. Using initial guesses of $x_l = 2$ and $x_u = 3$, a value of 2.2156 is obtained with $\varepsilon_a = 0.00069\%$ after 16 iterations.

14.6 (a) Analytical:

$$0.93 = \frac{x^2}{1+x^2}$$

$$0.93 + 0.93x^2 = x^2$$

$$0.07x^2 = 0.93$$

$$x = \sqrt{\frac{0.93}{0.07}} = 3.644957$$

(b) A quadratic interpolating polynomial can be fit to the last three points using the `polyfit` function,

```
>> format long
>> x = [3 4 5];
>> y = x.^2./(1+x.^2);
>> p = polyfit(x,y,2)
p =
  -0.01040723981900   0.11402714932127   0.65158371040724
```

Thus, the best fit quadratic is

$$f_2(x) = -0.01040724x^2 + 0.11402715x + 0.6515837$$

We must therefore find the root of

$$0.93 = -0.01040724x^2 + 0.11402715x + 0.6515837$$

or

$$f(x) = -0.01040724x^2 + 0.11402715x - 0.2784163$$

The quadratic formula yields

$$x = \frac{-0.11402715 \pm \sqrt{(0.11402715)^2 - 4(-0.01040724)(-0.2784163)}}{2(0.11402715)} = \frac{7.2835775}{3.6729442}$$

Thus, the estimate is 3.67294421.

(c) A cubic interpolating polynomial can be fit to the last four points using the `polyfit` function,

```
>> format long
>> x = [2 3 4 5];
>> y=x.^2./(1+x.^2)
>> p = polyfit(x,y,3)
p =
   0.00633484162896  -0.08642533936652   0.41176470588235   0.27149321266968
```

Thus, the best fit cubic is

$$f_3(x) = 0.006334842x^3 - 0.08642534x^2 + 0.4117647x + 0.2714932$$

We must therefore find the root of

$$0.93 = 0.006334842x^3 - 0.08642534x^2 + 0.4117647x + 0.2714932$$

or

$$f(x) = 0.006334842x^3 - 0.08642534x^2 + 0.4117647x - 0.6585068$$

Bisection can be employed the root of this polynomial. Using initial guesses of $x_l = 3$ and $x_u = 4$, a value of 3.61883 is obtained.

14.7 (a) Because they bracket the unknown, the two last points are used for linear interpolation,

$$f_1(0.118) = 6.5453 + \frac{6.7664 - 6.5453}{0.12547 - 0.11144}(0.118 - 0.11144) = 6.6487$$

(b) The quadratic interpolation can be implemented easily in MATLAB,

```
>> v = [0.10377 0.1144 0.12547];
>> s = [6.4147 6.5453 6.7664];
>> p = polyfit(v,s,2)
p =
  354.2358  -64.9976    9.3450
>> polyval(p,0.118)
ans =
    6.6077
```

Therefore, to the level of significance reported in the table the estimated entropy is 6.6077

(c) The inverse interpolation can be implemented in MATLAB. First, as in part **(b)**, we can fit a quadratic polynomial to the data to yield,

```
p =
  354.2358  -64.9976    9.3450
```

We must therefore find the root of

$$6.45 = 354.2358x^2 - 64.9976x + 9.3450$$

or

$$6.45 = 354.2358x^2 - 64.9976x + 2.8950$$

In MATLAB, we can generate this polynomial by subtracting 6.45 from the constant coefficient of the polynomial

```
>> p(3)=p(3)-6.45
p =
  354.2358  -64.9976    2.8950
```

Then, we can use the `roots` function to determine the solution,

```
>> roots(p)
ans =
    0.1074
    0.0761
```

Thus, the value of the specific volume corresponding to an entropy of 6.45 is 0.1074.

14.8 This problem is nicely suited for the Newton interpolating polynomial. First, we can order the data so that the points are closest to and centered around the unknown,

T	*D*
300	1.139

350	0.967
400	0.854
250	1.367
450	0.759
200	1.708

Then we can generate the divided difference table,

T	D	first	second	third	fourth	fifth
300	1.139	-0.003440	1.18000E-05	4.00000E-09	-2.93333E-10	-2.77333E-12
350	0.967	-0.002260	1.16000E-05	-4.00000E-08	-1.60000E-11	
400	0.854	-0.003420	7.60000E-06	-3.76000E-08		
250	1.367	-0.003040	1.51200E-05			
450	0.759	-0.003796				
200	1.708					

First-order (linear) fit:

$$f_1(330) = 1.139 - 0.00344(330 - 300) = 1.0358$$

Thus, the linear estimate is 1.036 to the level of significant digits provided in the original data.

Second-order (quadratic) fit:

$$f_2(330) = 1.0358 + 1.18 \times 10^{-5}(330 - 300)(330 - 350) = 1.0287$$

The quadratic estimate is 1.029 to the level of significant digits provided in the original data.

Third-order (cubic) fit:

$$f_3(330) = 1.0287 + 4 \times 10^{-9}(330 - 300)(330 - 350)(330 - 400) = 1.028888$$

The cubic estimate is also 1.029.

Fourth-order (quartic) fit:

$$f_4(330) = 1.0289 - 2.93333^{-10}(330 - 300)(330 - 350)(330 - 400)(330 - 250) = 1.0279$$

The quartic estimate now seems to be diverging slightly by moving to a value of 1.028. This may be an initial indication that the higher-order terms are beginning to induce slight oscillations.

Fifth-order (quintic) fit:

$$f_2(330) = 1.0279 - 2.77333^{-12}(330 - 300)(330 - 350)(330 - 400)(330 - 250)(330 - 450) = 1.02902$$

Oscillations are now evidently occurring as the fifth-order estimate now jumps back up to slightly above a value of 1.029.

On the basis of the foregoing, I would conclude that the cubic equation provides the best approximation and that a value of 1.029 is a sound estimate to the level of significant digits provided in the original data.

Inverse interpolation can be now used to determine the temperature corresponding to the value of density of 1.029. First, MATLAB can be used to fit a cubic polynomial through the four points that bracket this value. Interestingly, because of the large values of the temperatures, we get an error message,

```
>> T = [250 300 350 400];
>> D =[1.3670 1.139 0.967 0.854];
>> p = polyfit(T,D,3)
Warning: Polynomial is badly conditioned. Remove repeated data points
         or try centering and scaling as described in HELP POLYFIT.
(Type "warning off MATLAB:polyfit:RepeatedPointsOrRescale" to suppress
this warning.)
> In polyfit at 78

p =
    0.0000    0.0000   -0.0097    3.2420
```

Let's disregard this warn and proceed to adjust the polynomial so that it can be used to solve the inverse interpolation problem. To do this, we subtract the specified value of the density from the polynomial's constant coefficient

```
>> p(4)=p(4)-1.029

p =
    0.0000    0.0000   -0.0097    2.2130
```

Then we can use the `roots` function to determine the temperature that corresponds to this value

```
>> roots(p)
ans =
  1.0e+003 *
   -2.8237
    0.5938
    0.3300
```

Thus, even though the polynomial is badly conditioned one of the roots corresponds to T = 330 as expected.

Now let's perform the inverse interpolation, but with scaling. To do this, we will merely subtract the value at the midpoint of the temperature range (325) from all the temperatures. This acts to both reduce the magnitudes of the temperatures and centers them on zero,

```
>> format long
>> D = [1.3670 1.139 0.967 0.854];
```

```
>> T = [250 300 350 400];
>> T = T - 325;
```

Then, the cubic fit can be generated with no error message,

```
>> p = polyfit(T,D,3)
p =
   0.00000000400000   0.00001150000000  -0.00344250000000   1.04581250000000
```

We can set up the roots problem

```
>> p(4)=p(4)-1.029
p =
   0.00000000400000   0.00001150000000  -0.00344250000000   0.01681250000000
```

We can then use the `roots` function to determine the temperature that corresponds to the given density

```
>> r = roots(p)
ans =
  1.0e+003 *
  -3.14874694489127
   0.26878060289231
   0.00496634199799
```

By adding back the offset of 325, we arrive at the expected result of 330,

```
>> Tinv = r(3)+325
Tinv =
    3.299663419979927e+002
```

14.9 A MATLAB session provides a handy way to solve this problem

```
>> i = [-1 -0.5 -0.25 0.25 0.5 1];
>> V = [-193 -41 -13.5625 13.5625 41 193];
>> p = polyfit(i,V,5)
p =
    0.0000   -0.0000  148.0000   -0.0000   45.0000    0.0000
```

The interpolating polynomial is therefore

$$V = 148i^3 + 45i$$

The `polyval` function can be used to determine the interpolation at $i = 0.1$,

```
>> polyval(p,0.10)
ans =
   4.6480
```

14.10 Third-order case: The MATLAB polyfit function can be used to generate the cubic polynomial and perform the interpolation,

```
>> x = [1 1.5 2 2.5];
>> J = [0.765198 0.511828 0.223891 -0.048384];
```

```
>> p = polyfit(x,J,3)
p =
    0.0670   -0.3705    0.1014    0.9673
>> Jpred = polyval(p,1.82)
Jpred =
    0.3284
```

The built-in function `besselj` can be used to determine the true value which can then be used to determine the percent relative error

```
>> Jtrue = besselj(0,1.82)
Jtrue =
    0.3284
>> ea = abs((Jtrue-Jpred)/Jtrue)*100
ea =
    0.0043
```

Fourth-order case:

```
>> x = [1 1.5 2 2.5 3];
>> J = [0.765198 0.511828 0.223891 -0.048384 -0.260052];
>> p = polyfit(x,J,4)
p =
   -0.0035    0.0916   -0.4330    0.1692    0.9409
>> Jpred = polyval(p,1.82)
Jpred =
    0.3283
>> Jtrue = besselj(0,1.82);
>> ea = abs((Jtrue-Jpred)/Jtrue)*100
ea =
    0.0302
```

Fifth-order case:

```
>> x = [1 1.5 2 2.5 3 0.5];
>> J = [0.765198 0.511828 0.223891 -0.048384 -0.260052 0.938470];
>> p = polyfit(x,J,5)
p =
   -0.0027    0.0231   -0.0115   -0.2400   -0.0045    1.0008
>> Jpred = polyval(p,1.82)
Jpred =
    0.3284
>> Jtrue = besselj(0,1.82);
>> ea = abs((Jtrue-Jpred)/Jtrue)*100
ea =
  5.2461e-004
```

14.11 In the same fashion as Example 14.6, MATLAB can be used to evaluate each of the cases,

First order:

```
>> t = [1990 1980];
>> pop = [249.46 227.23];
>> ts = (t - 1955)/35;
>> p = polyfit(ts,pop,1);
```

```
>> polyval(p,(2000-1955)/35)
ans =
  271.6900
```

Second order:

```
>> t = [t 1970];
>> pop = [pop 205.05];
>> ts = (t - 1955)/35;
>> p = polyfit(ts,pop,2);
>> polyval(p,(2000-1955)/35)
ans =
  271.7400
```

Third order:

```
>> t = [t 1960];
>> pop = [pop 180.67];
>> ts = (t - 1955)/35;
>> p = polyfit(ts,pop,3);
>> polyval(p,(2000-1955)/35)
ans =
  273.9900
```

Fourth order:

```
>> t = [t 1950];
>> pop = [pop 152.27];
>> ts = (t - 1955)/35;
>> p = polyfit(ts,pop,4);
>> polyval(p,(2000-1955)/35)
ans =
  274.4200
```

Although the improvement is not great, the addition of each term causes the prediction for 2000 to increase. Thus, using higher-order approximations is moving the prediction closer to the actual value of 281.42 that occurred in 2000.

CHAPTER 15

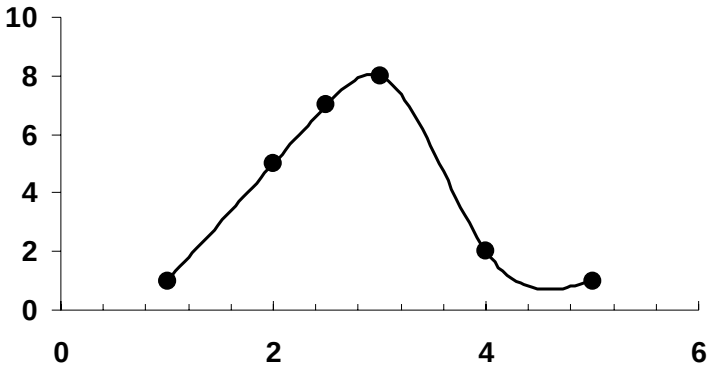
15.1 (a) The simultaneous equations for the natural spline can be set up as

$$\begin{bmatrix} 1 & & & & & \\ 1 & 3 & 0.5 & & & \\ & 0.5 & 2 & 0.5 & & \\ & & 0.5 & 3 & 1 & \\ & & & 1 & 4 & 1 \\ & & & & & 1 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ -6 \\ -24 \\ 15 \\ 0 \end{Bmatrix}$$

These equations can be solved for the *c*'s and then Eqs. (15.21) and (15.18) can be used to solve for the *b*'s and the *d*'s. The coefficients for the intervals can be summarized as

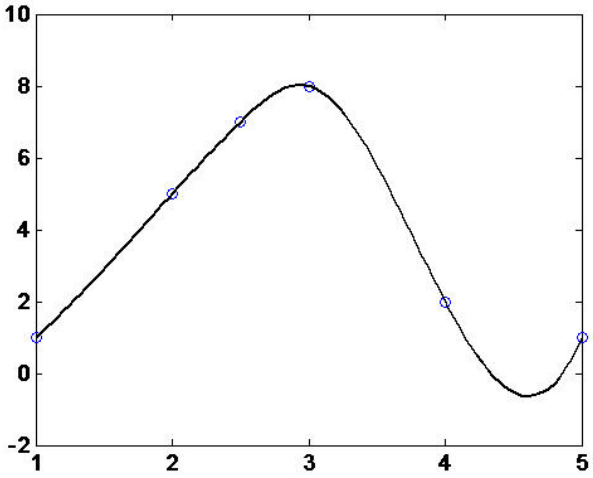
interval	*a*	*b*	*c*	*d*
1	1	3.970954	0	0.029046
2	5	4.058091	0.087137	-0.40664
3	7	3.840249	-0.52282	-6.31535
4	8	-1.41909	-9.99585	5.414938
5	2	-5.16598	6.248963	-2.08299

These can be used to generate the following plot of the natural spline:



(b) The not-a-knot spline and its plot can be generated with MATLAB as

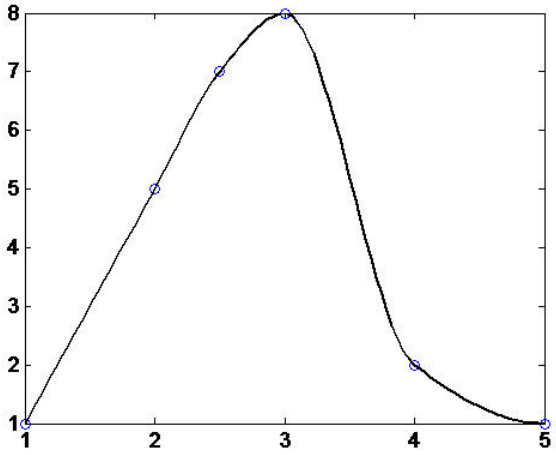
```
>> x = [1 2 2.5 3 4 5];
>> y = [1 5 7 8 2 1];
>> xx = linspace(1,5);
>> yy = spline(x,y,xx);
>> plot(x,y,'o',xx,yy)
```



Notice how the not-a-knot version exhibits much more curvature, particularly between the last points.

(c) The piecewise cubic Hermite polynomial and its plot can be generated with MATLAB as

```
>> x = [1 2 2.5 3 4 5];
>> y = [1 5 7 8 2 1];
>> xx = linspace(1,5);
>> yy = interp1(x,y,xx,'pchip');
>> plot(x,y,'o',xx,yy)
```



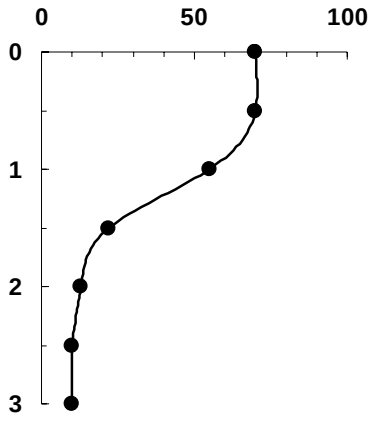
15.2 The simultaneous equations for the clamped spline with zero end slopes can be set up as

$$\begin{bmatrix} 1 & 0.5 & & & & & \\ 0.5 & 2 & 0.5 & & & & \\ & 0.5 & 2 & 0.5 & & & \\ & & 0.5 & 2 & 0.5 & & \\ & & & 0.5 & 2 & 0.5 & \\ & & & & 0.5 & 2 & 0.5 \\ & & & & & 0.5 & 1 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{Bmatrix} = \begin{Bmatrix} 0 \\ -90 \\ -108 \\ 144 \\ 36 \\ 18 \\ \end{Bmatrix}$$

These equations can be solved for the *c*'s and then Eqs. (15.21) and (15.18) can be used to solve for the *b*'s and the *d*'s. The coefficients for the intervals can be summarized as

interval	*a*	*b*	*c*	*d*
1	70	0	15.87692	-31.7538
2	70	-7.93846	-31.7538	-24.7385
3	55	-58.2462	-68.8615	106.7077
4	22	-47.0769	91.2	-66.0923
5	13	-5.44615	-7.93846	13.66154
6	10	-3.13846	12.55385	-12.5538

The fit can be displayed in graphical form. Note that we are plotting the points as depth versus temperature so that the graph depicts how the temperature changes down through the tank.



Inspection of the plot indicates that the inflection point occurs in the 3rd interval. The cubic equation for this interval is

$$T_3(x) = 55 - 58.2462(d-1) - 68.8615(d-1)^2 + 106.7077(d-1)^3$$

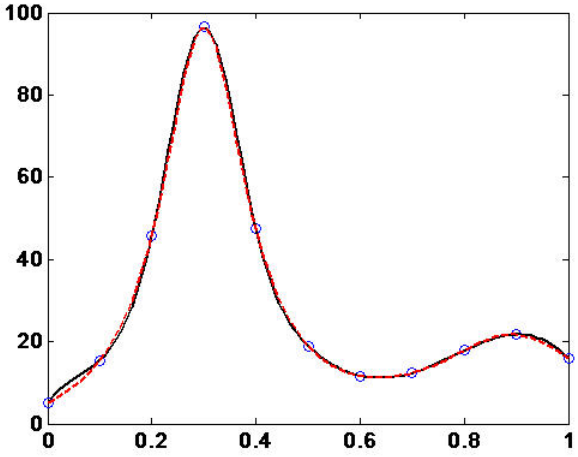
where T = temperature and d = depth. This equation can be differentiated twice to yield the second derivative

$$\frac{d^2T_3(x)}{dx^2} = -137.729 + 640.2462(d-1)$$

This can be set equal to zero and solved for the depth of the thermocline as d = 1.21511 m.

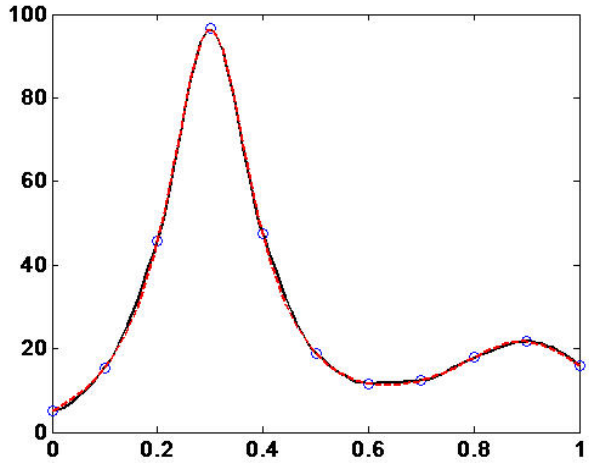
15.3 **(a)** The not-a-knot fit can be set up in MATLAB as

```
>> x = linspace(0,1,11);
>> y = 1./((x-0.3).^2+0.01)+1./((x-0.9).^2+0.04)-6;
>> xx = linspace(0,1);
>> yy = spline(x,y,xx);
>> yh = 1./((xx-0.3).^2+0.01)+1./((xx-0.9).^2+0.04)-6;
>> plot(x,y,'o',xx,yy,xx,yh,'--')
```



(b) The piecewise cubic Hermite polynomial fit can be set up in MATLAB as

```
>> x = linspace(0,1,11);
>> y = 1./((x-0.3).^2+0.01)+1./((x-0.9).^2+0.04)-6;
>> xx = linspace(0,1);
>> yy = interp1(x,y,xx,'pchip');
>> yh = 1./((xx-0.3).^2+0.01)+1./((xx-0.9).^2+0.04)-6;
>> plot(x,y,'o',xx,yy,xx,yh,'--')
```



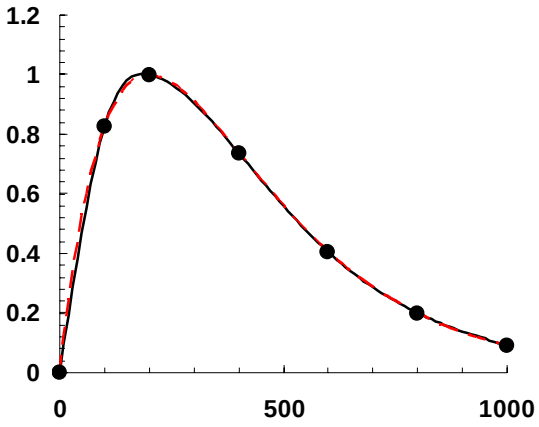
15.4 The simultaneous equations for the clamped spline with zero end slopes can be set up as

$$\begin{bmatrix} 1 & & & & & & \\ 100 & 400 & 100 & & & & \\ & 100 & 600 & 200 & & & \\ & & 200 & 800 & 200 & & \\ & & & 200 & 800 & 200 & \\ & & & & 200 & 800 & 200 \\ & & & & & & 1 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{Bmatrix} = \begin{Bmatrix} 0 \\ -0.01946 \\ -0.00923 \\ -0.00098 \\ 0.001843 \\ 0.001489 \\ 0 \end{Bmatrix}$$

These equations can be solved for the *c*'s and then Eqs. (15.21) and (15.18) can be used to solve for the *b*'s and the *d*'s. The coefficients for the intervals can be summarized as

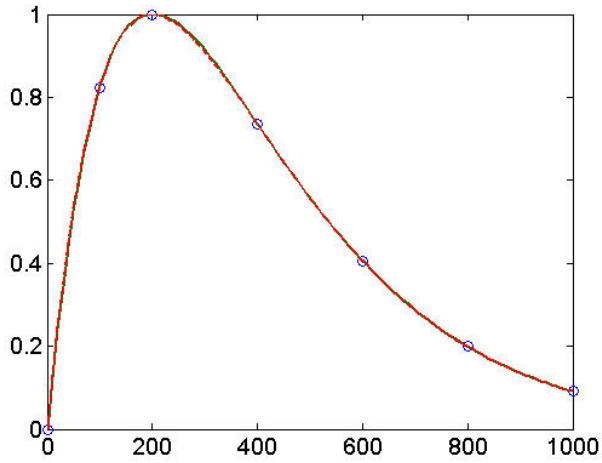
interval	*a*	*b*	*c*	*d*
1	0	0.009801	0	-1.6E-07
2	0.824361	0.005128	-4.7E-05	1.3E-07
3	1	-0.00031	-7.7E-06	1.31E-08
4	0.735759	-0.0018	2.13E-07	2.82E-09
5	0.406006	-0.00138	1.9E-06	-8.7E-10
6	0.199148	-0.00072	1.39E-06	-2.3E-09

The fit can be displayed in graphical form as



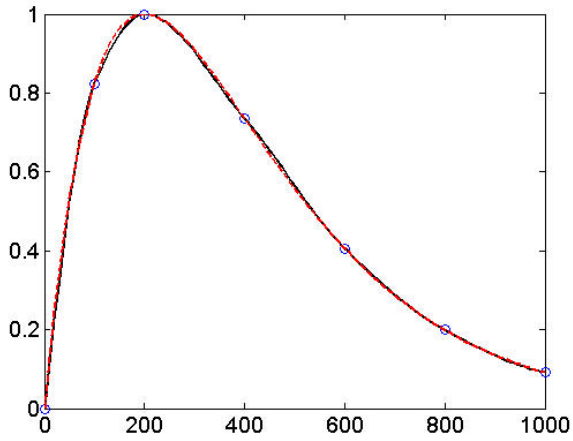
(b) The not-a-knot fit can be set up in MATLAB as

```
>> x = [0 100 200 400 600 800 1000];
>> y = x/200.*exp(-x/200+1);
>> xx = linspace(0,1000);
>> yc = xx/200.*exp(-xx/200+1);
>> yy = spline(x,y,xx);
>> plot(x,y,'o',xx,yy,xx,yc,'--')
```



(c) The piecewise cubic Hermite polynomial fit can be set up in MATLAB as

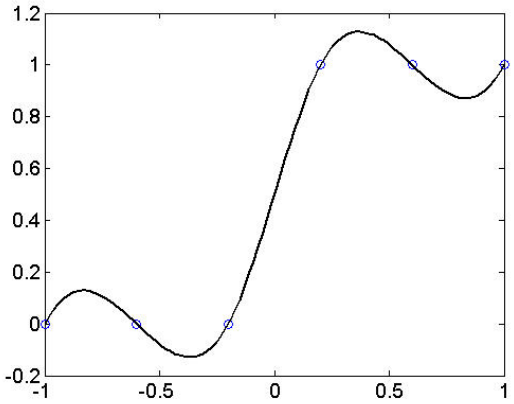
```
>> x = [0 100 200 400 600 800 1000];
>> y = x/200.*exp(-x/200+1);
>> xx = linspace(0,1000);
>> yc = xx/200.*exp(-xx/200+1);
>> yy = interp1(x,y,xx,'pchip');
>> plot(x,y,'o',xx,yy,xx,yc,'--')
```



Summary: For this case, the not-a-knot fit is the best.

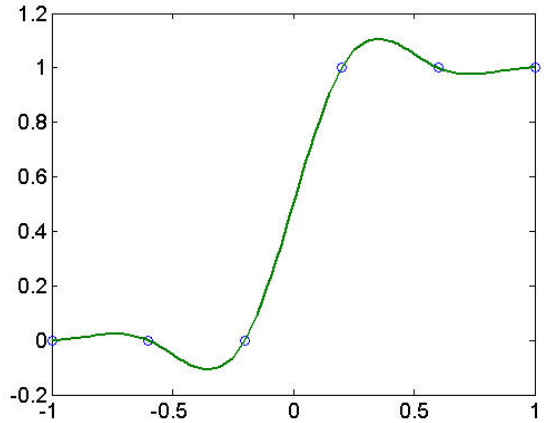
15.5 **(a)** The not-a-knot fit can be set up in MATLAB as

```
>> x = [-1 -0.6 -0.2 0.2 0.6 1];
>> y = [0 0 0 1 1 1];
>> xx = linspace(-1,1);
>> yy = spline(x,y,xx);
>> plot(x,y,'o',xx,yy)
```



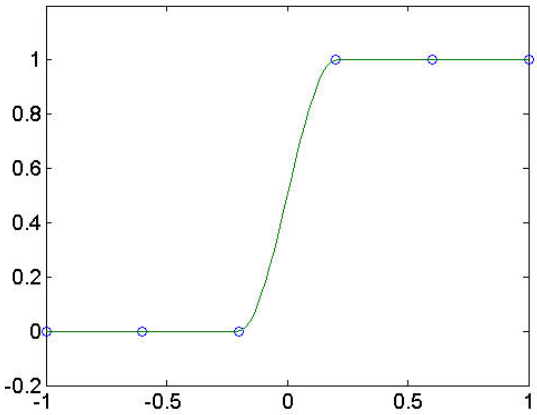
(b) The clamped spline with zero end slopes can be set up in MATLAB as

```
>> x = [-1 -0.6 -0.2 0.2 0.6 1];
>> y = [0 0 0 1 1 1];
>> ys = [0 y 0];
>> xx = linspace(-1,1);
>> yy = spline(x,ys,xx);
>> plot(x,y,'o',xx,yy)
```



(c) The piecewise cubic Hermite polynomial fit can be set up in MATLAB as

```
>> x = [-1 -0.6 -0.2 0.2 0.6 1];
>> y = [0 0 0 1 1 1];
>> xx = linspace(-1,1);
>> yy = interp1(x,y,xx,'pchip');
>> plot(x,y,'o',xx,yy)
```

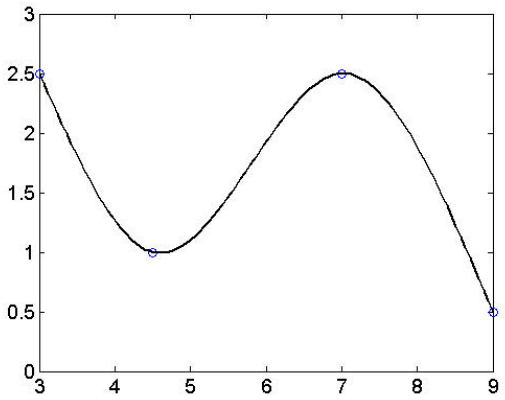


15.6 An M-file function to implement the natural spline can be written as

```
function yy = natspline(x,y,xx)
% natspline(x,y,xx):
%   uses a natural cubic spline interpolation to find yy, the values
%   of the underlying function y at the points in the vector xx.
%   The vector x specifies the points at which the data y is given.

n = length(x);
m = length(xx);
aa(1,1) = 1; aa(n,n) = 1;
bb(1) = 0; bb(n) = 0;
for i = 2:n-1
  aa(i,i-1) = h(x, i - 1);
  aa(i,i) = 2 * (h(x, i - 1) + h(x, i));
  aa(i,i+1) = h(x, i);
  bb(i) = 3 * (fd(i + 1, i, x, y) - fd(i, i - 1, x, y));
end
c = aa\bb';
for i = 1:n - 1
  a(i) = y(i);
  b(i) = fd(i + 1, i, x, y) - h(x, i) / 3 * (2 * c(i) + c(i + 1));
  d(i) = (c(i + 1) - c(i)) / 3 / h(x, i);
end
for i = 1:m
  yy(i) = SplineInterp(x, n, a, b, c, d, xx(i));
end

function hh = h(x, i)
hh = x(i + 1) - x(i);

function fdd = fd(i, j, x, y)
fdd = (y(i) - y(j)) / (x(i) - x(j));

function yyy = SplineInterp(x, n, a, b, c, d, xi)
for ii = 1:n - 1
  if xi >= x(ii) - 0.000001 & xi <= x(ii + 1) + 0.000001
    yyy=a(ii)+b(ii)*(xi-x(ii))+c(ii)*(xi-x(ii))^2+d(ii)*(xi-x(ii))^3;
    break
  end
end
```

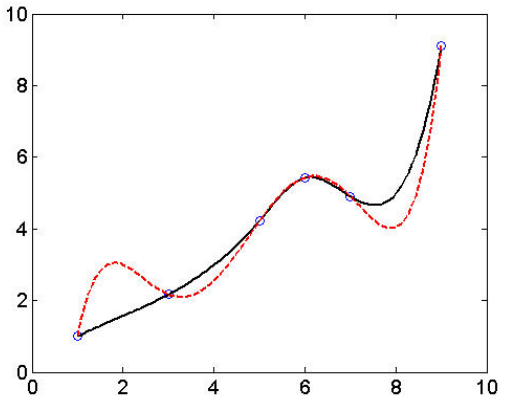
The program can be used to duplicate Example 15.3:

```
>> x = [3 4.5 7 9];
>> y = [2.5 1 2.5 .5];
>> xx = linspace(3,9);
>> yy = natspline(x,y,xx);
>> plot(x,y,'o',xx,yy)
```



15.7 (a) The not-a-knot fit can be set up in MATLAB as

```
>> x = [1 3 5 6 7 9];
>> y = 0.0185*x.^5-0.444*x.^4+3.9125*x.^3-15.456*x.^2+27.069*x-14.1;
>> xx = linspace(1,9);
>> yy = spline(x,y,xx);
>> yc = 0.0185*xx.^5-0.444*xx.^4+3.9125*xx.^3-15.456*xx.^2+27.069*xx-14.1;
>> plot(x,y,'o',xx,yy,xx,yc,'--')
```

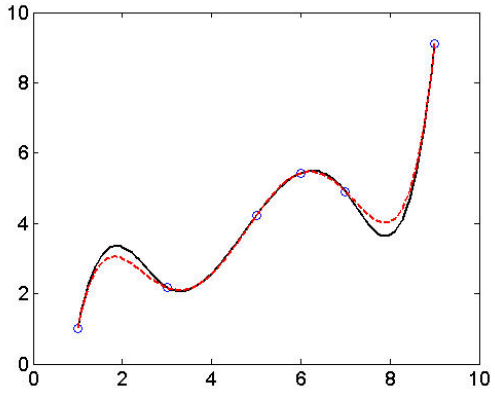


(b) The function can be differentiated to give

$$f'(x) = 0.0925x^4 - 1.776x^3 + 11.7375x^2 - 30.912x + 27.069$$

This function can be evaluated at the end nodes to give $f'(1) = 6.211$ and $f'(9) = 11.787$. These values can then be added to the y vector and the `spline` function invoked to develop the clamped fit:

```
>> yd = [6.211 y 11.787];
>> yy = spline(x,yd,xx);
>> plot(x,y,'o',xx,yy,xx,yc,'--')
```



CHAPTER 16

16.1 A table of integrals can be consulted to determine

$$\int \tanh dx = \frac{1}{a} \ln \cosh ax$$

Therefore,

$$\int_0^t \sqrt{\frac{gm}{c_d}} \tanh\left(\sqrt{\frac{gc_d}{m}}t\right) dt = \sqrt{\frac{gm}{c_d}} \sqrt{\frac{m}{gc_d}} \left[\ln \cosh\left(\sqrt{\frac{gc_d}{m}}t\right)\right]_0^t$$

$$\sqrt{\frac{gm^2}{gc_d^2}} \left[\ln \cosh\left(\sqrt{\frac{gc_d}{m}}t\right) - \ln \cosh(0)\right]$$

Since cosh(0) = 1 and ln(1) = 0, this reduces to

$$\frac{m}{c_d} \ln \cosh\left(\sqrt{\frac{gc_d}{m}}t\right)$$

16.2 **(a)** The analytical solution can be evaluated as

$$\int_0^4 (1 - e^{-2x})\, dx = \left[x + 0.5e^{-2x}\right]_0^4 = 4 + 0.5e^{-2(4)} - 0 - 0.5e^{-2(0)} = 3.500167731$$

(b) single application of the trapezoidal rule

$$(4-0)\frac{0 + 0.999665}{2} = 1.99329 \quad (\varepsilon_t = 42.88\%)$$

(c) composite trapezoidal rule

<u>*n* = 2:</u>

$$(4-0)\frac{0 + 2(0.981684) + 0.999665}{4} = 2.96303 \quad (\varepsilon_t = 15.35\%)$$

<u>*n* = 4:</u>

$$(4-0)\frac{0 + 2(0.86466 + 0.981684 + 0.99752) + 0.999665}{8} = 3.3437 \quad (\varepsilon_t = 4.47\%)$$

(d) single application of Simpson's 1/3 rule

$$(4-0)\frac{0+4(0.981684)+0.999665}{6}=3.28427 \quad (\varepsilon_t=6.17\%)$$

(e) composite Simpson's 1/3 rule (n = 4)

$$(4-0)\frac{0+4(0.86466+0.99752)+2(0.981684)+0.999665}{12}=3.47059 \quad (\varepsilon_t=0.84\%)$$

(f) Simpson's 3/8 rule.

$$(4-0)\frac{0+3(0.930517+0.995172)+0.999665}{8}=3.388365 \quad (\varepsilon_t=3.19\%)$$

16.3 (a) The analytical solution can be evaluated as

$$\int_0^{\pi/2}(6+3\cos x)\,dx=\left[6x+3\sin x\right]_0^{\pi/2}=6(\pi/2)+3\sin(\pi/2)-6(0)-3\sin(0)=12.424778$$

(b) single application of the trapezoidal rule

$$\left(\frac{\pi}{2}-0\right)\frac{9+6}{2}=11.78097 \quad (\varepsilon_t=5.18\%)$$

(c) composite trapezoidal rule

<u>n = 2:</u>

$$\left(\frac{\pi}{2}-0\right)\frac{9+2(8.12132)+6}{4}=12.26896 \quad (\varepsilon_t=1.25\%)$$

<u>n = 4:</u>

$$\left(\frac{\pi}{2}-0\right)\frac{9+2(8.77164+8.12132+7.14805)+6}{8}=12.386125 \quad (\varepsilon_t=0.3111\%)$$

(d) single application of Simpson's 1/3 rule

$$\left(\frac{\pi}{2}-0\right)\frac{9+4(8.12132)+6}{6}=12.4316 \quad (\varepsilon_t=0.0550\%)$$

(e) composite Simpson's 1/3 rule (n = 4)

$$\left(\frac{\pi}{2}-0\right)\frac{9+4(8.7716+7.14805)+2(8.12132)+6}{12}=12.42518 \quad (\varepsilon_t=0.0032\%)$$

(f) Simpson's 3/8 rule.

$$\left(\frac{\pi}{2}-0\right)\frac{9+3(8.59808+7.5)+6}{8}=12.42779 \quad (\varepsilon_t = 0.0243\%)$$

16.4 **(a)** The analytical solution can be evaluated as

$$\int_{-2}^{4} (1-x-4x^3+2x^5)\,dx = \left[x-\frac{x^2}{2}-x^4+\frac{x^6}{3}\right]_{-2}^{4}$$

$$= 4-\frac{4^2}{2}-4^4+\frac{4^6}{3}-(-2)+\frac{(-2)^2}{2}+(-2)^4-\frac{(-2)^6}{3}=1104$$

(b) single application of the trapezoidal rule

$$(4-(-2))\frac{-29+1789}{2}=5280 \quad (\varepsilon_t = 378.3\%)$$

(c) composite trapezoidal rule

<u>*n* = 2:</u>

$$(4-(-2))\frac{-29+2(-2)+1789}{4}=2634 \quad (\varepsilon_t = 138.6\%)$$

<u>*n* = 4:</u>

$$(4-(-2))\frac{-29+2(1.9375+(-2)+131.3125)+1789}{8}=1516.875 \quad (\varepsilon_t = 37.4\%)$$

(d) single application of Simpson's 1/3 rule

$$(4-(-2))\frac{-29+4(-2)+1789}{6}=1752 \quad (\varepsilon_t = 58.7\%)$$

(e) composite Simpson's 1/3 rule (*n* = 4)

$$(4-(-2))\frac{-29+4(1.9375+131.3125)+2(-2)+1789}{12}=1144.5 \quad (\varepsilon_t = 3.6685\%)$$

(f) Simpson's 3/8 rule.

$$(4-(-2))\frac{-29+3(1+31)+1789}{8}=1392 \quad (\varepsilon_t = 26.09\%)$$

(g) Boole's rule.

$$(4-(-2))\frac{7(-29)+32(1.9375)+12(-2)+32(131.3125)+7(1789)}{90}=1104 \quad (\varepsilon_t = 0\%)$$

16.5 (a) The analytical solution can be evaluated as

$$\int_0^{1.2} e^{-x}\,dx = \left[-e^{-x}\right]_0^{1.2} = -e^{-1.2} - (-e^0) = 0.69880579$$

(b) Trapezoidal rule

$$(0.1-0)\frac{1+0.90484}{2} + (0.3-0.1)\frac{0.90484+0.74082}{2} + (0.5-0.3)\frac{0.74082+0.60653}{2}$$

$$+ (0.7-0.5)\frac{0.60653+0.49659}{2} + (0.95-0.7)\frac{0.49659+0.38674}{2} + (1.2-0.957)\frac{0.38674+0.30119}{2}$$

$$= 0.09524 + 0.164566 + 0.134735 + 0.110312 + 0.110416 + 0.085992 = 0.70126 \quad (\varepsilon_t = 0.35\%)$$

(c) Trapezoidal and Simpson's Rules

$$(0.1-0)\frac{1+0.90484}{2} + (0.7-0.1)\frac{0.90484+3(0.74082+0.60653)+0.49659}{8}$$

$$+ (1.2-0.7)\frac{0.49659+4(0.38674)+0.30119}{6} = 0.09524 + 0.40826 + 0.195395 = 0.698897 \quad (\varepsilon_t = 0.0131\%)$$

16.6 (a) The integral can be evaluated analytically as,

$$\int_{-2}^{2}\left[\frac{x^3}{3} - 3y^2x + y^3\frac{x^2}{2}\right]_0^4 dy$$

$$\int_{-2}^{2}\frac{(4)^3}{3} - 3y^2(4) + y^3\frac{(4)^2}{2}\,dy$$

$$\int_{-2}^{2} 21.33333 - 12y^2 + 8y^3\,dy$$

$$\left[21.33333y - 4y^3 + 2y^4\right]_{-2}^{2}$$

$$21.33333(2) - 4(2)^3 + 2(2)^4 - 21.33333(-2) + 4(-2)^3 - 2(-2)^4 = 21.33333$$

(b) The composite trapezoidal rule with $n = 2$ can be used the evaluate the inner integral at the three equispaced values of y,

$y = -2$: $\quad (4-0)\dfrac{-12 + 2(-24) - 28}{4} = -88$

$y = 0$: $\quad (4-0)\dfrac{0 + 2(4) + 16}{4} = 24$

$y = 2$: $\quad (4-0)\dfrac{-12+2(8)+36}{4} = 40$

These results can then be integrated in y to yield

$$(2-(-2))\frac{-88+2(24)+40}{4} = 0$$

which represents a percent relative error of

$$\varepsilon_t = \left|\frac{21.33333-0}{21.33333}\right| \times 100\% = 100\%$$

which is not very good.

(c) Single applications of Simpson's 1/3 rule can be used the evaluate the inner integral at the three equispaced values of y,

$y = -2$: $\quad (4-0)\dfrac{-12+4(-24)-28}{6} = -90.66667$

$y = 0$: $\quad (4-0)\dfrac{0+4(4)+16}{6} = 21.33333$

$y = 2$: $\quad (4-0)\dfrac{-12+4(8)+36}{6} = 37.33333$

These results can then be integrated in y to yield

$$(2-(-2))\frac{-90.66667+4(21.33333)+37.33333}{6} = 21.33333$$

which represents a percent relative error of

$$\varepsilon_t = \left|\frac{21.33333-21.33333}{21.33333}\right| \times 100\% = 0\%$$

which is perfect

16.7 (a) The integral can be evaluated analytically as,

$$\int_{-4}^{4}\int_{0}^{6}\left[\frac{x^4}{4} - 2yzx\right]_{-1}^{3} dy\,dz = \int_{-4}^{4}\int_{0}^{6} 20 - 8yz\,dy\,dz$$

$$\int_{-4}^{4}\int_{0}^{6} 20 - 8yz\,dy\,dz = \int_{-4}^{4}\left[20y - 4zy^2\right]_{0}^{6} dz = \int_{-4}^{4} 120 - 144z\,dz$$

$$\int_{-4}^{4} 120 - 144z\, dz = \left[120z - 72z^2\right]_{-4}^{4} = 120(4) - 72(4)^2 - 120(-4) + 72(-4)^2 = 960$$

(b) Single applications of Simpson's 1/3 rule can be used the evaluate the inner integral at the three equispaced values of *y* for each value of *z*,

<u>*z* = –4:</u>

y = 0: $\quad (3-(-1))\dfrac{-1+4(1)+27}{6} = 20$

y = 3: $\quad (3-(-1))\dfrac{23+4(25)+51}{6} = 116$

y = 6: $\quad (3-(-1))\dfrac{47+4(49)+75}{6} = 212$

These results can then be integrated in *y* to yield

$$(6-0)\frac{20+4(116)+212}{6} = 696$$

<u>*z* = 0:</u>

y = 0: $\quad (3-(-1))\dfrac{-1+4(1)+27}{6} = 20$

y = 3: $\quad (3-(-1))\dfrac{-1+4(1)+27}{6} = 20$

y = 6: $\quad (3-(-1))\dfrac{-1+4(1)+27}{6} = 20$

These results can then be integrated in *y* to yield

$$(6-0)\frac{20+4(20)+20}{6} = 120$$

<u>*z* = 4:</u>

y = 0: $\quad (3-(-1))\dfrac{-1+4(1)+27}{6} = 20$

y = 3: $\quad (3-(-1))\dfrac{-25+4(-23)+3}{6} = -76$

$y = 6$: $\quad (3-(-1))\dfrac{-49+4(-47)-21}{6} = -172$

These results can then be integrated in y to yield

$$(6-0)\frac{20+4(-76)-172}{6} = -456$$

The results of the integrations in y can then be integrated in z to yield

$$(6-0)\frac{696+4(120)-456}{6} = 960$$

which represents a percent relative error of

$$\varepsilon_t = \left|\frac{960-960}{960}\right| \times 100\% = 0\%$$

16.8 **(a)** The trapezoidal rule can be implemented as,

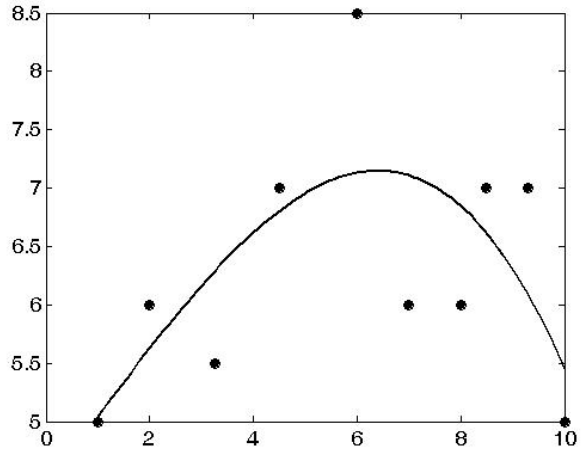
$$d = (2-1)\frac{5+6}{2} + (3.25-2)\frac{6+5.5}{2} + (4.5-3.25)\frac{5.5+7}{2} + (6-4.5)\frac{7+8.5}{2}$$
$$+ (7-6)\frac{8.5+6}{2} + (8-7)\frac{6+6}{2} + (8.5-8)\frac{6+7}{2} + (9.3-8.5)\frac{7+7}{2} + (10-9.3)\frac{7+5}{2} = 58.425$$

(b) The polynomial can be fit as,

```
>> format long
>> t = [1 2 3.25 4.5 6 7 8 8.5 9.3 10];
>> v = [5 6 5.5 7 8.5 6 6 7 7 5];
>> p = polyfit(t,v,3)

p =
  -0.00657842294444   0.01874733808337   0.56859435273356
4.46645555949356
```

The cubic can be plotted along with the data,

```
>> tt = linspace(1,10);
>> vv = polyval(p,tt);
>> plot(tt,vv,t,v,'o')
```



The cubic can then be integrated to estimate the distance traveled,

$$d = \int_1^{10} -0.006578t^3 + 0.018747t^2 + 0.568594t + 4.46646\,dt$$

$$= \left[-0.001645t^4 + 0.006249t^3 + 0.284297t^2 + 4.46646t\right]_1^{10} = 58.14199$$

16.9

z	$w(z)$	$\rho g w(z)(60 - z)$	$\rho g z w(z)(60 - z)$
60	200	0	0
50	190	1.8639E+07	9.3195E+08
40	175	3.4335E+07	1.3734E+09
30	160	4.7088E+07	1.4126E+09
20	135	5.2974E+07	1.0595E+09
10	130	6.3765E+07	6.3765E+08
0	122	7.1809E+07	0

$$f_t = 60\frac{7.1809 + 4(6.3765 + 4.7088 + 1.8639) + 2(5.2974 + 3.4335) + 0}{3(6)} \times 10^7 = 2.5480 \times 10^9$$

$$\int_0^D \rho g z w(z)(D - z)\,dz = 60\frac{0 + 4(0.63765 + 1.4126 + 0.93195) + 2(1.0595 + 1.3734) + 0}{3(6)} \times 10^9 = 5.5982 \times 10^{10}$$

$$d = \frac{5.5982 \times 10^{10}}{2.5480 \times 10^9} = 21.971$$

16.10 (a) Trapezoidal rule:

$$f = 30\frac{0 + 2(54.937 + 51.129 + 36.069 + 27.982 + 19.455) + 13.311}{2(6)} = 996.1363$$

$$f = \frac{30\dfrac{0 + 2(274.684 + 511.292 + 586.033 + 559.631 + 486.385) + 399.332}{2(6)}}{996.1363} = \frac{13088.45}{996.1363} = 13.139\,\text{m}$$

(b) Simpson's 1/3 rule:

$$f = 30\frac{0 + 4(54.937 + 36.069 + 19.455) + 2(51.129 + 27.982) + 13.311}{3(6)} = 1042.294$$

$$f = \frac{30\dfrac{0 + 4(274.684 + 586.033 + 486.385) + 2(511.292 + 559.631) + 399.332}{3(6)}}{1042.294} = \frac{13215.97}{1042.294} = 12.6797\,\text{m}$$

CHAPTER 17

17.1 The integral can be evaluated analytically as,

$$I = \int_1^2 \left(2x + \frac{3}{x}\right)^2 dx = \int_1^2 4x^2 + 12 + 9x^{-2}\, dx$$

$$I = \left[\frac{4x^3}{3} + 12x - \frac{9}{x}\right]_1^2 = \frac{4(2)^3}{3} + 12(2) - \frac{9}{2} - \frac{4(1)^3}{3} - 12(1) + \frac{9}{1} = 25.8333$$

The tableau depicting the implementation of Romberg integration to ε_s = 0.5% is

iteration →	1	2	3
ε_t →	6.9355%	0.1613%	0.0048%
ε_a →		1.6908%	0.0098%
1	27.62500000	25.87500000	25.83456463
2	26.31250000	25.83709184	
4	25.95594388		

Thus, the result is 25.83456.

17.2 (a) The integral can be evaluated analytically as,

$$I = \left[-0.01094x^5 + 0.21615x^4 - 1.3854x^3 + 3.14585x^2 + 2x\right]_0^8 = 34.87808$$

(b) The tableau depicting the implementation of Romberg integration to ε_s = 0.5% is

iteration →	1	2	3	4
ε_t →	20.1699%	42.8256%	0.0000%	0.0000%
ε_a →		9.9064%	2.6766%	0.000000%
1	27.84320000	19.94133333	34.87808000	34.87808000
2	21.91680000	33.94453333	34.87808000	
4	30.93760000	34.81973333		
8	33.84920000			

Thus, the result is exact.

(c) The transformations can be computed as

$$x = \frac{(8+0) + (8-0)x_d}{2} = 4 + 4x_d \qquad dx = \frac{8-0}{2} dx_d = 4dx_d$$

These can be substituted to yield

$$I = \int_{-1}^{1} \left[-0.0547(4 + 4x_d)^4 + 0.8646(4 + 4x_d)^3 - 4.1562(4 + 4x_d)^2 + 6.2917(4 + 4x_d) + 2\right] 4dx_d$$

The transformed function can be evaluated using the values from Table 17.1

$$I = 0.5555556\,f(-0.774596669) + 0.8888889\,f(0) + 0.5555556\,f(0.774596669) = 34.87808$$

which is exact.

(d)

```
>> format long
>> y = inline('-0.0547*x.^4+0.8646*x.^3-4.1562*x.^2+6.2917*x+2');
>> I = quad(y,0,8)

I =
  34.87808000000000
```

17.3 Although it's not required, the analytical solution can be evaluated simply as

$$I = \int_0^3 xe^x\,dx = \left[e^x(x-1)\right]_0^3 = 41.17107385$$

(a) The tableau depicting the implementation of Romberg integration to ε_s = 0.5% is

iteration →	1	2	3
ε_t →	119.5350%	5.8349%	0.1020%
ε_a →		26.8579%	0.3579%
1	90.38491615	43.57337260	41.21305531
2	55.27625849	41.36057514	
4	44.83949598		

which represents a percent relative error of 0.102 %.

(b) The transformations can be computed as

$$x = \frac{(3+0)+(3-0)x_d}{2} = 1.5 + 1.5x_d \qquad dx = \frac{3-0}{2}dx_d = 1.5dx_d$$

These can be substituted to yield

$$I = \int_{-1}^{1} \left[(1.5 + 1.5x_d)e^{1.5+1.5x_d}\right] 1.5dx_d$$

The transformed function can be evaluated using the values from Table 17.1

$$I = f(-0.577350269) + f(0.577350269) = 39.6075058$$

which represents a percent relative error of 3.8 %.

(c) Using MATLAB

```
>> format long
>> I = quad(inline('x.*exp(x)'),0,3)
```

```
I =
  41.17107385090233
```

which represents a percent relative error of 1.1×10^{-8} %.

```
>> I = quadl(inline('x.*exp(x)'),0,3)

I =
  41.17107466800178
```

which represents a percent relative error of 2×10^{-6} %.

17.4 The exact solution can be evaluated simply as

```
>> format long
>> erf(1.5)

ans =
   0.96610514647531
```

(a) The transformations can be computed as

$$x = \frac{(1.5+0)+(1.5-0)x_d}{2} = 0.75 + 0.75x_d \qquad dx = \frac{1.5-0}{2}dx_d = 0.75dx_d$$

These can be substituted to yield

$$I = \frac{2}{\sqrt{\pi}}\int_{-1}^{1}\left[e^{-(0.75+0.75x_d)^2}\right]0.75dx_d$$

The transformed function can be evaluated using the values from Table 17.1

$$I = f(-0.577350269) + f(0.577350269) = 0.974173129$$

which represents a percent relative error of 0.835 %.

(b) The transformed function can be evaluated using the values from Table 17.1

$$I = 0.5555556\,f(-0.774596669) + 0.8888889\,f(0) + 0.5555556\,f(0.774596669) = 0.965502083$$

which represents a percent relative error of 0.062 %.

17.5 (a) The tableau depicting the implementation of Romberg integration to ε_s = 0.5% is

iteration →	1	2	3	4
ε_a →		19.1131%	1.0922%	0.035826%
1	199.66621287	847.93212300	1027.49455856	1051.60670352
2	685.86564547	1016.27190634	1051.22995126	
4	933.67034112	1049.04507345		
8	1020.20139037			

Note that if 8 iterations are implemented, the method converges on a value of 1053.38523686. This result is also obtained if you use the composite Simpson's 1/3 rule with 1024 segments.

(b) The transformations can be computed as

$$x = \frac{(30+0)+(30-0)x_d}{2} = 15 + 15x_d \qquad dx = \frac{30-0}{2} dx_d = 15dx_d$$

These can be substituted to yield

$$I = 200\int_{-1}^{1} \left[\frac{15+15x_d}{22+15x_d} e^{-2.5(15+15x_d)/30} \right] 15dx_d$$

The transformed function can be evaluated using the values from Table 17.1

$$I = f(-0.577350269) + f(0.577350269) = 1162.93396$$

(c) Interestingly, the `quad` function encounters a problem and exceeds the maximum number of iterations

```
>> format long
>> I = quad(inline('200*x/(7+x)*exp(-2.5*x/30)'),0,30)
Warning: Maximum function count exceeded; singularity likely.
(Type "warning off MATLAB:quad:MaxFcnCount" to suppress this
warning.)
> In quad at 88

I =
    1.085280043451920e+003
```

The `quadl` function converges rapidly, but does not yield a very accurate result:

```
>> I = quadl(inline('200*x/(7+x)*exp(-2.5*x/30)'),0,30)

I =
    1.055900924411335e+003
```

17.6 The integral to be evaluated is

$$I = \int_0^{1/2} \left(10e^{-t} \sin 2\pi t\right)^2 dt$$

(a) The tableau depicting the implementation of Romberg integration to ε_s = 0.1% is

iteration →	1	2	3	4
ε_a →		25.0000%	2.0824%	0.025340%
1	0.00000000	20.21768866	15.16502516	15.41501768
2	15.16326649	15.48081663	15.41111155	
4	15.40142910	15.41546811		

8	15.41195836

(b) The transformations can be computed as

$$x = \frac{(0.5+0)+(0.5-0)x_d}{2} = 0.25 + 0.25x_d \qquad dx = \frac{0.5-0}{2}dx_d = 0.25dx_d$$

These can be substituted to yield

$$I = \int_{-1}^{1} \left[10e^{-(0.25+0.25x_d)} \sin 2\pi(0.25+0.25x_d)\right]^2 0.25dx_d$$

For the two-point application, the transformed function can be evaluated using the values from Table 17.1

$$I = f(-0.577350269) + f(0.577350269) = 7.684096 + 4.313728 = 11.99782$$

For the three-point application, the transformed function can be evaluated using the values from Table 17.1

$$I = 0.5555556\, f(-0.774596669) + 0.8888889\, f(0) + 0.5555556\, f(0.774596669)$$

$$= 0.5555556(1.237449) + 0.8888889(15.16327) + 0.5555556(2.684915) = 15.65755$$

(c)

```
>> format long
>> I = quad(inline('(10*exp(-x).*sin(2*pi*x)).^2'),0,0.5)

I =
  15.41260804934509
```

17.7 The integral to be evaluated is

$$I = \int_0^{0.75} 10\left(1 - \frac{r}{0.75}\right)^{1/7} 2\pi r\, dr$$

(a) The tableau depicting the implementation of Romberg integration to ε_s = 0.1% is

iteration →	1	2	3	4
ε_a →		25.0000%	1.0725%	0.098313%
1	0.00000000	10.67030554	12.88063803	13.74550712
2	8.00272915	12.74249225	13.73199355	
4	11.55755148	13.67014971		
8	13.14200015			

(b) The transformations can be computed as

$$x = \frac{(0.75+0)+(0.75-0)x_d}{2} = 0.375 + 0.375x_d \qquad dx = \frac{0.75-0}{2}dx_d = 0.375dx_d$$

These can be substituted to yield

$$I = \int_{-1}^{1} \left[10\left(1 - \frac{0.375 + 0.375x_d}{0.75}\right)^{1/7} 2\pi(0.375 + 0.375x_d) \right]^2 0.375 dx_d$$

For the two-point application, the transformed function can be evaluated using the values from Table 17.1

$$I = f(-0.577350269) + f(0.577350269) = 14.77171$$

(c)

```
>> format long
>> I = quad(inline('10*(1-x/0.75).^(1/7)*2*pi.*x'),0,0.75)

I =
  14.43168560836254
```

17.8 The integral to be evaluated is

$$I = \int_2^8 (9 + 4\cos^2 0.4t)(5e^{-0.5t} + 2e^{0.15t})\, dt$$

(a) The tableau depicting the implementation of Romberg integration to ε_s = 0.1% is

iteration →	1	2	3	4
ε_a →		7.4179%	0.1054%	0.001212%
1	411.26095167	317.15529472	322.59571622	322.34570788
2	340.68170896	322.25568988	322.34961426	
4	326.86219465	322.34374398		
8	323.47335665			

(b)

```
>> format long
>> y = inline('(9+4*cos(0.4*x).^2).*(5*exp(-0.5*x)+2*exp(0.15*x))')
>> I = quadl(y,2,8)

I =
    3.223483672542467e+002
```

17.9 **(a)** The integral can be evaluated analytically as,

$$\int_{-2}^{2} \left[\frac{x^3}{3} - 3y^2 x + y^3 \frac{x^2}{2} \right]_0^4 dy$$

$$\int_{-2}^{2} \frac{(4)^3}{3} - 3y^2(4) + y^3 \frac{(4)^2}{2}\, dy$$

$$\int_{-2}^{2} 21.33333 - 12y^2 + 8y^3\, dy$$

$$\left[21.33333y - 4y^3 + 2y^4\right]_{-2}^{2}$$

$$21.33333(2) - 4(2)^3 + 2(2)^4 - 21.33333(-2) + 4(-2)^3 - 2(-2)^4 = 21.33333$$

(b) The operation of the `dblquad` function can be understood by invoking `help`,

```
>> help dblquad
```

A session to use the function to perform the double integral can be implemented as,

```
>> dblquad(inline('x.^2-3*y.^2+x*y.^3'),0,4,-2,2)

ans =
   21.3333
```

CHAPTER 18

18.1 **(a)** The analytical solution can be derived by the separation of variables,

$$\int \frac{dy}{y} = \int t^3 - 1.5\, dt$$

The integrals can be evaluated to give,

$$\ln y = \frac{t^4}{4} - 1.5t + C$$

Substituting the initial conditions yields $C = 0$. Substituting this value and taking the exponential gives

$$y = e^{t^4/4 - 1.5t}$$

(b) Euler method ($h = 0.5$):

t	*y*	*dy/dt*
0	**1**	-1.5
0.5	**0.25**	-0.34375
1	**0.078125**	-0.03906
1.5	**0.058594**	0.109863
2	**0.113525**	

Euler method ($h = 0.25$):

t	*y*	*dy/dt*
0	**1**	-1.5
0.25	**0.625**	-0.92773
0.5	**0.393066**	-0.54047
0.75	**0.25795**	-0.2781
1	**0.188424**	-0.09421
1.25	**0.164871**	0.074707
1.5	**0.183548**	0.344153
1.75	**0.269586**	1.040434
2	**0.529695**	

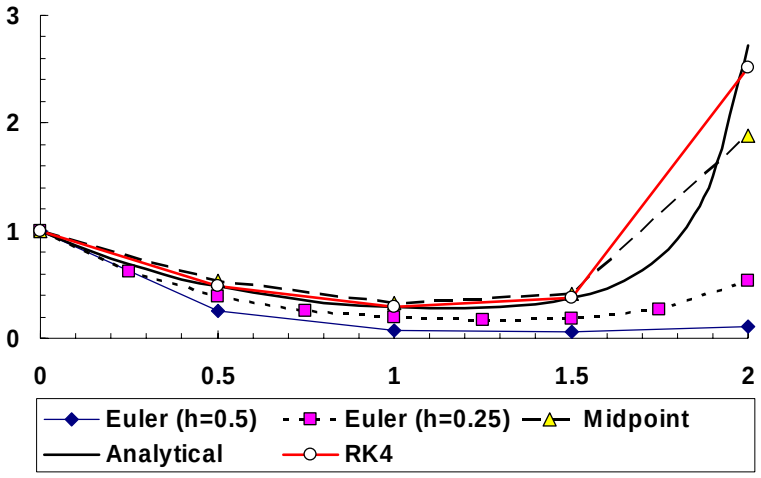
(c) Midpoint method ($h = 0.5$)

t	*y*	*dy/dt*	t_m	y_m	dy_m/dt
0	**1**	-1.5	0.25	0.625	-0.92773
0.5	**0.536133**	-0.73718	0.75	0.351837	-0.37932
1	**0.346471**	-0.17324	1.25	0.303162	0.13737
1.5	**0.415156**	0.778417	1.75	0.60976	2.353292
2	**1.591802**				

(d) RK4 (h = 0.5)

t	y	k_1	t_m	y_m	k_2	t_m	y_m	k_3	t_e	y_e	k_4	ϕ
0	**1.0000**	-1.5000	0.25	0.6250	-0.9277	0.25	0.7681	-1.1401	0.5	0.4300	-0.5912	-1.0378
0.5	**0.4811**	-0.6615	0.75	0.3157	-0.3404	0.75	0.3960	-0.4269	1	0.2676	-0.1338	-0.3883
1	**0.2869**	-0.1435	1.25	0.2511	0.1138	1.25	0.3154	0.1429	1.5	0.3584	0.6720	0.1736
1.5	**0.3738**	0.7008	1.75	0.5489	2.1186	1.75	0.9034	3.4866	2	2.1170	13.7607	4.2786
2	**2.5131**											

All the solutions can be presented graphically as



18.2 **(a)** The analytical solution can be derived by the separation of variables,

$$\int \frac{dy}{\sqrt{y}} = \int 1 + 2x\, dx$$

The integrals can be evaluated to give,

$$2\sqrt{y} = x + x^2 + C$$

Substituting the initial conditions yields C = 2. Substituting this value and rearranging gives

$$y = \left(\frac{x^2 + x + 2}{2} \right)^2$$

Some selected value can be computed as

x	y
0	**1**
0.25	**1.336914**
0.5	**1.890625**
0.75	**2.743164**

1 4

(b) Euler's method:

$$y(0.25) = y(0) + f(0,1)h$$

$$f(0,1) = (1 + 2(0))\sqrt{1} = 1$$

$$y(0.25) = 1 + 1(0.25) = 1.25$$

$$y(0.5) = y(0.25) + f(0.25,1.25)0.25$$

$$f(0.25,1.25) = (1 + 2(0.25))\sqrt{1.25} = 1.67705$$

$$y(0.5) = 1.25 + 1.67705(0.25) = 1.66926$$

The remaining steps can be implemented and summarized as

x	*y*	*dy/dx*
0	**1**	1
0.25	**1.25**	1.67705
0.5	**1.66926**	2.584
0.75	**2.31526**	3.804
1	**3.26626**	5.42184

(c) Heun's method:

Predictor:

$$k_1 = (1 + 2(0))\sqrt{1} = 1$$

$$y(0.25) = 1 + 1(0.25) = 1.25$$

$$k_2 = (1 + 2(0.25))\sqrt{1.25} = 1.6771$$

Corrector:

$$y(0.25) = 1 + \frac{1 + 1.6771}{2}0.25 = 1.33463$$

The remaining steps can be implemented and summarized as

x	*y*	k_1	x_e	y_e	k_2	*dy/dx*
0	**1**	1.0000	0.25	1.25	1.6771	1.3385
0.25	**1.33463**	1.7329	0.5	1.76785	2.6592	2.1961
0.5	**1.88364**	2.7449	0.75	2.56987	4.0077	3.3763
0.75	**2.72772**	4.1290	1	3.75996	5.8172	4.9731
1	**3.97099**					

(d) Ralston's method:

Predictor:

$$k_1 = (1 + 2(0))\sqrt{1} = 1$$

$$y(0.1875) = 1 + 1(0.1875) = 1.1875$$

$$k_2 = (1 + 2(0.1875))\sqrt{1.1875} = 1.49837$$

Corrector:

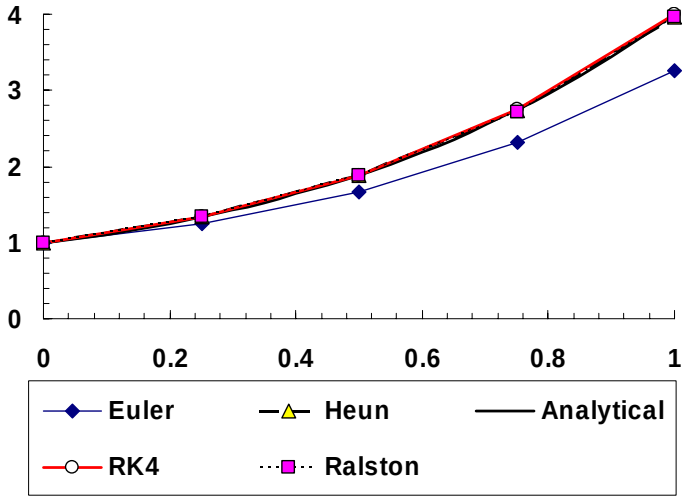
$$y(0.25) = 1 + \frac{1 + 2(1.49837)}{3} 0.25 = 1.33306$$

The remaining steps can be implemented and summarized as

x	*y*	k_1	$x + 3/4h$	$y + (3/4)k_1h$	k_2	dy/dx
0	**1**	1	0.1875	1.1875	1.49837	1.3322
0.25	**1.33306**	1.73187	0.4375	1.65779	2.41416	2.1867
0.5	**1.87974**	2.74208	0.6875	2.39388	3.67464	3.3638
0.75	**2.72069**	4.12363	0.9375	3.49387	5.37392	4.9572
1	**3.95998**					

(e) RK4

x	*y*	k_1	x_m	y_m	k_2	x_m	y_m	k_3	x_e	y_e	k_4	ϕ
0	**1.0000**	1	0.125	1.1250	1.32583	0.125	1.1657	1.34961	0.25	1.3374	1.73469	1.3476
0.25	**1.3369**	1.73436	0.375	1.5537	2.18133	0.375	1.6096	2.2202	0.5	1.8919	2.75096	2.2147
0.5	**1.8906**	2.74997	0.625	2.2343	3.36322	0.625	2.3110	3.42043	0.75	2.7457	4.14253	3.4100
0.75	**2.7431**	4.14056	0.875	3.2606	4.96574	0.875	3.3638	5.04368	1	4.0040	6.00299	5.0271
1	**3.9998**											



18.3 (a) Heun's method:

Predictor:

$$k_1 = -2(1) + (0)^2 = -2$$

$$y(0.5) = 1 + (-2)(0.5) = 0$$

$$k_2 = -2(0) + 0.5^2 = 0.25$$

Corrector:

$$y(0.5) = 1 + \frac{-2 + 0.25}{2} 0.5 = 0.5625$$

The remaining steps can be implemented and summarized as

t	y	k_1	x_{i+1}	y_{i+1}	k_2	dy/dt
0	**1**	-2.0000	0.5	0	0.2500	-0.875
0.5	**0.5625**	-0.8750	1	0.125	0.7500	-0.0625
1	**0.53125**	-0.0625	1.5	0.5	1.2500	0.59375
1.5	**0.82813**	0.5938	2	1.125	1.7500	1.17188
2	**1.41406**	1.1719				

(b) As in Part **(a)**, the corrector can be represented as

$$y_{i+1}^1 = 1 + \frac{-2 + (-2(0) + 0.5^2)}{2} 0.5 = 0.5625$$

The corrector can then be iterated to give

$$y_{i+1}^2 = 1 + \frac{-2 + (-2(0.5625) + 0.5^2)}{2} 0.5 = 0.28125$$

$$y_{i+1}^3 = 1 + \frac{-2 + (-2(0.28125) + 0.5^2)}{2} 0.5 = 0.421875$$

The iterations can be continued until the percent relative error falls below 0.1%. This occurs after 12 iterations with the result that $y(0.5) = 0.37491$ with $\varepsilon_a = 0.073\%$. The remaining values can be computed in a like fashion to give

t	y
0	**1.0000000**
0.5	**0.3749084**
1	**0.3334045**
1.5	**0.6526523**

2	**1.2594796**

(c) Midpoint method

$$k_1 = -2(1) + (0)^2 = -2$$

$$y(0.25) = 1 + (-2)(0.25) = 0.5$$

$$k_2 = -2(0.5) + 0.25^2 = -0.9375$$

$$y(0.5) = 1 + (-0.9375)0.5 = 0.53125$$

The remainder of the computations can be implemented in a similar fashion as listed below:

t	*y*	*dy/dt*	t_m	y_m	dy_m/dt
0	**1**	-2.0000	0.25	0.5	-0.9375
0.5	**0.53125**	-0.8125	0.75	0.328125	-0.0938
1	**0.48438**	0.0313	1.25	0.492188	0.57813
1.5	**0.77344**	0.7031	1.75	0.949219	1.16406
2	**1.35547**				

(d) Ralston's method:

$$k_1 = -2(1) + (0)^2 = -2$$

$$y(0.375) = 1 + (-2)(0.375) = 0.25$$

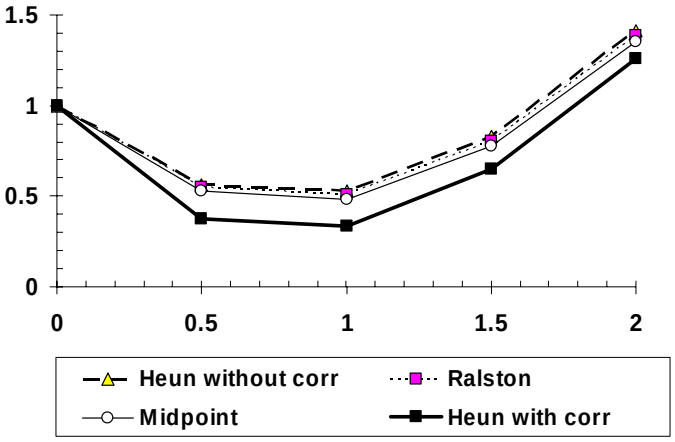
$$k_2 = -2(0.25) + 0.375^2 = -0.3594$$

$$y(0.25) = 1 + \frac{-2 + 2(-0.3594)}{3}0.5 = 0.54688$$

The remaining steps can be implemented and summarized as

t	*y*	k_1	$t + 3/4h$	$y + (3/4)k_1h$	k_2	*dy/dt*
0	**1**	-2.0000	0.375	0.25	-0.3594	-0.9063
0.5	**0.54688**	-0.8438	0.875	0.230469	0.3047	-0.0781
1	**0.50781**	-0.0156	1.375	0.501953	0.8867	0.58594
1.5	**0.80078**	0.6484	1.875	1.043945	1.4277	1.16797
2	**1.38477**					

All the versions can be plotted as:



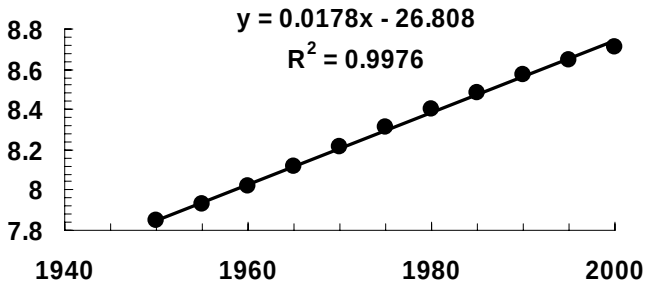
18.4 (a) The solution to the differential equation is

$$p = p_0 e^{k_g t}$$

Taking the natural log of this equation gives

$$\ln p = \ln p_0 + k_g t$$

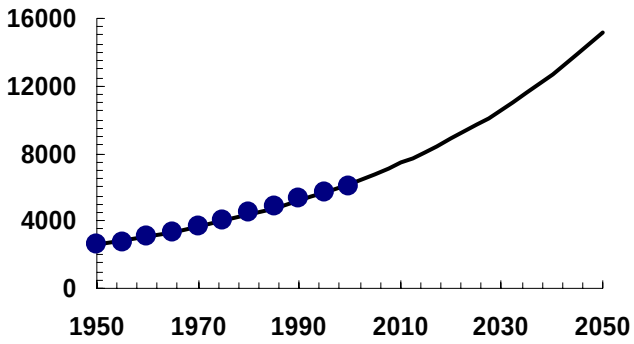
Therefore, a semi-log plot ($\ln p$ versus t) should yield a straight line with a slope of k_g. The plot, along with the linear regression best fit line is shown below. The estimate of the population growth rate is $k_g = 0.0178$/yr.



(b) The ODE can be integrated with the fourth-order RK method with the results tabulated and plotted below:

t	p	k_1	p_{mid}	k_2	p_{mid}	k_3	p_{end}	k_4	ϕ
1950	2555.00	45.41	2668.53	47.43	2673.58	47.52	2792.60	49.64	47.49
1955	2792.46	49.63	2916.55	51.84	2922.06	51.94	3052.15	54.25	51.91
1960	3051.99	54.25	3187.61	56.66	3193.64	56.76	3335.81	59.29	56.73
1965	3335.64	59.29	3483.87	61.92	3490.45	62.04	3645.84	64.80	62.00
1970	3645.66	64.80	3807.66	67.68	3814.85	67.81	3984.69	70.82	67.77
1975	3984.48	70.82	4161.54	73.97	4169.41	74.11	4355.02	77.41	74.06
1980	4354.80	77.40	4548.31	80.84	4556.91	81.00	4759.78	84.60	80.95

1985	4759.54	84.60	4971.03	88.36	4980.43	88.52	5202.15	92.46	88.47
1990	5201.89	92.46	5433.04	96.57	5443.31	96.75	5685.64	101.06	96.69
1995	5685.35	101.05	5937.98	105.54	5949.21	105.74	6214.06	110.45	105.68
2000	6213.75	110.44	6489.86	115.35	6502.13	115.57	6791.60	120.72	115.50
2005	6791.25	120.71	7093.02	126.07	7106.43	126.31	7422.81	131.93	126.24
2010	7422.43	131.93	7752.25	137.79	7766.90	138.05	8112.68	144.20	137.97
2015	8112.27	144.19	8472.74	150.60	8488.76	150.88	8866.67	157.60	150.79
2020	8866.22	157.59	9260.20	164.59	9277.70	164.90	9690.74	172.25	164.80
2025	9690.24	172.24	10120.84	179.89	10139.97	180.23	10591.40	188.25	180.12
2030	10590.85	188.24	11061.47	196.61	11082.38	196.98	11575.76	205.75	196.86
2035	11575.17	205.74	12089.52	214.88	12112.37	215.29	12651.61	224.87	215.16
2040	12650.96	224.86	13213.11	234.85	13238.09	235.30	13827.45	245.77	235.16
2045	13826.74	245.76	14441.14	256.68	14468.44	257.17	15112.57	268.61	257.01
2050	15111.79								



18.5 **(a)** The analytical solution can be used to compute values at times over the range. For example, the value at $t = 1955$ can be computed as

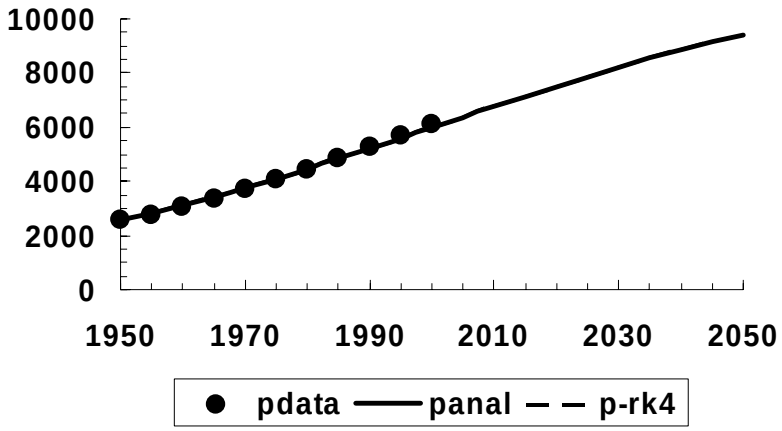
$$p = 2{,}555\frac{12{,}000}{2{,}555 + (12{,}000 - 2{,}555)e^{-0.026(1955-1950)}} = 2{,}826.2$$

Values at the other times can be computed and displayed along with the data in the plot below.

(b) The ODE can be integrated with the fourth-order RK method with the results tabulated and plotted below:

t	p-rk4	k_1	t_m	y_m	k_2	t_m	y_m	k_3	t_e	y_e	k_4	ϕ
1950	2555.0	52.29	1952.5	2685.7	54.20	1952.5	2690.5	54.27	1955.0	2826.3	56.18	54.23
1955	2826.2	56.17	1957.5	2966.6	58.06	1957.5	2971.3	58.13	1960.0	3116.8	59.99	58.09
1960	3116.6	59.99	1962.5	3266.6	61.81	1962.5	3271.1	61.87	1965.0	3425.9	63.64	61.83
1965	3425.8	63.64	1967.5	3584.9	65.36	1967.5	3589.2	65.41	1970.0	3752.8	67.06	65.37
1970	3752.6	67.06	1972.5	3920.3	68.63	1972.5	3924.2	68.66	1975.0	4096.0	70.15	68.63
1975	4095.8	70.14	1977.5	4271.2	71.52	1977.5	4274.6	71.55	1980.0	4453.5	72.82	71.52
1980	4453.4	72.82	1982.5	4635.4	73.97	1982.5	4638.3	73.98	1985.0	4823.3	75.00	73.95
1985	4823.1	75.00	1987.5	5010.6	75.88	1987.5	5012.8	75.89	1990.0	5202.6	76.62	75.86

1990	5202.4	76.62	1992.5	5394.0	77.20	1992.5	5395.5	77.21	1995.0	5588.5	77.63	77.18
1995	5588.3	77.63	1997.5	5782.4	77.90	1997.5	5783.1	77.90	2000.0	5977.8	78.00	77.87
2000	5977.7	78.00	2002.5	6172.7	77.94	2002.5	6172.5	77.94	2005.0	6367.4	77.71	77.91
2005	6367.2	77.71	2007.5	6561.5	77.32	2007.5	6560.5	77.32	2010.0	6753.8	76.77	77.29
2010	6753.7	76.77	2012.5	6945.6	76.06	2012.5	6943.9	76.07	2015.0	7134.0	75.21	76.04
2015	7133.9	75.21	2017.5	7321.9	74.21	2017.5	7319.4	74.23	2020.0	7505.0	73.09	74.20
2020	7504.9	73.09	2022.5	7687.6	71.83	2022.5	7684.5	71.85	2025.0	7864.2	70.47	71.82
2025	7864.0	70.47	2027.5	8040.2	68.98	2027.5	8036.5	69.01	2030.0	8209.1	67.43	68.98
2030	8208.9	67.43	2032.5	8377.5	65.75	2032.5	8373.3	65.80	2035.0	8537.9	64.04	65.76
2035	8537.7	64.05	2037.5	8697.8	62.23	2037.5	8693.3	62.28	2040.0	8849.1	60.41	62.25
2040	8849.0	60.41	2042.5	9000.0	58.50	2042.5	8995.2	58.56	2045.0	9141.8	56.61	58.53
2045	9141.6	56.62	2047.5	9283.1	54.65	2047.5	9278.2	54.72	2050.0	9415.2	52.73	54.68
2050	9415.0											



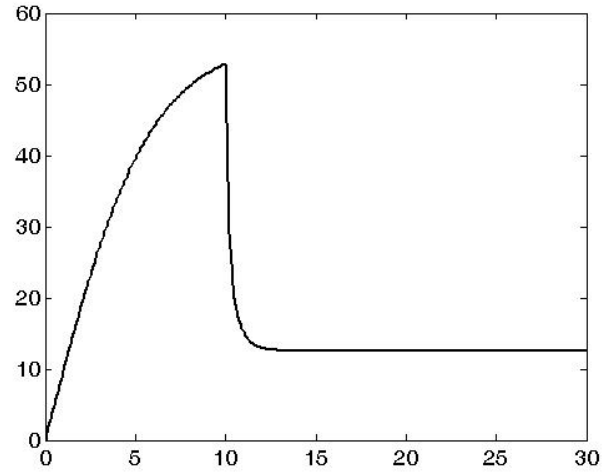
Thus, the RK4 results are so close to the analytical solution that the two results are indistinguishable graphically.

18.6 We can solve this problem with the M-file `Eulode` (Fig. 18.3). First, we develop a function to compute the derivative

```
function dv = dvdt(t, v)
if t < 10
  % chute is unopened
  dv = 9.81 - 0.25/80*v^2;
else
  % chute is opened
  dv = 9.81 - 5/80*v^2;
end
```

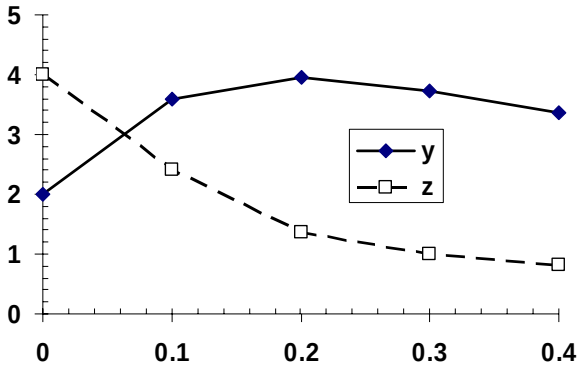
Notice how we have used an If statement to use a higher drag coefficient for times after the cord is pulled. The `Eulode` function can then be used to generate results and display them graphically..

```
>> [t,v] = Eulode(@dvdt,[0 30],0,0.1);
>> plot(t,v)
```



18.7 **(a)** Euler's method:

t	*y*	*z*	*dy*/*dt*	*dz*/*dt*
0	2	4	16	-16
0.1	3.6	2.4	3.658049	-10.368
0.2	3.965805	1.3632	-2.35114	-3.68486
0.3	3.730691	0.994714	-3.77687	-1.84568
0.4	3.353004	0.810147	-3.99072	-1.10035



(b) 4th-order RK method:

$$k_{1,1} = f_1(0,2,4) = -2(2) + 5(4)e^{-0} = 16$$

$$k_{1,2} = f_2(0,2,4) = -\frac{2(4)^2}{2} = -16$$

$$y(0.05) = 2 + 16(0.05) = 2.8$$

$$z(0.05) = 4 - 16(0.05) = 3.2$$

$$k_{2,1} = f_1(0.05,2.8,3.2) = -2(2.8) + 5(3.2)e^{-0.05} = 9.619671$$

$$k_{2,2} = f_2(0.05,2.8,3.2) = -\frac{2.8(3.2)^2}{2} = -14.336$$

$$y(0.05) = 2 + 9.619671(0.05) = 2.480984$$

$$z(0.05) = 4 - 14.336(0.05) = 3.2832$$

$$k_{3,1} = f_1(0.05,2.480984,3.2832) = -2(2.480984) + 5(3.2832)e^{-0.05} = 10.65342$$

$$k_{3,2} = f_2(0.05,2.480984,3.2832) = -\frac{2.480984(3.2832)^2}{2} = -13.3718$$

$$y(0.1) = 2 + 10.65342(0.1) = 3.065342$$

$$z(0.1) = 4 - 13.3718(0.1) = 2.662824$$

$$k_{4,1} = f_1(0.1,3.065342,2.662824) = -2(3.065342) + 5(3.2832)e^{-0.1} = 5.916431$$

$$k_{4,2} = f_2(0.1,3.065342,2.662824) = -\frac{3.065342(2.662824)^2}{2} = -10.8676$$

The *k*'s can then be used to compute the increment functions,

$$\phi_1 = \frac{16 + 2(9.619671 + 10.65342) + 5.916431}{6} = 10.41043$$

$$\phi_2 = \frac{-16 + 2(-14.336 - 13.3718) - 10.8676}{6} = -13.7139$$

These slope estimates can then be used to make the prediction for the first step

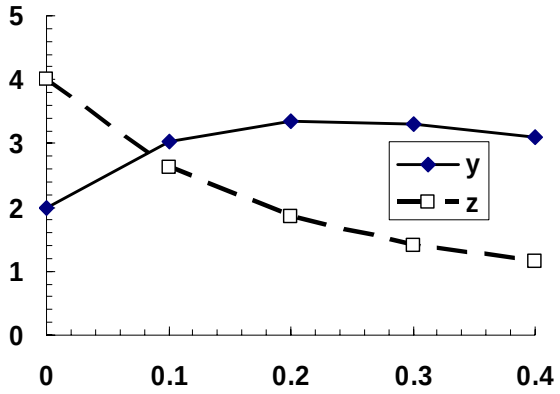
$$y(0.1) = 2 + 10.41043(0.1) = 3.041043$$

$$z(0.1) = 4 - 13.7139(0.1) = 2.628615$$

The remaining steps can be taken in a similar fashion and the results summarized as

t	*y*	*z*
0	2	4
0.1	3.041043	2.628615
0.2	3.342571	1.845308
0.3	3.301983	1.410581
0.4	3.107758	1.149986

A plot of these values can be developed.



18.8 The second-order van der Pol equation can be reexpressed as a system of 2 first-order ODEs,

$$\frac{dy}{dt} = z$$

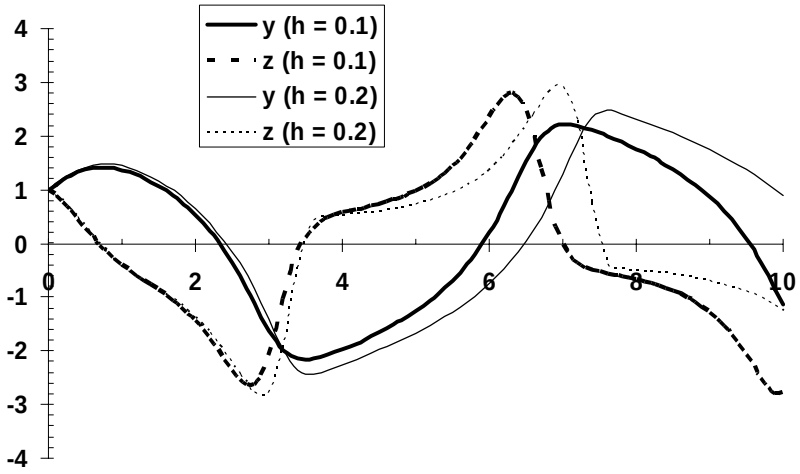
$$\frac{dz}{dt} = (1 - y^2)z - y$$

(a) Euler (h = 0.2). Here are the first few steps. The remainder of the computation would be implemented in a similar fashion and the results displayed in the plot below.

t	y (h = 0.2)	z (h = 0.2)	dy/dt	dz/dt
0	1	1	1	-1
0.2	1.2	0.8	0.8	-1.552
0.4	1.36	0.4896	0.4896	-1.77596
0.6	1.45792	0.1344072	0.134407	-1.6092
0.8	1.4848014	-0.187433	-0.18743	-1.25901

(b) Euler (h = 0.1). Here are the first few steps. The remainder of the computation would be implemented in a similar fashion and the results displayed in the plot below.

t	y (h = 0.1)	z (h = 0.1)	dy/dt	dz/dt
0	1	1	1	-1
0.1	1.1	0.9	0.9	-1.289
0.2	1.19	0.7711	0.7711	-1.51085
0.3	1.26711	0.6200145	0.620015	-1.64257
0.4	1.3291115	0.4557574	0.455757	-1.67847



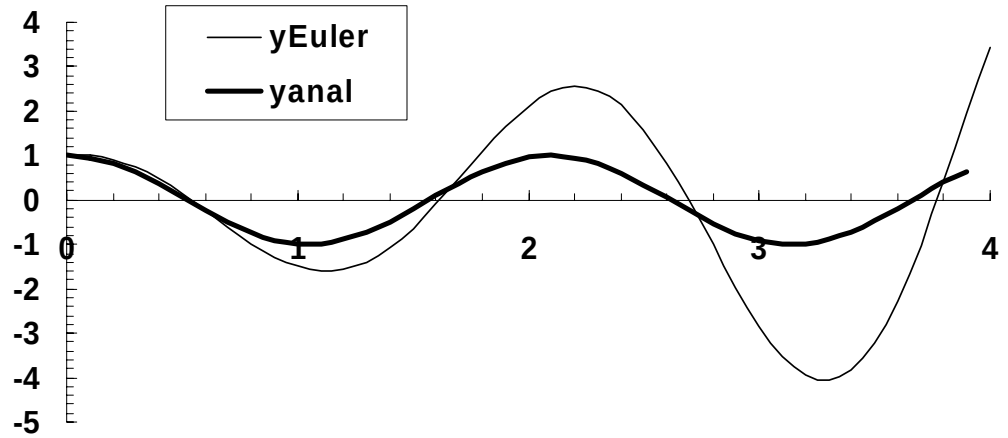
18.9 The second-order equation can be reexpressed as a system of two first-order ODEs,

$$\frac{dy}{dt} = z$$

$$\frac{dz}{dt} = -9y$$

(a) Euler. Here are the first few steps along with the analytical solution. The remainder of the computation would be implemented in a similar fashion and the results displayed in the plot below.

t	y_{Euler}	z_{Euler}	dy/dt	dz/dt	$y_{analytical}$
0	1	0	0	-9	1
0.1	1	-0.9	-0.9	-9	0.955336
0.2	0.91	-1.8	-1.8	-8.19	0.825336
0.3	0.73	-2.619	-2.619	-6.57	0.62161
0.4	0.4681	-3.276	-3.276	-4.2129	0.362358



(b) RK4. Here are the first few steps along with the analytical solution. The remainder of the computation would be implemented in a similar fashion and the results displayed in the plot below.

$$k_{1,1} = f_1(0,1,0) = z = 0$$

$$k_{1,2} = f_2(0,1,0) = -9y = -9(1) = -9$$

$$y(0.05) = 1 + 0(0.05) = 1$$

$$z(0.05) = 0 - 9(0.05) = -0.45$$

$$k_{2,1} = f_1(0.05,1,-0.45) = -0.45$$

$$k_{2,2} = f_2(0.05,1,-0.45) = -9(1) = -9$$

$$y(0.05) = 1 + (-0.45)(0.05) = 0.9775$$

$$z(0.05) = 0 - 9(0.05) = -0.45$$

$$k_{3,1} = f_1(0.05,0.9775,-0.45) = -0.45$$

$$k_{3,2} = f_2(0.05,0.9775,-0.45) = -9(0.9775) = -8.7975$$

$$y(0.1) = 1 + (-0.45)(0.1) = 0.9550$$

$$z(0.1) = 0 - 8.7975(0.1) = -0.8798$$

$$k_{4,1} = f_1(0.1,0.9550,-0.8798) = -0.8798$$

$$k_{4,2} = f_2(0.1,0.9550,-0.8798) = -9(0.9550) = -8.5950$$

The *k*'s can then be used to compute the increment functions,

$$\phi_1 = \frac{0 + 2(-0.45 - 0.45) - 0.8798}{6} = -0.4466$$

$$\phi_2 = \frac{-9 + 2(-9 - 8.7975) - 8.5950}{6} = -8.8650$$

These slope estimates can then be used to make the prediction for the first step

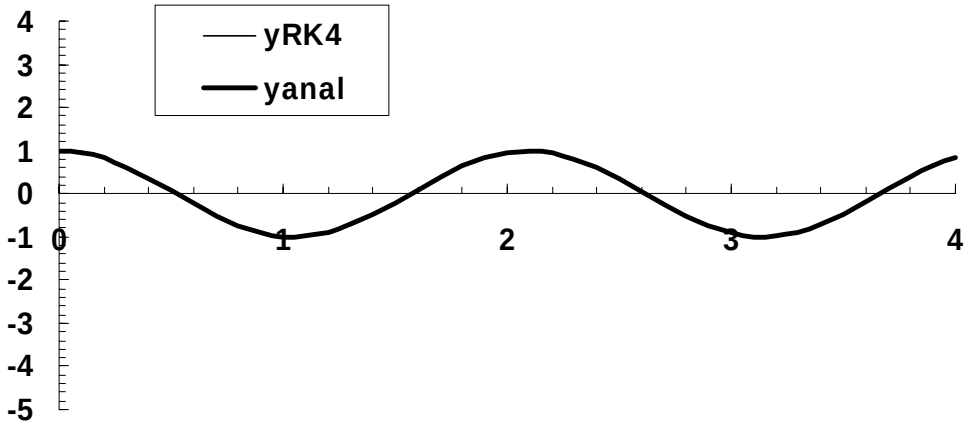
$$y(0.1) = 1 - 0.4466(0.1) = 0.9553$$

$$z(0.1) = 0 - 8.8650(0.1) = -0.8865$$

The remaining steps can be taken in a similar fashion and the first few results summarized as

t	y	z	y_{anal}
0	1.0000	0.0000	1.00000
0.1	0.9553	-0.8865	0.95534
0.2	0.8253	-1.6938	0.82534
0.3	0.6216	-2.3498	0.62161
0.4	0.3624	-2.7960	0.36236
0.5	0.0708	-2.9924	0.07074

As can be seen, the results agree with the analytical solution closely. A plot of all the values can be developed and indicates the same close agreement.



18.10 A MATLAB M-file for Heun's method with iteration can be developed as

```
function [t,y] = Heun(dydt,tspan,y0,h,es,maxit)
% [t,y] = Heun(dydt,tspan,y0,h):
%   uses the midpoint method to integrate an ODE
% input:
%   dydt = name of the M-file that evaluates the ODE
%   tspan = [ti, tf] where ti and tf = initial and
%           final values of independent variable
%   y0 = initial value of dependent variable
%   h = step size
%   es = stopping criterion (%)
%        optional (default = 0.001)
%   maxit = maximum iterations of corrector
%        optional (default = 50)
%   es = (optional) stopping criterion (%)
%   maxit = (optional) maximum allowable iterations
% output:
%   t = vector of independent variable
%   y = vector of solution for dependent variable

% if necessary, assign default values
if nargin<6, maxit = 50; end  %if maxit blank set to 50
if nargin<5, es = 0.001; end  %if es blank set to 0.001
ti = tspan(1);
tf = tspan(2);
t = (ti:h:tf)';
n = length(t);
% if necessary, add an additional value of t
```

```
% so that range goes from t = ti to tf
if t(n)<tf
  t(n+1) = tf;
  n = n+1;
end
y = y0*ones(n,1); %preallocate y to improve efficiency
iter = 0;
for i = 1:n-1
  hh = t(i+1) - t(i);
  k1 = feval(dydt,t(i),y(i));
  y(i+1) = y(i) + k1*hh;
  while (1)
    yold = y(i+1);
    k2 = feval(dydt,t(i)+hh,y(i+1));
    y(i+1) = y(i) + (k1+k2)/2*hh;
    iter = iter + 1;
    if y(i+1) ~= 0, ea = abs((y(i+1) - yold)/y(i+1)) * 100; end
    if ea <= es | iter >= maxit, break, end
  end
end
plot(t,y)
```

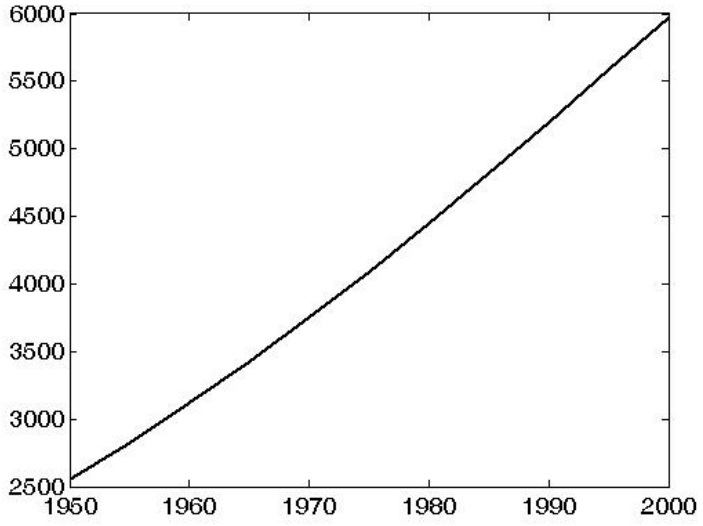
Here is the test of the solution of Prob. 18.5. First, an M-file holding the differential equation is written as

```
function dp = dpdt(t, p)
dp = 0.026*(1-p/12000)*p;
```

Then the M-file can be invoked as in

```
>> [t,p]=Heun(@dpdt,[1950 2000],2555,5,0.1);
>> disp([t,p])
  1.0e+003 *

    1.9500    2.5550
    1.9550    2.8261
    1.9600    3.1165
    1.9650    3.4256
    1.9700    3.7523
    1.9750    4.0953
    1.9800    4.4527
    1.9850    4.8222
    1.9900    5.2012
    1.9950    5.5868
    2.0000    5.9759
```

The following plot is generated



18.11 A MATLAB M-file for the midpoint method can be developed as

```
function [t,y] = midpoint(dydt,tspan,y0,h)
% [t,y] = midpoint(dydt,tspan,y0,h):
%   uses the midpoint method to integrate an ODE
% input:
%   dydt = name of the M-file that evaluates the ODE
%   tspan = [ti, tf] where ti and tf = initial and
%           final values of independent variable
%   y0 = initial value of dependent variable
%   h = step size
% output:
%   t = vector of independent variable
%   y = vector of solution for dependent variable

ti = tspan(1);
tf = tspan(2);
t = (ti:h:tf)';
n = length(t);
% if necessary, add an additional value of t
% so that range goes from t = ti to tf
if t(n)<tf
  t(n+1) = tf;
  n = n+1;
end
y = y0*ones(n,1); %preallocate y to improve efficiency
for i = 1:n-1
  hh = t(i+1) - t(i);
  k1 = feval(dydt,t(i),y(i));
  ymid = y(i) + k1*hh/2;
  k2 = feval(dydt,t(i)+hh/2,ymid);
  y(i+1) = y(i) + k2*hh;
end
plot(t,y)
```

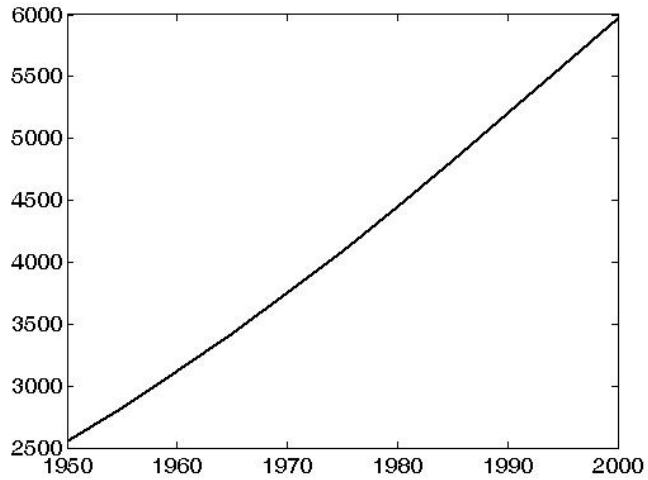
Here is the test of the solution of Prob. 18.5. First, an M-file holding the differential equation is written as

```
function dp = dpdt(t, p)
dp = 0.026*(1-p/12000)*p;
```

Then the M-file can be invoked as in

```
>> [t,p]=midpoint(@dpdt,[1950 2000],2555,5);
>> disp([t,p])
  1.0e+003 *
    1.9500    2.5550
    1.9550    2.8260
    1.9600    3.1163
    1.9650    3.4253
    1.9700    3.7521
    1.9750    4.0953
    1.9800    4.4529
    1.9850    4.8227
    1.9900    5.2021
    1.9950    5.5881
    2.0000    5.9776
```

The following plot is generated



18.12 A MATLAB M-file for the fourth-order RK method can be developed as

```
function [t,y] = rk4(dydt,tspan,y0,h)
% [t,y] = rk4(dydt,tspan,y0,h):
%   uses the fourth-order Runge-Kutta method to integrate an ODE
% input:
%   dydt = name of the M-file that evaluates the ODE
%   tspan = [ti, tf] where ti and tf = initial and
%           final values of independent variable
%   y0 = initial value of dependent variable
%   h = step size
% output:
%   t = vector of independent variable
%   y = vector of solution for dependent variable

ti = tspan(1);
tf = tspan(2);
t = (ti:h:tf)';
n = length(t);
% if necessary, add an additional value of t
```

```
% so that range goes from t = ti to tf
if t(n)<tf
  t(n+1) = tf;
  n = n+1;
end
y = y0*ones(n,1); %preallocate y to improve efficiency
for i = 1:n-1
  hh = t(i+1) - t(i);
  k1 = feval(dydt,t(i),y(i));
  ymid = y(i) + k1*hh/2;
  k2 = feval(dydt,t(i)+hh/2,ymid);
  ymid = y(i) + k2*hh/2;
  k3 = feval(dydt,t(i)+hh/2,ymid);
  yend = y(i) + k3*hh;
  k4 = feval(dydt,t(i)+hh,yend);
  phi = (k1+2*(k2+k3)+k4)/6;
  y(i+1) = y(i) + phi*hh;
end
plot(t,y)
```

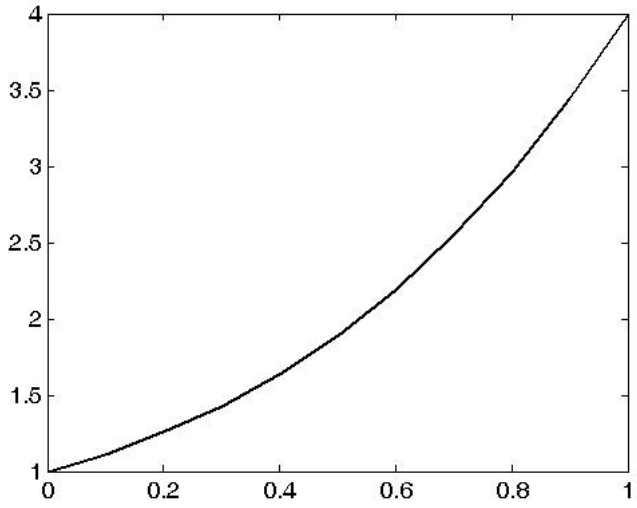
Here is the test of the solution of Prob. 18.2. First, an M-file holding the differential equation is written as

```
function dy = dydx(x, y)
dy = (1+2*x)*sqrt(y);
```

Then the M-file can be invoked as in

```
>> [x,y] = rk4(@dydx,[0 1],1,0.1);
>> disp([x,y])
         0    1.0000
    0.1000    1.1130
    0.2000    1.2544
    0.3000    1.4280
    0.4000    1.6384
    0.5000    1.8906
    0.6000    2.1904
    0.7000    2.5440
    0.8000    2.9584
    0.9000    3.4410
    1.0000    4.0000
```

The following plot is generated



18.13 Note that students can take two approaches to developing this M-file. The first program shown below is strictly developed to solve 2 equations.

```
function [t,y1,y2] = rk42(dy1dt,dy2dt,tspan,y10,y20,h)
% [t,y1,y2] = rk42(dy1dt,dy2dt,tspan,y10,y20,h):
%   uses the fourth-order RK method to integrate a pair of ODEs
% input:
%   dy1dt = name of the M-file that evaluates the first ODE
%   dy2dt = name of the M-file that evaluates the second ODE
%   tspan = [ti, tf] where ti and tf = initial and
%           final values of independent variable
%   y10 = initial value of first dependent variable
%   y20 = initial value of second dependent variable
%   h = step size
% output:
%   t = vector of independent variable
%   y1 = vector of solution for first dependent variable
%   y2 = vector of solution for second dependent variable

ti = tspan(1);
tf = tspan(2);
t = (ti:h:tf)';
n = length(t);
% if necessary, add an additional value of t
% so that range goes from t = ti to tf
if t(n)<tf
  t(n+1) = tf;
  n = n+1;
end
y1 = y10*ones(n,1); %preallocate y's to improve efficiency
y2 = y20*ones(n,1);
for i = 1:n-1
  hh = t(i+1) - t(i);
  k11 = feval(dy1dt,t(i),y1(i),y2(i));
  k12 = feval(dy2dt,t(i),y1(i),y2(i));
  ymid1 = y1(i) + k11*hh/2;
  ymid2 = y2(i) + k12*hh/2;
  k21 = feval(dy1dt,t(i)+hh/2,ymid1,ymid2);
  k22 = feval(dy2dt,t(i)+hh/2,ymid1,ymid2);
  ymid1 = y1(i) + k21*hh/2;
  ymid2 = y2(i) + k22*hh/2;
  k31 = feval(dy1dt,t(i)+hh/2,ymid1,ymid2);
```

```
  k32 = feval(dy2dt,t(i)+hh/2,ymid1,ymid2);
  yend1 = y1(i) + k31*hh;
  yend2 = y2(i) + k32*hh;
  k41 = feval(dy1dt,t(i)+hh,yend1,yend2);
  k42 = feval(dy2dt,t(i)+hh,yend1,yend2);
  phi1 = (k11+2*(k21+k31)+k41)/6;
  phi2 = (k12+2*(k22+k32)+k42)/6;
  y1(i+1) = y1(i) + phi1*hh;
  y2(i+1) = y2(i) + phi2*hh;
end
plot(t,y1,t,y2,'--')
```

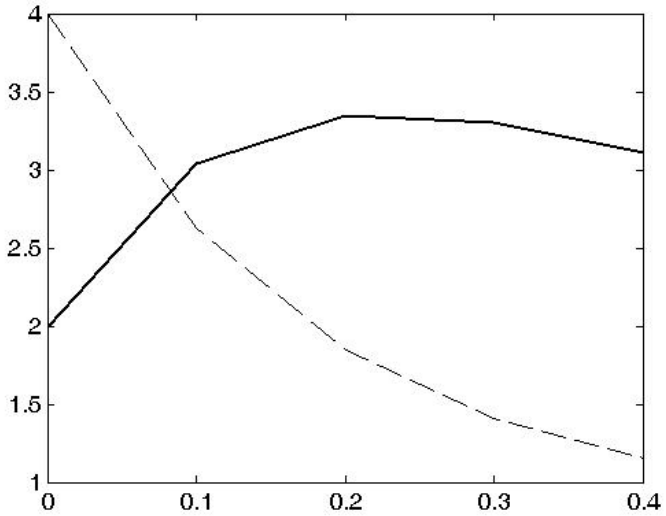
Here is the test of the solution of Prob. 18.7. First, M-files holding the differential equations are written as

```
function dy = dy1dt(t, y1, y2)
dy = -2*y1 + 5*y2*exp(-t);

function dy = dy2dt(t, y1, y2)
dy = -y1*y2^2/2;
```

Then the M-file can be invoked as in

```
>> [t,y1,y2]=rk42(@dy1dt,@dy2dt,[0 0.4],2,4,0.1);
>> disp([t,y1,y2])
         0    2.0000    4.0000
    0.1000    3.0410    2.6286
    0.2000    3.3426    1.8453
    0.3000    3.3020    1.4106
    0.4000    3.1078    1.1500
```

The following plot is generated



A better approach is to develop an M-file that can be used for any number of simultaneous first-order ODEs as in the following code:

```
function [t,y] = rk4sys(dydt,tspan,y0,h)
```

```
% [t,y] = rk4sys(dydt,tspan,y0,h):
%   uses the fourth-order RK method to integrate a pair of ODEs
% input:
%   dydt = name of the M-file that evaluates the ODEs
%   tspan = [ti, tf] where ti and tf = initial and
%           final values of independent variable
%   y0 = initial values of dependent variables
%   h = step size
% output:
%   t = vector of independent variable
%   y = vector of solution for dependent variables

ti = tspan(1);
tf = tspan(2);
t = (ti:h:tf)';
n = length(t);
% if necessary, add an additional value of t
% so that range goes from t = ti to tf
if t(n)<tf
  t(n+1) = tf;
  n = n+1;
end
y(1,:) = y0;
for i = 1:n-1
  hh = t(i+1) - t(i);
  k1 = feval(dydt,t(i),y(i,:))';
  ymid = y(i,:) + k1*hh/2;
  k2 = feval(dydt,t(i)+hh/2,ymid)';
  ymid = y(i,:) + k2*hh/2;
  k3 = feval(dydt,t(i)+hh/2,ymid)';
  yend = y(i,:) + k3*hh;
  k4 = feval(dydt,t(i)+hh,yend)';
  phi = (k1+2*(k2+k3)+k4)/6;
  y(i+1,:) = y(i,:) + phi*hh;
end
plot(t,y(:,1),t,y(:,2),'--')
```

This code solves as many ODEs as are specified. Here is the test of the solution of Prob. 18.7. First, a single M-file holding the differential equations can be written as

```
function dy = dydtsys(t, y)
dy = [-2*y(1) + 5*y(2)*exp(-t);-y(1)*y(2)^2/2];
```

Then the M-file can be invoked as in

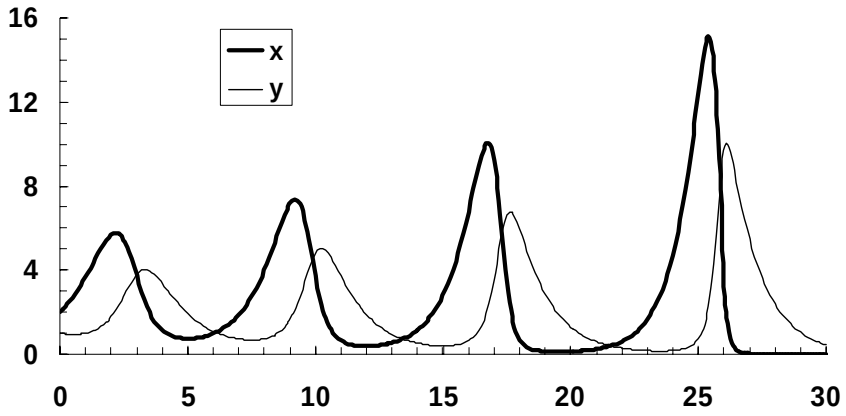
```
>> [t,y]=rk4sys(@dydtsys,[0 0.4],[2 4],0.1);
>> disp([t,y])
         0    2.0000    4.0000
    0.1000    3.0410    2.6286
    0.2000    3.3426    1.8453
    0.3000    3.3020    1.4106
    0.4000    3.1078    1.1500
```

CHAPTER 19

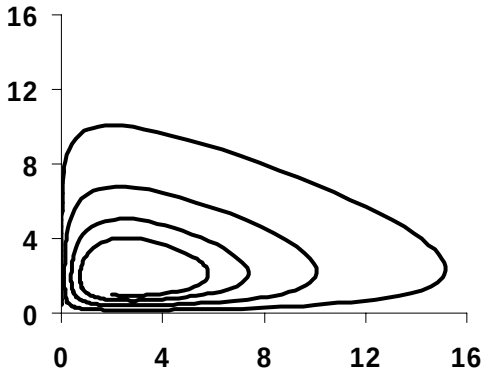
19.1 (a) Euler's method. Here are the first few steps

t	*x*	*y*	*dx/dt*	*dy/dt*
0	2.0000	1.0000	1.2000	-0.2000
0.1	2.1200	0.9800	1.2974	-0.1607
0.2	2.2497	0.9639	1.3985	-0.1206
0.3	2.3896	0.9519	1.5028	-0.0791
0.4	2.5399	0.9440	1.6093	-0.0359
0.5	2.7008	0.9404	1.7171	0.0096

The computation can be continued and the results plotted versus time:



Notice that the amplitudes of the oscillations are expanding. This is also illustrated by a state-space plot (*y* versus *x*):



(b) RK4. Here is the first step in detail.

$$k_{1,1} = f_1(0,2,1) = 1.5(2) - 0.7(2)(1) = 1.6$$

$$k_{1,2} = f_2(0,2,1) = -0.9(1) + 0.4(2)(1) = -0.1$$

$$x(0.05) = 2 + 1.6(0.05) = 2.08$$

$$y(0.05) = 1 - 0.1(0.05) = 0.995$$

$$k_{2,1} = f_1(0.05, 2.08, 0.995) = 1.67128$$

$$k_{2,2} = f_2(0.05, 2.08, 0.995) = -0.06766$$

$$x(0.05) = 2 + 1.67128(0.05) = 2.083564$$

$$y(0.05) = 1 - 9(0.05) = 0.996617$$

$$k_{3,1} = f_1(0.05, 2.083564, 0.996617) = 1.671785$$

$$k_{3,2} = f_2(0.05, 2.083564, 0.996617) = -0.06635$$

$$x(0.1) = 2 + 1.671785(0.1) = 2.167179$$

$$y(0.1) = 1 - 0.06635(0.1) = 0.993365$$

$$k_{4,1} = f_1(0.1, 2.167179, 0.993365) = 1.743808$$

$$k_{4,2} = f_2(0.1, 2.167179, 0.993365) = -0.03291$$

The *k*'s can then be used to compute the increment functions,

$$\phi_1 = \frac{1.6 + 2(1.67128 + 1.671785) + 1.743808}{6} = 1.671656$$

$$\phi_2 = \frac{-0.1 + 2(-0.06766 - 0.06635) - 0.03291}{6} = -0.06682$$

These slope estimates can then be used to make the prediction for the first step

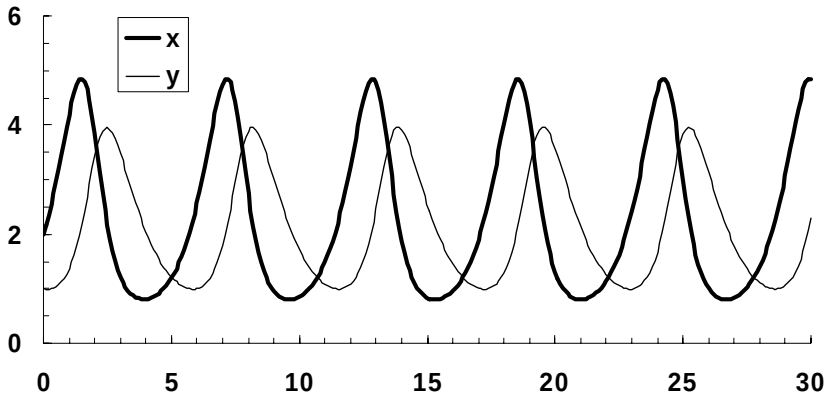
$$x(0.1) = 2 + 1.671656(0.1) = 2.16766$$

$$y(0.1) = 1 - 0.06682(0.1) = 0.993318$$

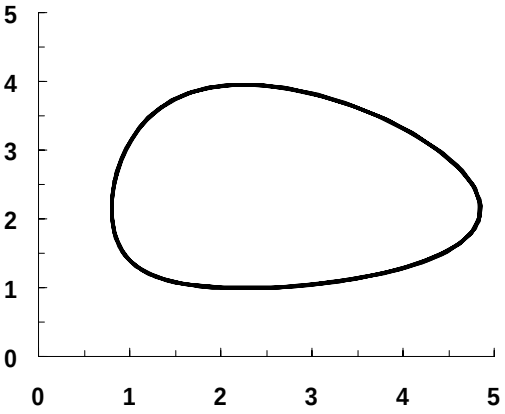
The remaining steps can be taken in a similar fashion and the first few results summarized as

t	*x*	*y*
0	2	1
0.1	2.167166	0.993318
0.2	2.348838	0.993588
0.3	2.545029	1.001398
0.4	2.755314	1.017509
0.5	2.978663	1.042891

A plot of all the values can be developed. Note that in contrast to Euler's method, the cycles do not amplify as time proceeds.



This periodic nature is also evident from the state-space plot. Because this is the expected behavior we can see that the RK4 is far superior to Euler's method for this particular problem.

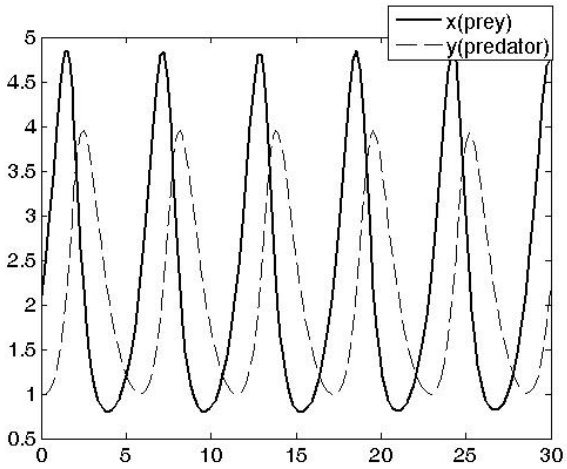


(c) To implement `ode45`, first a function is developed to evaluate the predator-prey ODEs,

```
function yp = predprey(t,y)
yp = [1.5*y(1)-0.7*y(1)*y(2);-0.9*y(2)+0.4*y(1)*y(2)];
```

Then, the solution and plot can be obtained:

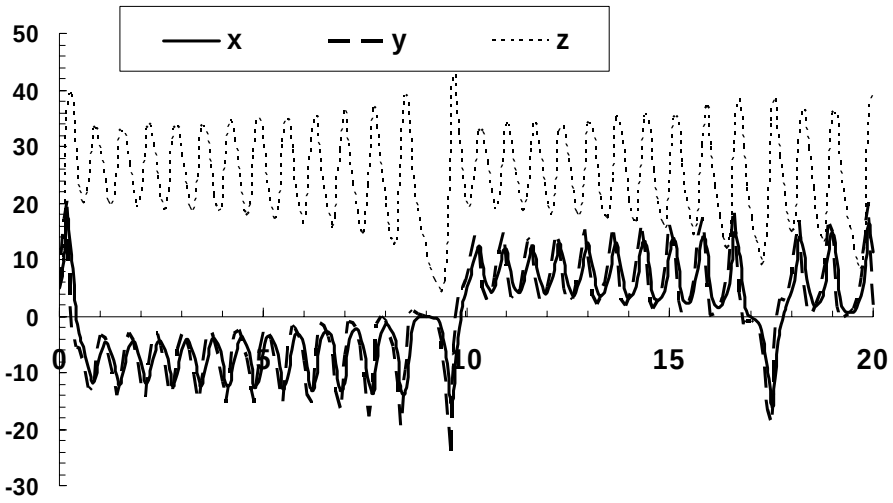
```
>> [t,y] = ode45(@predprey,[0 30],[2 1]);
>> plot(t,y(:,1),t,y(:,2),'--')
>> legend('x(prey)','y(predator)')
```



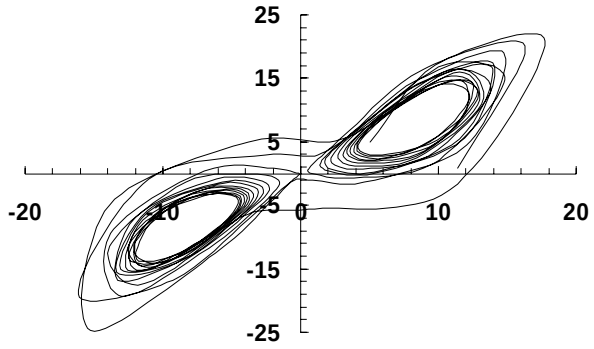
19.2 (a) Here are the results for the first few steps as computed with the classical RK4 technique

t	*x*	*y*	*z*
0	5	5	5
0.1	9.78147	17.07946	10.43947
0.2	17.70297	20.8741	35.89688
0.3	10.81088	-2.52924	39.30744
0.4	0.549578	-5.54419	28.07462
0.5	-3.1646	-5.84128	22.36888
0.6	-5.57588	-8.42037	19.92312
0.7	-8.88719	-12.6789	22.14148
0.8	-11.9142	-13.43	29.80001
0.9	-10.6668	-7.21784	33.39903
1	-6.84678	-3.43018	29.30717

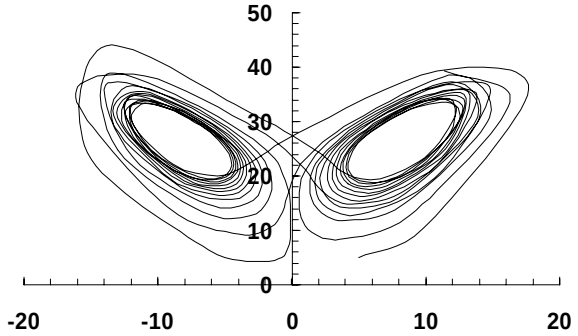
The results from $t = 0$ to 20 can be displayed graphically as



The solution appears chaotic bouncing around from negative to positive values. Although the pattern might appear random, an underlying pattern emerges when we look at the state-space plots. For example, here is the plot of *y* versus *x*.



And here is *z* versus *x*,

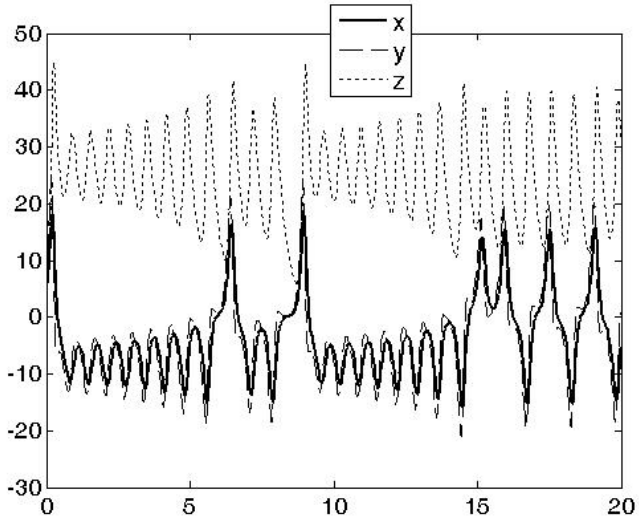


(b) To implement any of the MATLAB functions, first a function is developed to evaluate the Lorenz ODEs,

```
function yp = lorenz(t,y)
yp = [-10*y(1)+10*y(2);28*y(1)-y(2)-y(1)*y(3);-2.666667*y(3)+y(1)*y(2)];
```

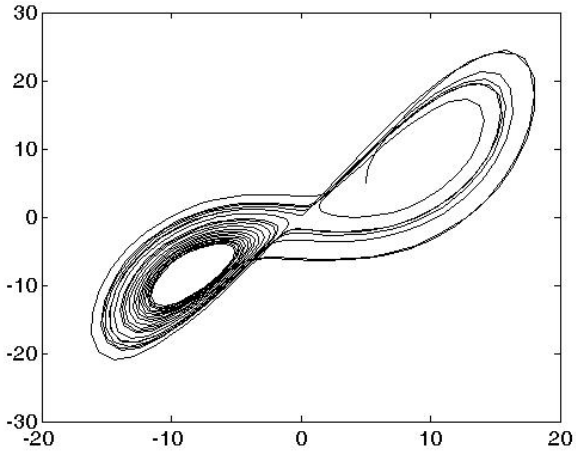
Then, the solution and plots for the `ode23` function can be obtained:

```
>> [t,y] = ode23(@lorenz,[0 20],[5 5 5]);
>> plot(t,y(:,1),t,y(:,2),'--',t,y(:,3),':')
>> legend('x','y','z')
>> plot(y(:,1),y(:,2))
```



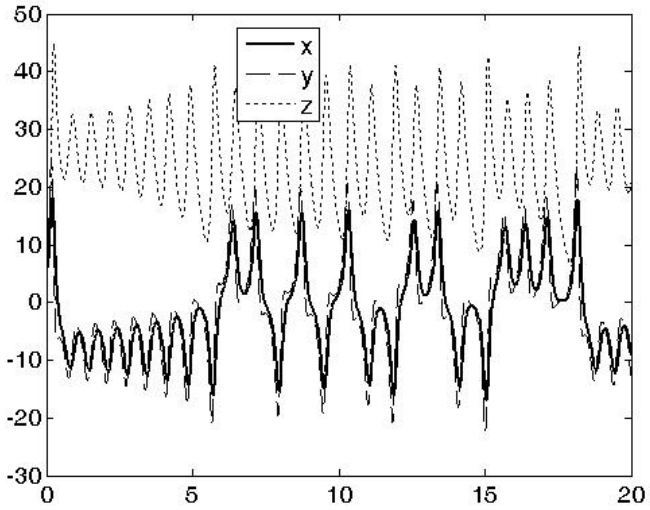
Notice how this plot, although qualitatively similar to the constant step RK4 result in (a), the details are quite different. However, the state-space representation looks much more consistent.

```
>> plot(y(:,1),y(:,2))
```



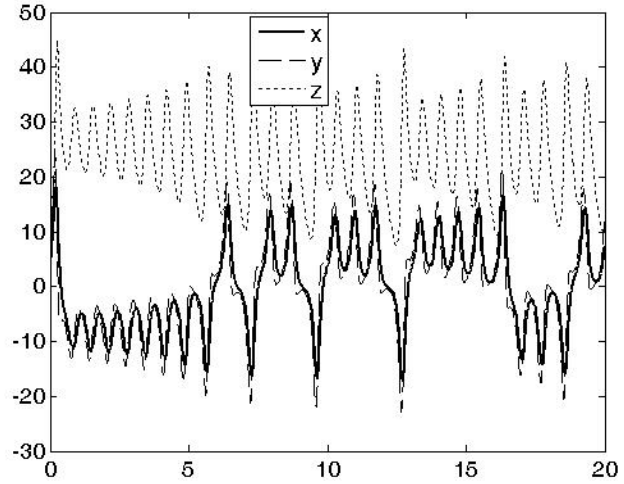
(c) The `ode45` again differs in the details of the time-series plot,

```
>> [t,y] = ode45(@lorenz,[0 20],[5 5 5]);
>> plot(t,y(:,1),t,y(:,2),'--',t,y(:,3),':')
>> legend('x','y','z')
```



(d) The `ode23tb` also differs in the details of the time-series plot,

```
>> [t,y] = ode23tb(@lorenz,[0 20],[5 5 5]);
>> plot(t,y(:,1),t,y(:,2),'--',t,y(:,3),':')
>> legend('x','y','z')
```



Close inspection of all the above results indicates that they all yield identical results for a period of time. Thereafter, they abruptly begin to diverge. The reason for this behavior is that these equations are highly sensitive to their initial conditions. After a number of steps, because they all employ different algorithms, they begin to diverge slightly. When the discrepancy becomes large enough (which for these equations is not that much), the solution will tend to make a large jump. Thus, after awhile, the various solutions become uncorrelated. Such solutions are said to be chaotic. It was this characteristic of these particular equations that led Lorenz to suggest that long-range weather forecasts might not be possible.

19.3 First step:

Predictor:

$y_1^0 = 5.222138+[-0.5(4.143883)+e^{-2}]1 = 3.285532$

Corrector:

$$y_1^1 = 4.143883 + \frac{-0.5(4.143883) + e^{-2} - 0.5(3.285532) + e^{-2.5}}{2} 0.5 = 3.269562$$

The corrector can be iterated to yield

j	y_{i+1}^j	$\lvert\varepsilon_a\rvert$,%
1	3.269562	
2	3.271558	0.061

Second step:

Predictor:

$y_2^0 = 4.143883+[-0.5(3.271558)+e^{-2.5}]1 = 2.590189$

Predictor Modifier:

$y_2^0 = 2.590189+4/5(3.271558-3.285532) = 2.579010$

Corrector:

$$y_2^1 = 3.271558 + \frac{-0.5(3.271558) + e^{-2.5} - 0.5(2.579010) + e^{-3}}{2} 0.5 = 2.573205$$

The corrector can be iterated to yield

j	y_{i+1}^j	$\lvert\varepsilon_a\rvert$,%
1	2.573205	
2	2.573931	0.0282

19.4 Before solving, for comparative purposes, we can develop the analytical solution as

$$y = e^{\frac{t^3}{3} - t}$$

Thus, the true values being simulated in this problem are

t	y
0	1
0.25	0.782868
0.5	0.632337

The first step is taken with the fourth-order RK:

$$k_1 = f(0,1) = 1(0)^2 - 1 = -1$$

$$y(0.125) = 1 - 1(0.125) = 0.875$$

$$k_2 = f(0.125,0.875) = -0.861328$$

$$y(0.125) = 1 - 0.861328(0.125) = 0.89233$$

$$k_3 = f(0.125,0.89233) = -0.87839$$

$$y(0.25) = 1 - 0.87839(0.25) = 0.78040$$

$$k_4 = f(0.25,0.78040) = -0.73163$$

$$\phi = \frac{-1 + 2(-0.861328 - 0.87839) - 0.73163}{6} = -0.86851$$

$$y(0.25) = 1 - 0.86851(0.25) = 0.7828723$$

This result compares favorably with the analytical solution.

The second step can then be implemented with the non-self-starting Heun method:

Predictor:

$$y(0.5) = 1 + (0.7828723(0.25)^2 - 0.7828723)0.5 = 0.633028629$$

Corrector: (First iteration):

$$y(0.5) = 0.7828723 + \frac{-0.7339 + (0.633028629(0.5)^2 - 0.633028629)}{2}0.25 = 0.63178298$$

Corrector: (Second iteration):

$$y(0.5) = 0.7828723 + \frac{-0.7339 + (0.63178298(0.5)^2 - 0.63178298)}{2}0.25 = 0.63189976$$

The iterative process can be continued with the final result converging on 0.63188975.

19.5 (a) $h < 2/100{,}000 = 2\times10^{-5}$.

(*b*) The implicit Euler can be written for this problem as

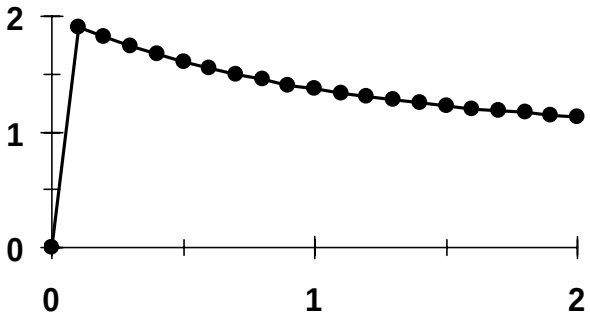
$$y_{i+1} = y_i + \left(-100{,}000\,y_{i+1} + 99{,}999e^{-t_{i+1}}\right)h$$

which can be solved for

$$y_{i+1} = \frac{y_i + 99{,}999e^{-t_{i+1}}h}{1 + 100{,}000h}$$

The results of applying this formula for the first few steps are shown below. A plot of the entire solution is also displayed

t	y
0	0
0.1	1.904638
0.2	1.818731
0.3	1.740819
0.4	1.67032
0.5	1.606531



19.6 The implicit Euler can be written for this problem as

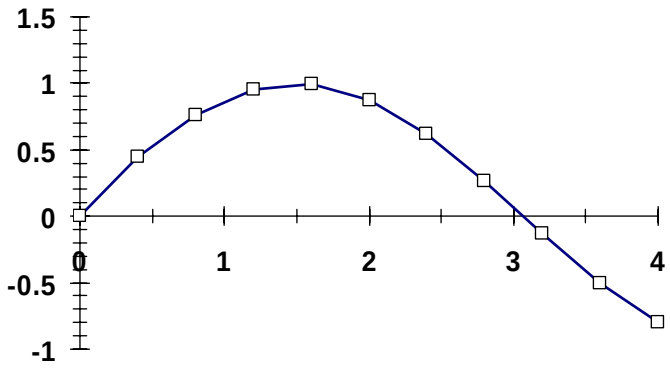
$$y_{i+1} = y_i + \left(30(\sin t_{i+1} - y_{i+1}) + 3\cos t_{i+1}\right)h$$

which can be solved for

$$y_{i+1} = \frac{y_i + 30\sin t_{i+1}h + 3\cos t_{i+1}h}{1 + 30h}$$

The results of applying this formula are tabulated and graphed below.

t	y	t	y	t	y	t	y
0	0	1.2	0.952306	2.4	0.622925	3.6	-0.50089
0.4	0.444484	1.6	0.993242	2.8	0.270163	4	-0.79745
0.8	0.760677	2	0.877341	3.2	-0.12525		



19.7 (a) The explicit Euler can be written for this problem as

$$x_{1,i+1} = x_{1,i} + \left(999x_{1,i} + 1999x_{2,i}\right)h$$
$$x_{2,i+1} = x_{2,i} + \left(-1000x_{1,i} - 2000x_{2,i}\right)h$$

Because the step-size is much too large for the stability requirements, the solution is unstable,

t	x_1	x_2	dx_1/dt	$dx2/dt$
0	1	1	2998	-3000
0.05	150.9	-149	-147102	147100
0.1	-7204.2	7206	7207803	-7207805
0.15	353186	-353184	-3.5E+08	3.53E+08
0.2	-1.7E+07	17305943	1.73E+10	-1.7E+10

(b) The implicit Euler can be written for this problem as

$$x_{1,i+1} = x_{1,i} + \left(999x_{1,i+1} + 1999x_{2,i+1}\right)h$$
$$x_{2,i+1} = x_{2,i} + \left(-1000x_{1,i+1} - 2000x_{2,i+1}\right)h$$

or collecting terms

$$(1 - 999h)x_{1,i+1} - 1999hx_{2,i+1} = x_{1,i}$$
$$1000hx_{1,i+1} + (1 + 2000h)x_{2,i+1} = x_{2,i}$$

or substituting $h = 0.05$ and expressing in matrix format

$$\begin{bmatrix} -48.95 & -99.95 \\ 50 & 101 \end{bmatrix} \begin{Bmatrix} x_{1,i+1} \\ x_{2,i+1} \end{Bmatrix} = \begin{Bmatrix} x_{1,i} \\ x_{2,i} \end{Bmatrix}$$

Thus, to solve for the first time step, we substitute the initial conditions for the right-hand side and solve the 2x2 system of equations. The best way to do this is with LU decomposition since we will have to solve the system repeatedly. For the present case, because its easier to display, we will use the matrix inverse to obtain the solution. Thus, if the matrix is inverted, the solution for the first step amounts to the matrix multiplication,

$$\begin{Bmatrix} x_{1,i+1} \\ x_{2,i+1} \end{Bmatrix} = \begin{bmatrix} 1.886088 & 1.86648 \\ -0.93371 & -0.9141 \end{bmatrix} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix} = \begin{Bmatrix} 3.752568 \\ -1.84781 \end{Bmatrix}$$

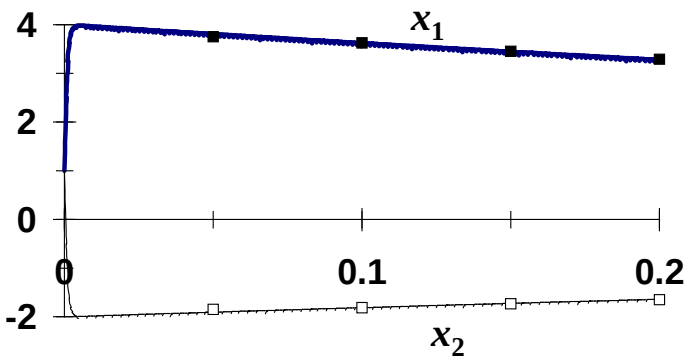
For the second step (from $x = 0.05$ to 0.1),

$$\begin{Bmatrix} x_{1,i+1} \\ x_{2,i+1} \end{Bmatrix} = \begin{bmatrix} 1.886088 & 1.86648 \\ -0.93371 & -0.9141 \end{bmatrix} \begin{Bmatrix} 3.752568 \\ -1.84781 \end{Bmatrix} = \begin{Bmatrix} 3.62878 \\ -1.81472 \end{Bmatrix}$$

The remaining steps can be implemented in a similar fashion to give

t	x_1	x_2
0	1	1
0.05	3.752568	-1.84781
0.1	3.62878	-1.81472
0.15	3.457057	-1.72938
0.2	3.292457	-1.64705

The results are plotted below, along with a solution with the explicit Euler using a step of 0.0005.



19.8 **(a)** The exact solution is

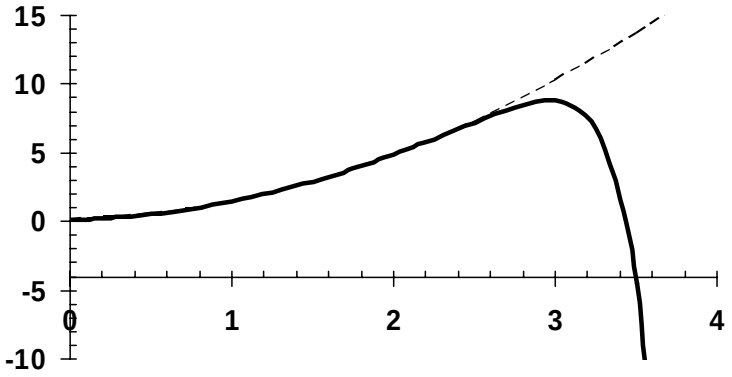
$$y = Ae^{5t} + t^2 + 0.4t + 0.08$$

If the initial condition at $t = 0$ is 0.8, $A = 0$,

$$y = t^2 + 0.4t + 0.08$$

Note that even though the choice of the initial condition removes the positive exponential terms, it still lurks in the background. Very tiny round off errors in the numerical solutions bring it to the fore. Hence all of the following solutions eventually diverge from the analytical solution.

(b) 4th order RK. The plot shows the numerical solution (bold line) along with the exact solution (fine line).



(c)

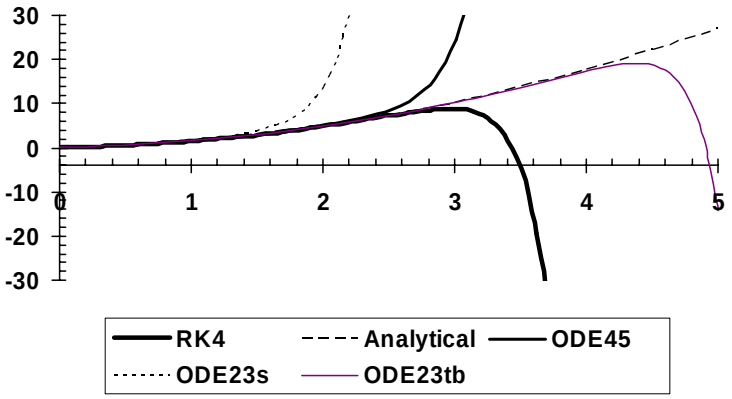
```
function yp = dy(t,y)
yp = 5*(y-t^2);

>> tspan = [0,5];
>> y0 = 0.08;
>> [t,y] = ode45(@dy1,tspan,y0);
```

(*d*)

```
>> [t,y] = ode23s(@dy1,tspan,y0);
```

(*e*)

```
>> [t,y] = ode23tb(@dy1,tspan,y0);
```



19.9 **(a)** As in Example 17.5, the humps function can be integrated with the `quad` function as in

```
>> format long
>> quad(@humps,0,1)

ans =
  29.85832612842764
```

(b) Using `ode45` is based on recognizing that the evaluation of the definite integral

$$I = \int_a^b f(x)\,dx$$

is equivalent to solving the differential equation

$$\frac{dy}{dx} = f(x)$$

for $y(b)$ given the initial condition $y(a) = 0$. Thus, we must solve the following initial-value problem:

$$\frac{dy}{dx} = \frac{1}{(x-0.3)^2 + 0.01} + \frac{1}{(x-0.9)^2 + 0.04} - 6$$

where y(0) = 0. To do this with `ode45`, we must first set up an M-file to evaluate the right-hand side of the differential equation,

```
function dy = humpsODE(x,y)
dy = 1./((x-0.3).^2 + 0.01) + 1./((x-0.9).^2+0.04) - 6;
```

Then, the integral can be evaluated as

```
>> [x,y] = ode45(@humpsODE,[0 0.5 1],0);
>> disp([x,y])
                  0                   0
   0.50000000000000  21.78356481821654
   1.00000000000000  29.85525185285369
```

Thus, the integral estimate is within 0.01% of the estimate obtained with the `quad` function. Note that a better estimate can be obtained by using the `odeset` function to set a smaller relative tolerance as in

```
>> options = odeset('RelTol',1e-8);
>> [x,y] = ode45(@humpsODE,[0 0.5 1],0,options);
>> disp([x,y])
                  0                   0
   0.50000000000000  21.78683736423308
   1.00000000000000  29.85832514287622
```

19.10 The nonlinear model can be expressed as the following set of ODEs,

$$\frac{d\theta}{dt} = v$$
$$\frac{dv}{dt} = -\frac{g}{l}\sin\theta$$

where v = the angular velocity. A function can be developed to compute the right-hand-side of this pair of ODEs for the case where g = 9.81 and l = 0.6 m,

```
function dy = dpnon(t, y)
dy = [y(2);-9.81/0.6*sin(y(1))];
```

The linear model can be expressed as the following set of ODEs,

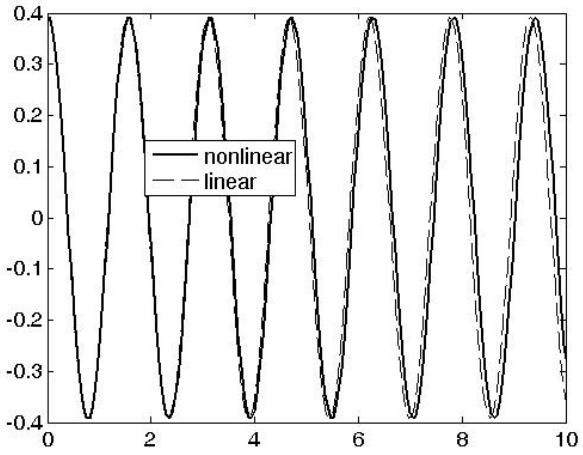
$$\frac{d\theta}{dt} = v$$
$$\frac{dv}{dt} = -\frac{g}{l}\theta$$

A function can be developed as,

```
function dy = dplin(t, y)
dy = [y(2);-9.81/0.6*y(1)];
```

Then, the solution and plot can be obtained for the case where $\theta(0) = \pi/8$. Note that we only depict the displacement (θ or `y(1)`) in the plot

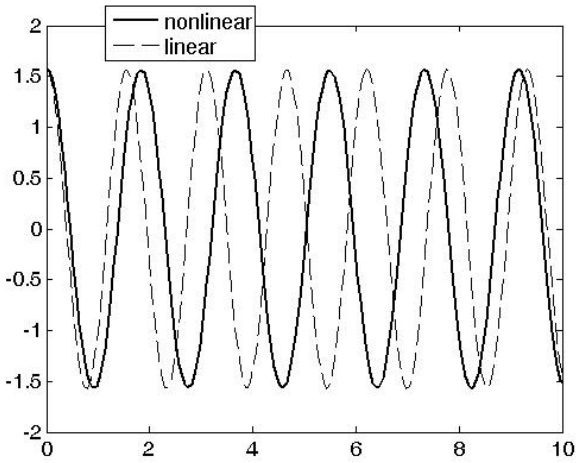
```
>> [tn yn] = ode45(@dpnon,[0 10],[pi/8 0]);
>> [tl yl] = ode45(@dplin,[0 10],[pi/8 0]);
>> plot(tn,yn(:,1),tl,yl(:,1),'--')
>> legend('nonlinear','linear')
```



You should notice two aspects of this plot. First, because the displacement is small, the linear solution provides a decent approximation of the more physically realistic nonlinear case. Second, the two solutions diverge as the computation progresses.

For the larger initial displacement ($\theta(0) = \pi/8$), the solution and plot can be obtained as,

```
>> [tn yn] = ode45(@dpnon,[0 10],[pi/2 0]);
>> [tl yl] = ode45(@dplin,[0 10],[pi/2 0]);
>> plot(tn,yn(:,1),tl,yl(:,1),'--')
>> legend('nonlinear','linear')
```



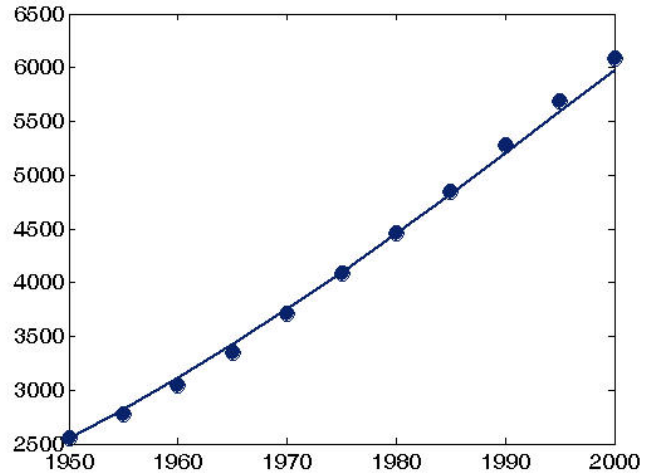
Because the linear approximation is only valid at small displacements, there are now clear and significant discrepancies between the nonlinear and linear cases that are exacerbated as the solution progresses.

19.11 A function can be developed to compute the right-hand-side of the ODE,

```
function yp = dpdt(t, p)
yp = 0.026*(1-p/12000)*p;
```

The function `ode45` can be used to integrate this equation and generate results corresponding to the dates for the measured population data. A plot can also be generated of the solution and the data,

```
>> tspan = 1950:5:2000;
>> pdata = [2555 2780 3040 3346 3708 4087 4454 4850 5276 5686 6079]';
>> [t,p] = ode45(@dpdt,tspan,2555);
>> plot(t,p,t,pdata,'o')
```



The sum of the squares of the residuals can be computed as

```
>> SSR = sum((p - pdata).^2)

SSR =
  4.2365e+004
```

CHAPTER 20

20.1 The matrix inverse can be evaluated and the power method expressed as

$$\begin{bmatrix} 0.0375 & 0.025 & 0.0125 \\ 0.025 & 0.05 & 0.025 \\ 0.0125 & 0.025 & 0.0375 \end{bmatrix} - \lambda[I] = 0$$

Iteration 1:

$$\begin{bmatrix} 0.0375 & 0.025 & 0.0125 \\ 0.025 & 0.05 & 0.025 \\ 0.0125 & 0.025 & 0.0375 \end{bmatrix} \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix} = \begin{Bmatrix} 0.075 \\ 0.1 \\ 0.075 \end{Bmatrix} = 0.1 \begin{Bmatrix} 0.75 \\ 1 \\ 0.75 \end{Bmatrix}$$

Iteration 2:

$$\begin{bmatrix} 0.0375 & 0.025 & 0.0125 \\ 0.025 & 0.05 & 0.025 \\ 0.0125 & 0.025 & 0.0375 \end{bmatrix} \begin{Bmatrix} 0.75 \\ 1 \\ 0.75 \end{Bmatrix} = \begin{Bmatrix} 0.0625 \\ 0.0875 \\ 0.0625 \end{Bmatrix} = 0.0875 \begin{Bmatrix} 0.71428571 \\ 1 \\ 0.71428571 \end{Bmatrix}$$

$$\varepsilon_a = \left| \frac{0.0875 - 0.1}{0.0875} \right| \times 100\% = 14.29\%$$

Iteration 3:

$$\begin{bmatrix} 0.0375 & 0.025 & 0.0125 \\ 0.025 & 0.05 & 0.025 \\ 0.0125 & 0.025 & 0.0375 \end{bmatrix} \begin{Bmatrix} 0.71428571 \\ 1 \\ 0.71428571 \end{Bmatrix} = \begin{Bmatrix} 0.060714 \\ 0.085714 \\ 0.060714 \end{Bmatrix} = 0.085714 \begin{Bmatrix} 0.708333 \\ 1 \\ 0.708333 \end{Bmatrix}$$

$$\varepsilon_a = \left| \frac{0.085714 - 0.0875}{0.085714} \right| \times 100\% = 2.08\%$$

The iterations can be continued. After 10 iterations, the relative error falls to 0.00000884% with the result

$$0.085355 \begin{Bmatrix} 0.70710678 \\ 1 \\ 0.70710678 \end{Bmatrix}$$

Thus, the smallest eigenvalue is 1/0.085355 = 11.71573.

20.2 (a) Minors:

$$(2-\lambda)\begin{vmatrix} 3-\lambda & 4 \\ 4 & 7-\lambda \end{vmatrix} - 2\begin{vmatrix} 8 & 4 \\ 10 & 7-\lambda \end{vmatrix} + 10\begin{vmatrix} 8 & 3-\lambda \\ 10 & 4 \end{vmatrix} = -\lambda^3 + 10\lambda^2 + 101\lambda + 18$$

(b) The eigenvalues can be determined by finding the roots of the characteristic polynomial determined in **(a)**. This can be done in MATLAB,

```
>> a = [1.0000 -10.0000 -101.0000 -18.0000];
>> roots(a)

ans =
   16.2741
   -6.0926
   -0.1815
```

(c) The power method for the highest eigenvalue can be implemented with MATLAB commands,

```
>> A = [2 2 10;8 3 4;10 4 5];
>> x = [1 1 1]';
```

First iteration:

```
>> x = A*x
x =
    14
    15
    19

>> e = max(x)
e =
    19

>> x = x/e
x =
    0.7368
    0.7895
    1.0000
```

Second iteration:

```
>> x = A*x
x =
   13.0526
   12.2632
   15.5263

>> e = max(x)
e =
   15.5263

>> x = x/e
x =
    0.8407
    0.7898
    1.0000
```

Third iteration:

```
>> x = A*x
x =
   13.2610
   13.0949
   16.5661

>> e = max(x)
e =
```

```
   16.5661

>> x = x/e
x =
    0.8005
    0.7905
    1.0000
```

Fourth iteration:

```
>> x = A*x
x =
   13.1819
   12.7753
   16.1668

>> e = max(x)
e =
   16.1668

>> x = x/e
x =
    0.8154
    0.7902
    1.0000
```

Thus, after four iterations, the result is converging on a highest eigenvalue of 16.2741 with a corresponding eigenvector of [0.811 0.790 1].

(d) The power method for the lowest eigenvalue can be implemented with MATLAB commands,

```
>> A = [2 2 10;8 3 4;10 4 5];
>> x = [1 1 1]';
>> AI = inv(A)

AI =
   -0.0556    1.6667   -1.2222
   -0.0000   -5.0000    4.0000
    0.1111    0.6667   -0.5556
```

First iteration:

```
>> x = AI*x

x =
    0.3889
   -1.0000
    0.2222

>> [e,i] = max(abs(x))
e =
     1
i =
     2

>> x = x/x(i)
x =
   -0.3889
    1.0000
   -0.2222
```

Second iteration:

```
>> x = AI*x
x =
    1.9599
   -5.8889
    0.7469

>> [e,i] = max(abs(x))
e =
    5.8889
i =
     2

>> x = x/x(i)
x =
   -0.3328
    1.0000
   -0.1268
```

Third iteration:

```
>> x = AI*x
x =
    1.8402
   -5.5073
    0.7002

>> [e,i] = max(abs(x))
e =
    5.5073
i =
     2

>> x = x/x(i)
x =
   -0.3341
    1.0000
   -0.1271
```

Thus, after three iterations, the estimate of the lowest eigenvalue is converging on the correct value of 1/(–5.5085) = –0.1815 with an eigenvector of [–0.3341 1 -0.1271].

20.3 MATLAB can be used to solve for the eigenvalues with the polynomial method. First, the matrix can be put into the proper form for an eigenvalue analysis by bringing all terms to the left-hand-side of the equation.

$$\begin{bmatrix} 4-9\lambda & 7 & 3 \\ 7 & 8-4\lambda & 2 \\ 3 & 2 & 1-2\lambda \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = 0$$

Then, each row can be divided by the coefficient of λ in that row.

$$\begin{bmatrix} 0.4444-\lambda & 0.7778 & 0.3333 \\ 1.75 & 2-\lambda & 0.5 \\ 1.5 & 1 & 0.5-\lambda \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = 0$$

MATLAB can then be used to determine the eigenvalues as the roots of the characteristic polynomial,

```
>> A=[4/9 7/9 3/9;7/4 8/4 2/4;3/2 2/2 1/2];
>> p=poly(A)
p =
    1.0000   -2.9444   -0.2500    0.2917

>> e=roots(p)
e =
    2.9954
   -0.3386
    0.2876
```

20.4 **(a)** MATLAB can be used to solve for the eigenvalues with the polynomial method. First, the matrix can be put into the proper form by dividing each row by 0.36.

```
>> A = [2/.36 -1/.36 0 0;-1/.36 2/.36 -1/.36 0;0 -1/.36 2/.36 -
1/.36;0 0 -1/.36 2/.36]

A =
    5.5556   -2.7778         0         0
   -2.7778    5.5556   -2.7778         0
         0   -2.7778    5.5556   -2.7778
         0         0   -2.7778    5.5556
```

Then, the poly function can be used to generate the characteristic polynomial,

```
>> p = poly(A)
p =
    1.0000  -22.2222  162.0370 -428.6694  297.6871
```

The roots of this equation represent the eigenvalues,

```
>> e = roots(p)
e =
   10.0501
    7.2723
    3.8388
    1.0610
```

(b) The power method can be used to determine the highest eigenvalue:

```
>> A = [2/.36 -1/.36 0 0;
-1/.36 2/.36 -1/.36 0;
0 -1/.36 2/.36 -1/.36;
0 0 -1/.36 2/.36];
>> x = [1 1 1 1]';
```

First iteration:

```
>> x = A*x
x =
```

```
    2.7778
         0
         0
    2.7778
>> e = max(x)
e =
    2.7778
>> x = x/e
x =
     1
     0
     0
     1
```

Second iteration:

```
>> x = A*x
x =
    5.5556
   -2.7778
   -2.7778
    5.5556
>> e = max(x)
e =
    5.5556
>> x = x/e
x =
    1.0000
   -0.5000
   -0.5000
    1.0000
```

Third iteration:

```
>> x = A*x
x =
    6.9444
   -4.1667
   -4.1667
    6.9444
>> e = max(x)
e =
    6.9444
>> x = x/e
x =
    1.0000
   -0.6000
   -0.6000
    1.0000
```

The process can be continued. After 9 iterations, the method does not converge on the highest eigenvalue. Rather, it converges on the second highest eigenvalue of 7.2723 with a corresponding eigenvector of [1 –0.6180 –0.6180 1].

(c) The power method can be used to determine the lowest eigenvalue by first determining the matrix inverse:

```
>> A = [2/.36 -1/.36 0 0;-1/.36 2/.36 -1/.36 0;0 -1/.36 2/.36 -
1/.36;0 0 -1/.36 2/.36];
>> AI = inv(A)
AI =
    0.2880    0.2160    0.1440    0.0720
    0.2160    0.4320    0.2880    0.1440
    0.1440    0.2880    0.4320    0.2160
    0.0720    0.1440    0.2160    0.2880
>> x = [1 1 1 1]';
```

First iteration:

```
>> x = AI*x
x =
    0.7200
    1.0800
    1.0800
    0.7200
>> e = max(x)
e =
    1.0800
>> x = x/e
x =
    0.6667
    1.0000
    1.0000
    0.6667
```

Second iteration:

```
>> x = AI*x
x =
    0.6000
    0.9600
    0.9600
    0.6000
>> e = max(x)
e =
    0.9600
>> x = x/e
x =
    0.6250
    1.0000
    1.0000
    0.6250
```

Third iteration:

```
>> x = AI*x
x =
    0.5850
    0.9450
```

```
    0.9450
    0.5850
>> e = max(x)
e =
    0.9450
>> x = x/e
x =
    0.6190
    1.0000
    1.0000
    0.6190
```

The process can be continued. After 9 iterations, the method converges on the lowest eigenvalue of 1/0.9450 = 1.0610 with a corresponding eigenvector of [0.6180 1 1 0.6180].

20.5 The parameters can be substituted into force balance equations to give

$$\begin{matrix}\left(0.45-\omega^2\right)X_1 & -0.2 & X_2 & & = 0\\ -0.24 & X_1+\left(0.42-\omega^2\right)X_2 & -0.18 & X_3 & = 0\\ & -0.225 & X_2+\left(0.225-\omega^2\right)X_3 & & = 0\end{matrix}$$

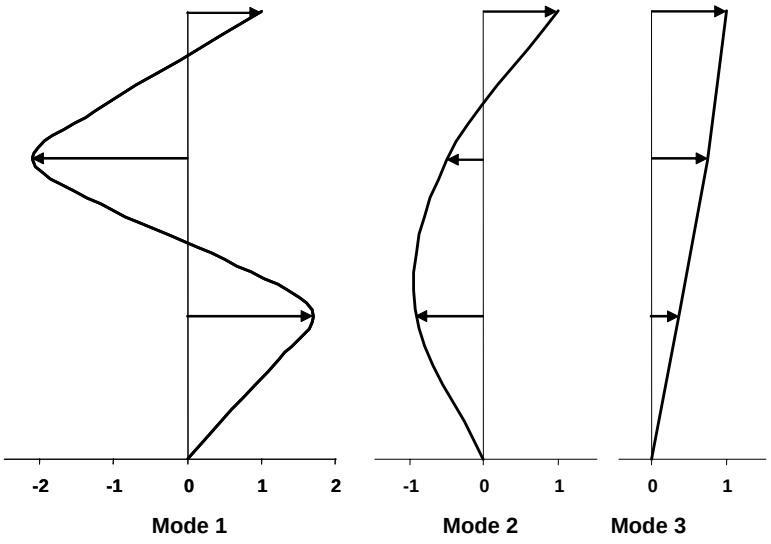
A MATLAB session can be conducted to evaluate the eigenvalues and eigenvectors as

```
>> A = [0.450 -0.200 0.000;-0.240 0.420 -0.180;0.000 -0.225 0.225];
>> [v,d] = eig(A)

v =
   -0.5879   -0.6344    0.2913
    0.7307   -0.3506    0.5725
   -0.3471    0.6890    0.7664
d =
    0.6986         0         0
         0    0.3395         0
         0         0    0.0569
```

Therefore, the eigenvalues are 0.6986, 0.3395 and 0.0569. The corresponding eigenvectors are (normalizing so that the amplitude for the third floor is one),

$$\begin{Bmatrix}1.693748\\-2.10516\\1\end{Bmatrix}\quad\begin{Bmatrix}-0.92075\\-0.50885\\1\end{Bmatrix}\quad\begin{Bmatrix}0.380089\\0.746999\\1\end{Bmatrix}$$

A graph can be made showing the three modes



20.6 As was done in Section 20.2, assume that the solution is $i_j = I_j \sin(\omega t)$. Therefore, the second derivative is

$$\frac{d^2 i_j}{dt^2} = -\omega^2 I_j \sin(\omega t)$$

Substituting these relationships into the differential equations gives

$$-L_1\omega^2 I_1 \sin(\omega t) + \frac{1}{C_1}(I_1 \sin(\omega t) - I_2 \sin(\omega t)) = 0$$
$$-L_2\omega^2 I_2 \sin(\omega t) + \frac{1}{0.001}(I_2 \sin(\omega t) - I_3 \sin(\omega t)) - \frac{1}{0.001}(I_1 \sin(\omega t) - I_2 \sin(\omega t)) = 0$$
$$-L_3\omega^2 I_3 \sin(\omega t) + \frac{1}{0.001} i_3 - \frac{1}{0.001}(I_2 \sin(\omega t) - I_3 \sin(\omega t)) = 0$$

All the $\sin(\omega t)$ terms can be cancelled. In addition, the L's and C's are constant. Therefore, the system simplifies to

$$\begin{bmatrix} 1-\lambda & -1 & 0 \\ -1 & 2-\lambda & -1 \\ 0 & -1 & 2-\lambda \end{bmatrix} \begin{Bmatrix} I_1 \\ I_2 \\ I_3 \end{Bmatrix} = 0$$

where $\lambda = LC\omega^2$. The following MATLAB session can then be used to evaluate the eigenvalues and eigenvectors

```
>> a = [1 -1 0;-1 2 -1;0 -1 2]
>> [v,d] = eig(a)
v =
   -0.7370   -0.5910    0.3280
   -0.5910    0.3280   -0.7370
   -0.3280    0.7370    0.5910
d =
    0.1981         0         0
```

```
         0    1.5550         0
         0         0    3.2470
```

The matrix `v` consists of the system's three eigenvectors (arranged as columns), and `d` is a matrix with the corresponding eigenvalues on the diagonal. Thus, MATLAB computes that the eigenvalues are λ = 0.1981, 1.5550, and 3.2470. These values in turn can be used to compute the natural frequencies for the system

$$\omega = \begin{Bmatrix} \dfrac{0.4450}{\sqrt{LC}} \\ \dfrac{1.2470}{\sqrt{LC}} \\ \dfrac{1.8019}{\sqrt{LC}} \end{Bmatrix}$$

20.7 The force balances can be written as

$$\begin{bmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_3 \end{bmatrix} \begin{Bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \\ \ddot{x}_3 \end{Bmatrix} + \begin{bmatrix} 2k & -k & -k \\ -k & 2k & -k \\ -k & -k & 2k \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = 0$$

Assuming that the solution is $x_i = X_i \sin(\omega t)$, we get the following matrix

$$\begin{bmatrix} 2k - m_1\omega^2 & -k & -k \\ -k & 2k - m_2\omega^2 & -k \\ -k & -k & 2k - m_3\omega^2 \end{bmatrix} \begin{Bmatrix} X_1 \\ X_2 \\ X_3 \end{Bmatrix} = 0$$

Using MATLAB,

```
>> k = 1;
>> kmw2 = [2*k,-k,-k;-k,2*k,-k;-k,-k,2*k];
>> [v,d] = eig(kmw2)

v =
    0.8034    0.1456    0.5774
   -0.2757   -0.7686    0.5774
   -0.5278    0.6230    0.5774

d =
    3.0000         0         0
         0    3.0000         0
         0         0    0.0000
```

Therefore, the eigenvalues are 0, 3, and 3. Setting these eigenvalues equal to $m\omega^2$, the three frequencies can be obtained.

$m\omega_1^{\,2} = 0 \Rightarrow \omega_1 = 0$ (Hz) 1st mode of oscillation

$m\omega_2^{\,2} = 0 \Rightarrow \omega_2 = \sqrt{3}$ (Hz) 2nd mode

$m\omega_3^{\,2} = 0 \Rightarrow \omega_3 = \sqrt{3}$ (Hz) 3rd mode

20.8 The pair of second-order differential equations can be reexpressed as a system of four first-order ODE's,

$$\frac{dx_1}{dt} = x_3$$

$$\frac{dx_2}{dt} = x_4$$

$$\frac{dx_3}{dt} = -5x_1 + 5(x_2 - x_1)$$

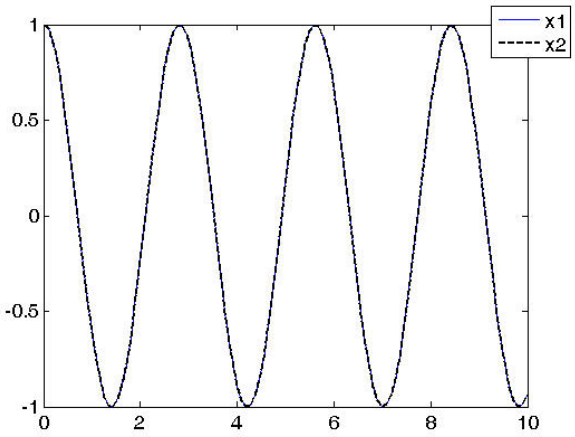
$$\frac{dx_4}{dt} = -5(x_2 - x_1) - 5x_2$$

An M-file can be set up to evaluate the right-hand side of these ODEs:

```
function dx = dxdt(t, x)
dx = [x(3);x(4);-5*x(1)+5*(x(2)-x(1));-5*(x(2)-x(1))-5*x(2)];
```

(a) $x_1 = x_2 = 1$

```
>> tspan = [0,10];
>> y0 = [1,1,0,0];
>> [t,y] = ode45('dxdt',tspan,y0);
>> plot(t,y(:,1),t,y(:,2),'--')
>> legend('x1','x2')
```

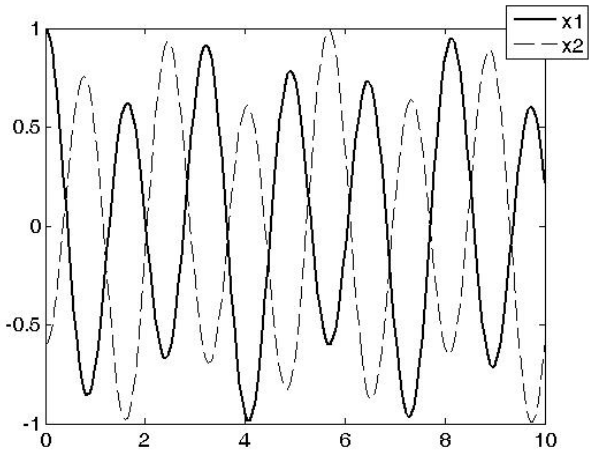


Because we have set the initial conditions consistent with one of the eigenvectors, the two masses oscillate in unison.

(b) $x_1 = 1$, $x_2 = -0.6$

```
>> tspan=[0,10];
>> y0=[1,-0.6,0,0];
```

```
>> [t,y]=ode45('dxdt',tspan,y0);
>> plot(t,y(:,1),t,y(:,2),'--')
>> legend('x1','x2')
```



Now, because the initial conditions do not correspond to one of the eigenvectors, the motion involves the superposition of both modes.

20.9

```
function [e, v] = powmax(A)
% [e, v] = powmax(A):
%   uses the power method to find the highest eigenvalue and
%   the corresponding eigenvector
% input:
%   A = matrix to be analyzed
% output:
%   e = eigenvalue
%   v = eigenvector

es = 0.0001;
maxit = 100;
n = size(A);
for i=1:n
  v(i)=1;
end
v = v';
e = 1;
iter = 0;
while (1)
  eold = e;
  x = A*v;
  [e,i] = max(abs(x));
  e = sign(x(i))*e;
  v = x/e;
  iter = iter + 1;
  ea = abs((e - eold)/e) * 100;
  if ea <= es | iter >= maxit, break, end
end
```

Application to solve Prob. 20.2,

```
>> A = [2 2 10;8 3 4;10 4 5];
>> [e, v] = powmax(A)
e =
   16.2741
v =
    0.8113
    0.7903
    1.0000
```

20.10

```
function [e, v] = powmin(A)
% [e, v] = powmin(A):
%   uses the power method to find the lowest eigenvalue and
%   the corresponding eigenvector
% input:
%   A = matrix to be analyzed
% output:
%   e = eigenvalue
%   v = eigenvector

es = 0.0001;
maxit = 100;
n = size(A);
for i=1:n
  v(i)=1;
end
v = v';
e = 1;
Ai = inv(A);
iter = 0;
while (1)
  eold = e;
  x = Ai*v;
  [e,i] = max(abs(x));
  e = sign(x(i))*e;
  v = x/e;
  iter = iter + 1;
  ea = abs((e - eold)/e) * 100;
  if ea <= es | iter >= maxit, break, end
end
e = 1./e;
```

Application to solve Prob. 20.2,

```
>> [e, v] = powmin(A)
e =
   -0.1815
v =
   -0.3341
    1.0000
   -0.1271
```