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 (a) For the case where the initial velocity is positive (downward), Eq. (1.21) is

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

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 $v = +v_0$ at $t = 0$, then

$$C = \frac{1}{a}\tanh^{-1}\frac{v_0}{a}$$

Substitute back into the solution

$$\frac{1}{a}\tanh^{-1}\frac{v}{a} = \frac{c_d}{m}t + \frac{1}{a}\tanh^{-1}\frac{v_0}{a}$$

Multiply both sides by a, taking the hyperbolic tangent of each side and substituting a gives,

$$v = \sqrt{\frac{mg}{c_d}}\tanh\left(\sqrt{\frac{gc_d}{m}}t + \tanh^{-1}\sqrt{\frac{c_d}{mg}}v_0\right) \qquad (1)$$

(b) For the case where the initial velocity is negative (upward), Eq. (1.21) is

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

Multiplying both sides of Eq. (1.8) by m/c_d and defining $a = \sqrt{mg / c_d}$ yields

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

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

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

Therefore, the integration yields

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

The initial condition, $v(0) = v_0$ gives

$$C = \frac{1}{a} \tan^{-1} \frac{v_0}{a}$$

Substituting this result back into the solution yields

$$\frac{1}{a} \tan^{-1} \frac{v}{a} = \frac{c_d}{m} t + \frac{1}{a} \tan^{-1} \frac{v_0}{a}$$

Multiplying both sides by a and taking the tangent gives

$$v = a \tan\left(a \frac{c_d}{m} t + \tan^{-1} \frac{v_0}{a} \right)$$

or substituting the values for a and simplifying gives

$$v = \sqrt{\frac{mg}{c_d}} \tan\left(\sqrt{\frac{gc_d}{m}} t + \tan^{-1} \sqrt{\frac{c_d}{mg}} v_0 \right) \qquad (2)$$

(c) We use Eq. (2) until the velocity reaches zero. Inspection of Eq. (2) indicates that this occurs when the argument of the tangent is zero. That is, when

$$\sqrt{\frac{gc_d}{m}} t_{zero} + \tan^{-1} \sqrt{\frac{c_d}{mg}} v_0 = 0$$

The time of zero velocity can then be computed as

$$t_{zero} = -\sqrt{\frac{m}{gc_d}} \tan^{-1} \sqrt{\frac{c_d}{mg}} v_0$$

Thereafter, the velocities can then be computed with Eq. (1.9),

$$v = \sqrt{\frac{mg}{c_d}} \tanh\left(\sqrt{\frac{gc_d}{m}}(t - t_{zero})\right) \qquad (3)$$

Here are the results for the parameters from Example 1.2, with an initial velocity of –40 m/s.

$$t_{zero} = -\sqrt{\frac{68.1}{9.81(0.25)}} \tan^{-1}\left(\sqrt{\frac{0.25}{68.1(9.81)}}(-40)\right) = 3.470239 \text{ s}$$

Therefore, for $t = 2$, we can use Eq. (2) to compute

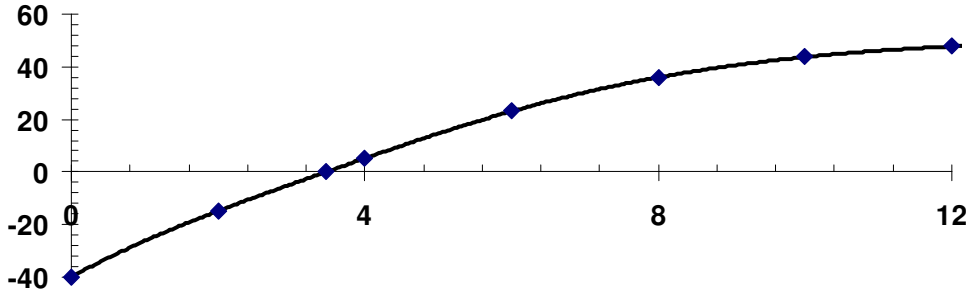
$$v = \sqrt{\frac{68.1(9.81)}{0.25}} \tan\left(\sqrt{\frac{9.81(0.25)}{68.1}}(2) + \tan^{-1}\sqrt{\frac{0.25}{68.1(9.81)}}(-40)\right) = -14.8093 \frac{\text{m}}{\text{s}}$$

For $t = 4$, the jumper is now heading downward and Eq. (3) applies

$$v = \sqrt{\frac{68.1(9.81)}{0.25}} \tanh\left(\sqrt{\frac{9.81(0.25)}{68.1}}(4 - 3.470239)\right) = 5.17952 \frac{\text{m}}{\text{s}}$$

The same equation is then used to compute the remaining values. The results for the entire calculation are summarized in the following table and plot:

t (s)	v (m/s)
0	-40
2	-14.8093
3.470239	0
4	5.17952
6	23.07118
8	35.98203
10	43.69242
12	47.78758



1.3 (*a*) This is a transient computation. For the period ending June 1:

Balance = Previous Balance + Deposits – Withdrawals + Interest

Balance = 1512.33 + 220.13 – 327.26 + 0.01(1512.33) = 1420.32

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

Date	Deposit	Withdrawal	Interest	Balance
1-May				$1,512.33
	$220.13	$327.26	$15.12	
1-Jun				$1,420.32
	$216.80	$378.61	$14.20	
1-Jul				$1,272.72
	$450.25	$106.80	$12.73	
1-Aug				$1,628.89
	$127.31	$350.61	$16.29	
1-Sep				**$1,421.88**

(*b*) $\dfrac{dB}{dt} = D(t) - W(t) + iB$

(*c*) for $t = 0$ to 0.5:

$$\frac{dB}{dt} = 220.13 - 327.26 + 0.01(1512.33) = -92.01$$

$$B(0.5) = 1512.33 - 92.01(0.5) = 1466.33$$

for $t = 0.5$ to 1:

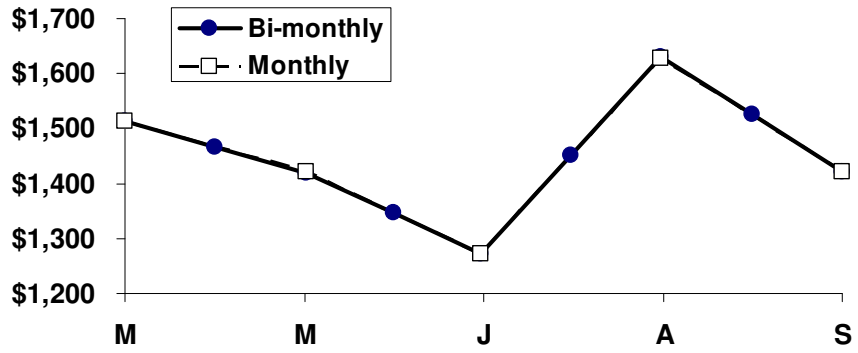
$$\frac{dB}{dt} = 220.13 - 327.260 + 0.01(1466.33) = -92.47$$

$$B(0.5) = 1466.33 - 92.47(0.5) = 1420.09$$

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

Date	Deposit	Withdrawal	Interest	*dB*/*dt*	Balance
1-May	$220.13	$327.26	$15.12	-$92.01	$1,512.33
16-May	$220.13	$327.26	$14.66	-$92.47	$1,466.33
1-Jun	$216.80	$378.61	$14.20	-$147.61	$1,420.09
16-Jun	$216.80	$378.61	$13.46	-$148.35	$1,346.29
1-Jul	$450.25	$106.80	$12.72	$356.17	$1,272.12
16-Jul	$450.25	$106.80	$14.50	$357.95	$1,450.20
1-Aug	$127.31	$350.61	$16.29	-$207.01	$1,629.18
16-Aug	$127.31	$350.61	$15.26	-$208.04	$1,525.67
1-Sep					**$1,421.65**

(*d*) As in the plot below, the results of the two approaches are very close.



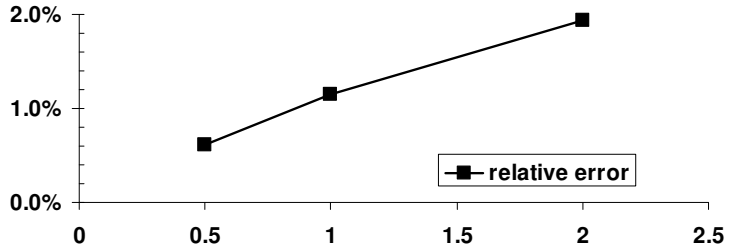
1.4 At $t = 12$ s, the analytical solution is 50.6175 (Example 1.1). The numerical results are:

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.

1.5 (*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

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

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

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

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

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

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

Alternative solution: Another way to obtain solutions is to use separation of variables,

$$\int \frac{1}{g - \frac{c'}{m} v} dv = \int dt$$

The integrals can be evaluated as

$$-\frac{\ln\left(g - \frac{c'}{m} v\right)}{c'/m} = t + C$$

where C = a constant of integration, which can be evaluated by applying the initial condition

$$C = -\frac{\ln\left(g - \frac{c'}{m} v(0)\right)}{c'/m}$$

which can be substituted back into the solution

$$-\frac{\ln\left(g-\frac{c'}{m}v\right)}{c'/m}=t-\frac{\ln\left(g-\frac{c'}{m}v(0)\right)}{c'/m}$$

This result can be rearranged algebraically to solve for v,

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

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

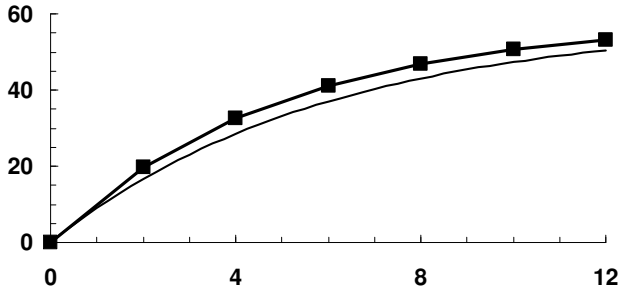
$$\boxed{v=\frac{mg}{c'}\left(1-e^{-(c'/m)t}\right)}$$

(b) The numerical solution can be implemented as

$$v(2)=0+\left[9.81-\frac{12.5}{68.1}(0)\right]2=19.62$$
$$v(4)=19.62+\left[9.81-\frac{12.5}{68.1}(19.62)\right]2=32.0374$$

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

t	*v*	*dv/dt*
0	0	9.81
2	19.6200	6.4968
4	32.6136	4.3026
6	41.2187	2.8494
8	46.9176	1.8871
10	50.6917	1.2497
12	53.1911	0.8276
∞	58.0923	



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

1.6 $v(t)=\frac{gm}{c'}(1-e^{-(c'/m)t})$

jumper #1: $v(t)=\frac{9.81(70)}{12}(1-e^{-(12/70)9})=44.99204\frac{\text{m}}{\text{s}}$

jumper #2: $44.99204 = \dfrac{9.81(80)}{15}(1 - e^{-(15/80)t})$

$$44.99204 = 52.32 - 52.32e^{-0.1875t}$$
$$0.14006 = e^{-0.1875t}$$
$$t = \frac{\ln 0.14006}{-0.1875} = 10.4836 \text{ s}$$

1.7 Note that the differential equation should be formulated as

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

This ensures that the sign of the drag is correct when the parachutist has a negative upward velocity. Before the chute opens ($t < 10$), Euler's method can be implemented as

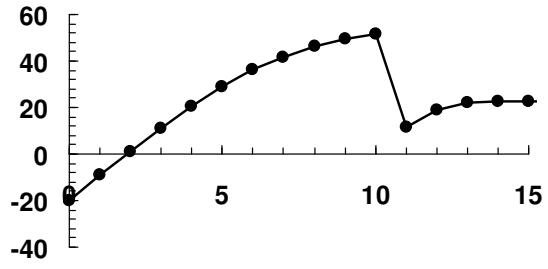
$$v(t + \Delta t) = v(t) + \left[9.81 - \frac{0.25}{80} v|v|\right]\Delta t$$

After the chute opens ($t \geq 10$), the drag coefficient is changed and the implementation becomes

$$v(t + \Delta t) = v(t) + \left[9.81 - \frac{1.5}{80} v|v|\right]\Delta t$$

Here is a summary of the results along with a plot:

Chute closed			Chute opened		
t	***v***	***dv/dt***	***t***	***v***	***dv/dt***
0	-20.0000	11.0600	10	51.5260	-39.9698
1	-8.9400	10.0598	11	11.5561	7.3060
2	1.1198	9.8061	12	18.8622	3.1391
3	10.9258	9.4370	13	22.0013	0.7340
4	20.3628	8.5142	14	22.7352	0.1183
5	28.8770	7.2041	15	22.8535	0.0172
6	36.0812	5.7417	16	22.8707	0.0025
7	41.8229	4.3439	17	22.8732	0.0003
8	46.1668	3.1495	18	22.8735	0.0000
9	49.3162	2.2097	19	22.8736	0.0000
			20	22.8736	0.0000



1.8 (a) The first two steps are

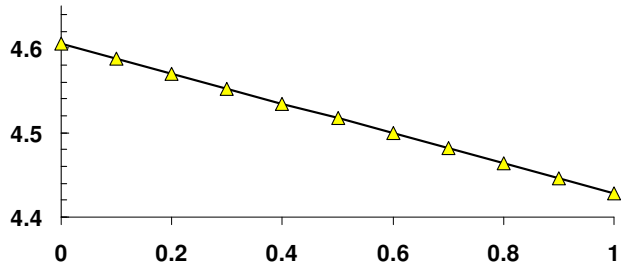
$$c(0.1) = 100 - 0.175(10)0.1 = 98.25 \text{ Bq/L}$$

$$c(0.2) = 98.25 - 0.175(98.25)0.1 = 96.5306 \text{ Bq/L}$$

The process can be continued to yield

t	*c*	*dc/dt*
0	100.0000	-17.5000
0.1	98.2500	-17.1938
0.2	96.5306	-16.8929
0.3	94.8413	-16.5972
0.4	93.1816	-16.3068
0.5	91.5509	-16.0214
0.6	89.9488	-15.7410
0.7	88.3747	-15.4656
0.8	86.8281	-15.1949
0.9	85.3086	-14.9290
1	83.8157	-14.6678

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



The slope of this line can be estimated as

$$\frac{\ln(83.8157) - \ln(100)}{1} = -0.17655$$

Thus, the slope is approximately equal to the negative of the decay rate. If we had used a smaller step size, the result would be more exact.

1.9 The first two steps yield

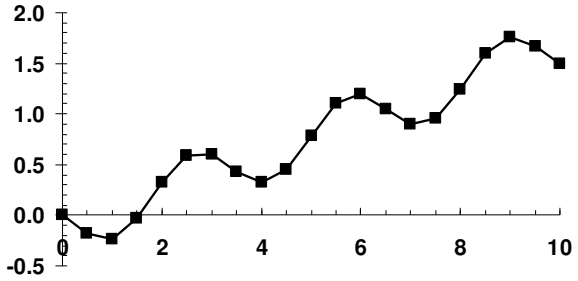
$$y(0.5) = 0 + \left[3\frac{450}{1250}\sin^2(0) - \frac{450}{1250}\right]0.5 = \ 0 + (-0.36)\ 0.5 = -0.18$$

$$y(1) = -0.18 + \left[3\frac{450}{1250}\sin^2(0.5) - \frac{450}{1250}\right]0.5 = \ -0.18 + (-0.11176)\ 0.5 = -0.23508$$

The process can be continued to give the following table and plot:

t	*y*	*dy/dt*	*t*	*y*	*dy/dt*
0	0.00000	-0.36000	5.5	1.10271	0.17761
0.5	-0.18000	-0.11176	6	1.19152	-0.27568
1	-0.23588	0.40472	6.5	1.05368	-0.31002
1.5	-0.03352	0.71460	7	0.89866	0.10616
2	0.32378	0.53297	7.5	0.95175	0.59023
2.5	0.59026	0.02682	8	1.24686	0.69714
3	0.60367	-0.33849	8.5	1.59543	0.32859
3.5	0.43443	-0.22711	9	1.75972	-0.17657

4	0.32087	0.25857	9.5	1.67144	-0.35390
4.5	0.45016	0.67201	10	1.49449	-0.04036
5	0.78616	0.63310			



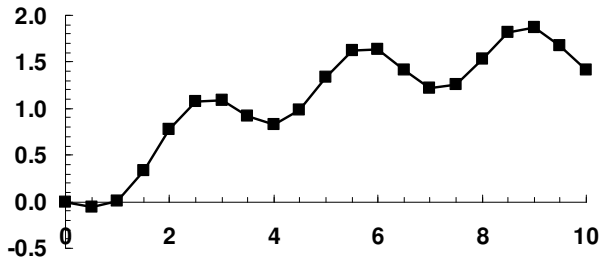
1.10 The first two steps yield

$$y(0.5) = 0 + \left[3\frac{450}{1250}\sin^2(0) - \frac{150(1+0)^{1.5}}{1250}\right]0.5 = 0 - 0.12(0.5) = -0.06$$

$$y(1) = -0.06 + \left[3\frac{450}{1250}\sin^2(0.5) - \frac{150(1-0.06)^{1.5}}{1250}\right]0.5 = -0.06 + 0.13887(0.5) = 0.00944$$

The process can be continued to give

t	*y*	*dy/dt*	*t*	*y*	*dy/dt*
0	0.00000	-0.12000	5.5	1.61981	0.02876
0.5	-0.06000	0.13887	6	1.63419	-0.42872
1	0.00944	0.64302	6.5	1.41983	-0.40173
1.5	0.33094	0.89034	7	1.21897	0.06951
2	0.77611	0.60892	7.5	1.25372	0.54423
2.5	1.08058	0.02669	8	1.52584	0.57542
3	1.09392	-0.34209	8.5	1.81355	0.12227
3.5	0.92288	-0.18708	9	1.87468	-0.40145
4	0.82934	0.32166	9.5	1.67396	-0.51860
4.5	0.99017	0.69510	10	1.41465	-0.13062
5	1.33772	0.56419			



1.11 When the water level is above the outlet pipe, the volume balance can be written as

$$\frac{dV}{dt} = 3\sin^2(t) - 3(y - y_{\text{out}})^{1.5}$$

In order to solve this equation, we must relate the volume to the level. To do this, we recognize that the volume of a cone is given by $V = \pi r^2 y/3$. Defining the side slope as $s = y_{top}/r_{top}$, the radius can be related to the level ($r = y/s$) and the volume can be reexpressed as

$$V = \frac{\pi}{3s^2} y^3$$

which can be solved for

$$y = \sqrt[3]{\frac{3s^2 V}{\pi}} \tag{1}$$

and substituted into the volume balance

$$\frac{dV}{dt} = 3\sin^2(t) - 3\left(\sqrt[3]{\frac{3s^2 V}{\pi}} - y_{out}\right)^{1.5} \tag{2}$$

For the case where the level is below the outlet pipe, outflow is zero and the volume balance simplifies to

$$\frac{dV}{dt} = 3\sin^2(t) \tag{3}$$

These equations can then be used to solve the problem. Using the side slope of $s = 4/2.5 = 1.6$, the initial volume can be computed as

$$V(0) = \frac{\pi}{3(1.6)^2} 0.8^3 = 0.20944 \text{ m}^3$$

For the first step, $y < y_{out}$ and Eq. (3) gives

$$\frac{dV}{dt}(0) = 3\sin^2(0) = 0$$

and Euler's method yields

$$V(0.5) = V(0) + \frac{dV}{dt}(0)\Delta t = 0.20944 + 0(0.5) = 0.20944$$

For the second step, Eq. (3) still holds and

$$\frac{dV}{dt}(0.5) = 3\sin^2(0.5) = 0.689547$$

$$V(1) = V(0.5) + \frac{dV}{dt}(0.5)\Delta t = 0.20944 + 0.689547(0.5) = 0.554213$$

Equation (1) can then be used to compute the new level,

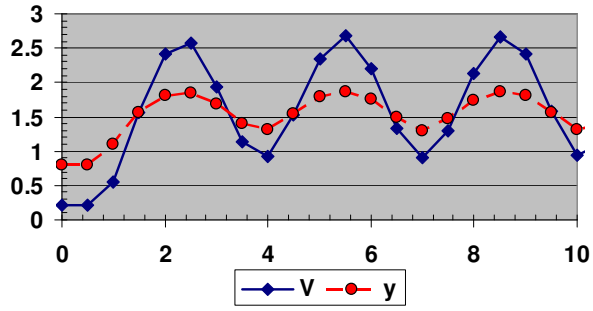
$$y = \sqrt[3]{\frac{3(1.6)^2(0.554213)}{\pi}} = 1.106529 \text{ m}$$

Because this level is now higher than the outlet pipe, Eq. (2) holds for the next step

$$\frac{dV}{dt}(1) = 2.12422 - 3\left(1.106529 - 1\right)^{1.5} = 2.019912$$
$$V(1.5) = 0.554213 + 2.019912(0.5) = 2.984989$$

The remainder of the calculation is summarized in the following table and figure.

t	Q_{in}	*V*	*y*	Q_{out}	*dV/dt*
0	0	0.20944	0.8	0	0
0.5	0.689547	0.20944	0.8	0	0.689547
1	2.12422	0.554213	1.106529	0.104309	2.019912
1.5	2.984989	1.564169	1.563742	1.269817	1.715171
2	2.480465	2.421754	1.809036	2.183096	0.29737
2.5	1.074507	2.570439	1.845325	2.331615	-1.25711
3	0.059745	1.941885	1.680654	1.684654	-1.62491
3.5	0.369147	1.12943	1.40289	0.767186	-0.39804
4	1.71825	0.93041	1.31511	0.530657	1.187593
4.5	2.866695	1.524207	1.55031	1.224706	1.641989
5	2.758607	2.345202	1.78977	2.105581	0.653026
5.5	1.493361	2.671715	1.869249	2.431294	-0.93793
6	0.234219	2.202748	1.752772	1.95937	-1.72515
6.5	0.13883	1.340173	1.48522	1.013979	-0.87515
7	1.294894	0.902598	1.301873	0.497574	0.79732
7.5	2.639532	1.301258	1.470703	0.968817	1.670715
8	2.936489	2.136616	1.735052	1.890596	1.045893
8.5	1.912745	2.659563	1.866411	2.419396	-0.50665
9	0.509525	2.406237	1.805164	2.167442	-1.65792
9.5	0.016943	1.577279	1.568098	1.284566	-1.26762
10	0.887877	0.943467	1.321233	0.5462	0.341677



1.12

$$Q_{\text{students}} = 35 \text{ ind} \times 80\frac{\text{J}}{\text{ind s}} \times 20 \text{ min} \times 60\frac{\text{s}}{\text{min}} \times \frac{\text{kJ}}{1000 \text{ J}} = 3{,}360 \text{ kJ}$$

$$m = \frac{PV\text{Mwt}}{RT} = \frac{(101.325 \text{ kPa})(11\text{m} \times 8\text{m} \times 3\text{m} - 35 \times 0.075 \text{ m}^3)(28.97 \text{ kg/kmol})}{(8.314 \text{ kPa m}^3/(\text{kmol K})((20 + 273.15)\text{K})} = 314.796 \text{ kg}$$

$$\Delta T = \frac{Q_{\text{students}}}{mC_v} = \frac{3{,}360 \text{ kJ}}{(314.796 \text{ kg})(0.718 \text{ kJ/(kg K)})} = 14.86571 \text{ K}$$

Therefore, the final temperature is 20 + 14.86571 = 34.86571°C**.**

1.13 $\sum M_{in} - \sum M_{out} = 0$

$$\text{Food} + \text{Drink} + \text{Air In} + \text{Metabolism} = \text{Urine} + \text{Skin} + \text{Feces} + \text{Air Out} + \text{Sweat}$$
$$\text{Drink} = \text{Urine} + \text{Skin} + \text{Feces} + \text{Air Out} + \text{Sweat} - \text{Food} - \text{Air In} - \text{Metabolism}$$
$$\text{Drink} = 1.4 + 0.35 + 0.2 + 0.4 + 0.3 - 1 - 0.05 - 0.3 = 1.3 \text{ L}$$

1.14 (a) The force balance can be written as:

$$m\frac{dv}{dt} = -mg(0)\frac{R^2}{(R+x)^2} + c_d v|v|$$

Dividing by mass gives

$$\frac{dv}{dt} = -g(0)\frac{R^2}{(R+x)^2} + \frac{c_d}{m} v|v|$$

(b) Recognizing that $dx/dt = v$, the chain rule is

$$\frac{dv}{dt} = v\frac{dv}{dx}$$

Setting drag to zero and substituting this relationship into the force balance gives

$$\frac{dv}{dx} = -\frac{g(0)}{v}\frac{R^2}{(R+x)^2}$$

(c) Using separation of variables

$$v\,dv = -g(0)\frac{R^2}{(R+x)^2}dx$$

Integrating gives

$$\frac{v^2}{2} = g(0)\frac{R^2}{R+x} + C$$

Applying the initial condition yields

$$\frac{v_0^2}{2} = g(0)\frac{R^2}{R+0} + C$$

which can be solved for $C = v_0^2/2 - g(0)R$, which can be substituted back into the solution to give

$$\frac{v^2}{2} = g(0)\frac{R^2}{R+x} + \frac{v_0^2}{2} - g(0)R$$

or

$$v = \pm\sqrt{v_0^2 + 2g(0)\frac{R^2}{R+x} - 2g(0)R}$$

Note that the plus sign holds when the object is moving upwards and the minus sign holds when it is falling.

(d) Euler's method can be developed as

$$v(x_{i+1}) = v(x_i) + \left[-\frac{g(0)}{v(x_i)}\frac{R^2}{(R+x_i)^2}\right](x_{i+1} - x_i)$$

The first step can be computed as

$$v(10,000) = 1,500 + \left[-\frac{9.81}{1,500}\frac{(6.37\times10^6)^2}{(6.37\times10^6+0)^2}\right](10,000-0) = 1,500 + (-0.00654)10,000 = 1434.600$$

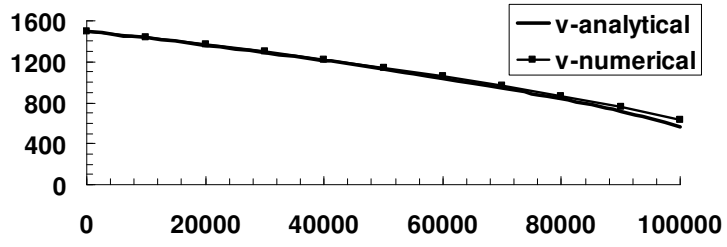
The remainder of the calculations can be implemented in a similar fashion as in the following table

x	*v*	*dv*/*dx*	*v*-analytical
0	1500.000	-0.00654	1500.000
10000	1434.600	-0.00682	1433.216
20000	1366.433	-0.00713	1363.388
30000	1295.089	-0.00750	1290.023
40000	1220.049	-0.00794	1212.475
50000	1140.643	-0.00847	1129.884
60000	1055.973	-0.00912	1041.049
70000	964.798	-0.00995	944.206
80000	865.317	-0.01106	836.579
90000	754.742	-0.01264	713.299
100000	628.359	-0.01513	564.197

For the analytical solution, the value at 10,000 m can be computed as

$$v = \sqrt{1,500^2 + 2(9.81)\frac{(6.37\times10^6)^2}{(6.37\times10^6+10,000)} - 2(9.81)(6.37\times10^6)} = 1433.216$$

The remainder of the analytical values can be implemented in a similar fashion as in the last column of the above table. The numerical and analytical solutions can be displayed graphically.



1.15 The volume of the droplet is related to the radius as

$$V = \frac{4\pi r^3}{3} \tag{1}$$

This equation can be solved for radius as

$$r = \sqrt[3]{\frac{3V}{4\pi}} \tag{2}$$

The surface area is

$$A = 4\pi r^2 \tag{3}$$

Equation (2) can be substituted into Eq. (3) to express area as a function of volume

$$A = 4\pi\left(\frac{3V}{4\pi}\right)^{2/3}$$

This result can then be substituted into the original differential equation,

$$\frac{dV}{dt} = -k4\pi\left(\frac{3V}{4\pi}\right)^{2/3} \tag{4}$$

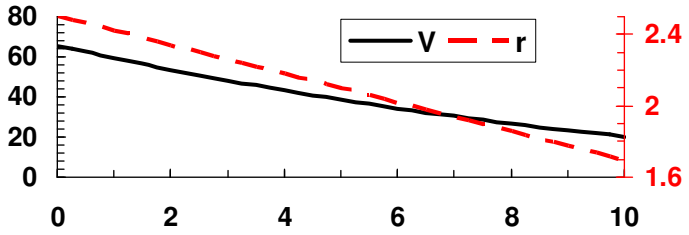
The initial volume can be computed with Eq. (1),

$$V = \frac{4\pi r^3}{3} = \frac{4\pi(2.5)^3}{3} = 65.44985 \text{ mm}^3$$

Euler's method can be used to integrate Eq. (4). Here are the beginning and last steps

t	*V*	*dV/dt*
0	65.44985	-6.28319
0.25	63.87905	-6.18225
0.5	62.33349	-6.08212
0.75	60.81296	-5.98281
1	59.31726	-5.8843
•		
•		
•		
9	23.35079	-3.16064
9.25	22.56063	-3.08893
9.5	21.7884	-3.01804
9.75	21.03389	-2.94795
10	20.2969	-2.87868

A plot of the results is shown below. We have included the radius on this plot (dashed line and right scale):



Eq. (2) can be used to compute the final radius as

$$r = \sqrt[3]{\frac{3(20.2969)}{4\pi}} = 1.692182$$

Therefore, the average evaporation rate can be computed as

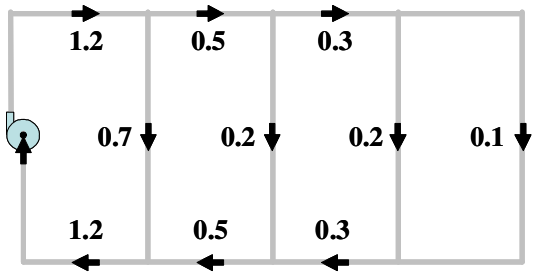
$$k = \frac{(2.5 - 1.692182)\text{ mm}}{10\text{ min}} = 0.080782\frac{\text{mm}}{\text{min}}$$

which is approximately equal to the given evaporation rate of 0.08 mm/min.

1.16 Continuity at the nodes can be used to determine the flows as follows:

$$Q_1 = Q_2 + Q_3 = 0.7 + 0.5 = 1.2 \text{ m}^3/\text{s}$$
$$Q_{10} = Q_1 = 1.2 \text{ m}^3/\text{s}$$
$$Q_9 = Q_{10} - Q_2 = 1.2 - 0.7 = 0.5 \text{ m}^3/\text{s}$$
$$Q_4 = Q_9 - Q_8 = 0.5 - 0.3 = 0.2 \text{ m}^3/\text{s}$$
$$Q_5 = Q_3 - Q_4 = 0.5 - 0.2 = 0.3 \text{ m}^3/\text{s}$$
$$Q_6 = Q_5 - Q_7 = 0.3 - 0.1 = 0.2 \text{ m}^3/\text{s}$$

Therefore, the final results are

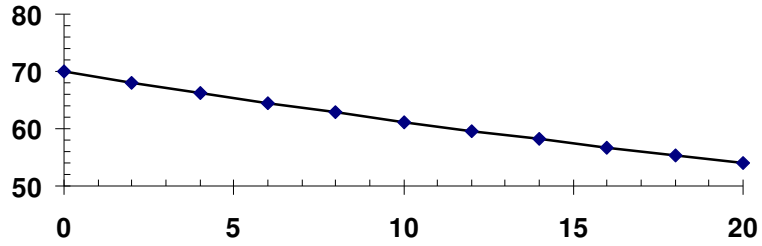


1.17 The first two steps can be computed as

$$T(1) = 70 + \left[-0.019(70 - 20)\right] 2 = 68 + (-0.95)2 = 68.1$$
$$T(2) = 68.1 + \left[-0.019(68.1 - 20)\right] 2 = 68.1 + (-0.9139)2 = 66.2722$$

The remaining results are displayed below along with a plot of the results.

t	*T*	*dT/dt*	*t*	*T*	*dT/dt*
0	70.00000	-0.95000	12.00000	59.62967	-0.75296
2	68.10000	-0.91390	14.00000	58.12374	-0.72435
4	66.27220	-0.87917	16.00000	56.67504	-0.69683
6	64.51386	-0.84576	18.00000	55.28139	-0.67035
8	62.82233	-0.81362	20.00000	53.94069	-0.64487
10	61.19508	-0.78271			



1.18 (a) For the constant temperature case, Newton's law of cooling is written as

$$\frac{dT}{dt} = -0.135(T - 10)$$

The first two steps of Euler's methods are

$$T(0.5) = T(0) - \frac{dT}{dt}(0) \times \Delta t = 37 + 0.12(10 - 37)(0.5) = 37 - 3.2400 \times 0.50 = 35.3800$$

$$T(1) = 35.3800 + 0.12(10 - 35.3800)(0.5) = 35.3800 - 3.0456 \times 0.50 = 33.8572$$

The remaining calculations are summarized in the following table:

t	T_a	*T*	*dT/dt*
0:00	10	37.0000	-3.2400
0:30	10	35.3800	-3.0456
1:00	10	33.8572	-2.8629
1:30	10	32.4258	-2.6911
2:00	10	31.0802	-2.5296
2:30	10	29.8154	-2.3778
3:00	10	28.6265	-2.2352
3:30	10	27.5089	-2.1011
4:00	10	26.4584	-1.9750
4:30	10	25.4709	-1.8565
5:00	10	24.5426	-1.7451

(b) For this case, the room temperature can be represented as

$$T_a = 20 - 2t$$

where t = time (hrs). Newton's law of cooling is written as

$$\frac{dT}{dt} = -0.12(T - 20 + 2t)$$

The first two steps of Euler's methods are

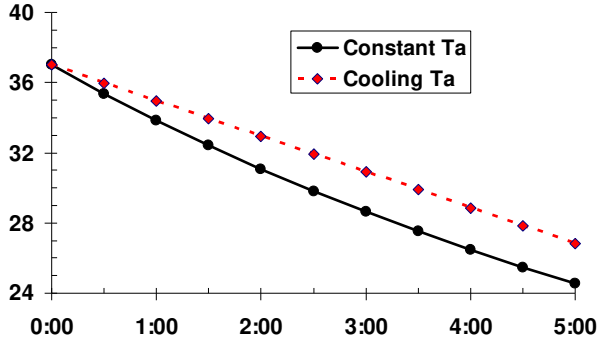
$$T(0.5) = 37 + 0.12(20 - 37)(0.5) = 37 - 2.040 \times 0.50 = 35.9800$$
$$T(1) = 35.9800 + 0.12(19 - 35.9800)(0.5) = 35.9800 - 2.0376 \times 0.50 = 34.9612$$

The remaining calculations are summarized in the following table:

t	T_a	*T*	*dT/dt*
0:00	20	37.0000	-2.0400
0:30	19	35.9800	-2.0376
1:00	18	34.9612	-2.0353
1:30	17	33.9435	-2.0332
2:00	16	32.9269	-2.0312
2:30	15	31.9113	-2.0294
3:00	14	30.8966	-2.0276
3:30	13	29.8828	-2.0259
4:00	12	28.8699	-2.0244
4:30	11	27.8577	-2.0229
5:00	10	26.8462	-2.0215

Comparison with (a) indicates that the effect of the room air temperature has a significant effect on the expected temperature at the end of the 5-hr period (difference = 26.8462 – 24.5426 = 2.3036°C).

(c) The solutions for (a) Constant T_a, and (b) Cooling T_a are plotted below:



1.19 The two equations to be solved are

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

Euler's method can be applied for the first step as

$$v(2) = v(0) + \frac{dv}{dt}(0)\Delta t = 0 + \left(9.81 - \frac{0.25}{68.1}(0)^2\right)(2) = 19.6200$$
$$x(2) = x(0) + \frac{dx}{dt}(0)\Delta t = 0 + 0(2) = 0$$

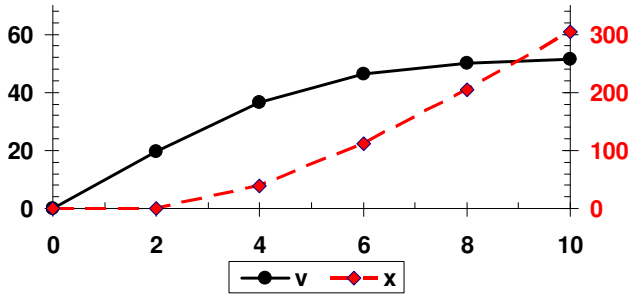
For the second step:

$$v(4) = v(2) + \frac{dv}{dt}(2)\Delta t = 19.6200 + \left(9.81 - \frac{0.25}{68.1}(19.6200)^2\right)(2) = 19.6200 + 8.3968(2) = 36.4137$$

$$x(4) = x(2) + \frac{dx}{dt}(2)\Delta t = 0 + 19.6200(2) = 39.2400$$

The remaining steps can be computed in a similar fashion as tabulated and plotted below:

t	*x*	*v*	*dx/dt*	*dv/dt*
0	0.0000	0.0000	0.0000	9.8100
2	0.0000	19.6200	19.6200	8.3968
4	39.2400	36.4137	36.4137	4.9423
6	112.0674	46.2983	46.2983	1.9409
8	204.6640	50.1802	50.1802	0.5661
10	305.0244	51.3123	51.3123	0.1442



1.20 (a) The force balance with buoyancy can be written as

$$m\frac{dv}{dt} = mg - \frac{1}{2}\rho v|v|AC_d - \rho Vg$$

Divide both sides by mass,

$$\frac{dv}{dt} = g\left(1 - \frac{\rho V}{m}\right) - \frac{\rho AC_d}{2m}v|v|$$

(b) For a sphere, the mass is related to the volume as in $m = \rho_s V$ where ρ_s = the sphere's density (kg/m^3). Substituting this relationship gives

$$\frac{dv}{dt} = g\left(1 - \frac{\rho}{\rho_s}\right) - \frac{\rho AC_d}{2\rho_s V}v|v|$$

The formulas for the volume and projected area can be substituted to give

$$\frac{dv}{dt} = g\left(1 - \frac{\rho}{\rho_s}\right) - \frac{3\rho C_d}{4\rho_s d}v|v|$$

(c) At steady state ($dv/dt = 0$),

$$g\left(\frac{\rho_s - \rho}{\rho_s}\right) = \frac{3\rho C_d}{4\rho_s d}v^2$$

which can be solved for the terminal velocity

$$v = \sqrt{\frac{4gd}{3C_d}\left(\frac{\rho_s - \rho}{\rho}\right)}$$

(d) Before implementing Euler's method, the parameters can be substituted into the differential equation to give

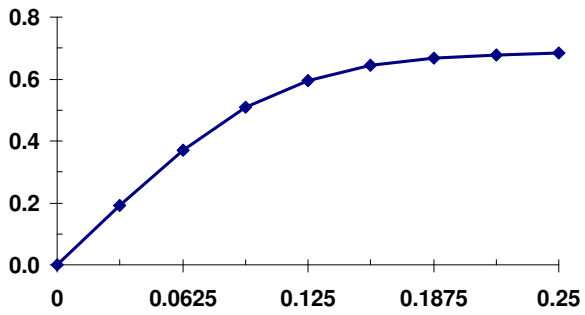
$$\frac{dv}{dt} = 9.81\left(1 - \frac{1000}{2700}\right) - \frac{3(1000)0.47}{4(2700)(0.01)}v^2 = 6.176667 - 13.055556v^2$$

The first two steps for Euler's method are

$$v(0.03125) = 0 + (6.176667 - 13.055556(0)^2)0.03125 = 0.193021$$
$$v(0.0625) = 0.193021 + (6.176667 - 13.055556(0.193021)^2)0.03125 = 0.370841$$

The remaining steps can be computed in a similar fashion as tabulated and plotted below:

t	*v*	*dv/dt*
0	0.000000	6.176667
0.03125	0.193021	5.690255
0.0625	0.370841	4.381224
0.09375	0.507755	2.810753
0.125	0.595591	1.545494
0.15625	0.643887	0.763953
0.1875	0.667761	0.355136
0.21875	0.678859	0.160023
0.25	0.683860	0.071055



CHAPTER 2

2.1 (a)

```
>> t = linspace(4,34,6)
t =
     4    10    16    22    28    34
```

(b)

```
>> x = linspace(-4,2,7)
x =
    -4    -3    -2    -1     0     1     2
```

2.2 (a)

```
>> v = -2:0.5:1.5
v =
   -2.0000 -1.5000 -1.0000 -0.5000       0  0.5000  1.0000  1.5000
```

(b)

```
>> r = 8:-0.5:4.5
r =
    8.0000  7.5000  7.0000  6.5000  6.0000  5.5000  5.0000  4.5000
```

2.3 The command `linspace(a,b,n)` is equivalent to the colon notation

```
>> a:(b-a)/(n-1):b
```

Test case:

```
>> a=-3;b=5;n=6;
>> linspace(a,b,n)
ans =
   -3.0000   -1.4000    0.2000    1.8000    3.4000    5.0000
>> a:(b-a)/(n-1):b
ans =
   -3.0000   -1.4000    0.2000    1.8000    3.4000    5.0000
```

2.4 (a)

```
>> A=[3 2 1;0:0.5:1;linspace(6, 8, 3)]
A =
    3.0000    2.0000    1.0000
         0    0.5000    1.0000
    6.0000    7.0000    8.0000
```

(b)

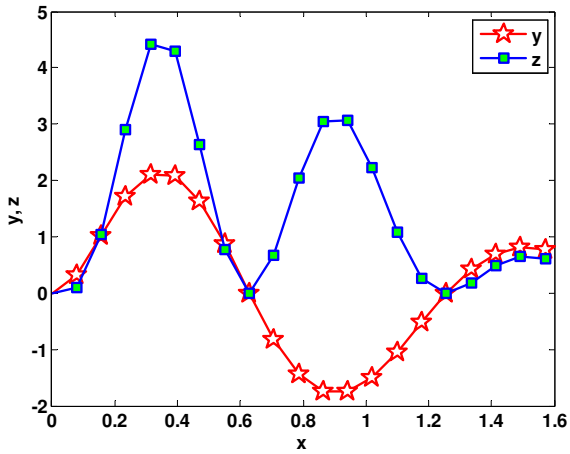
```
>> C=A(2,:)*A(:,3)
C =
   8.5
```

2.5

```
format short g
a=2;b=5;
x=0:pi/40:pi/2;
y=b*exp(-a*x).*sin(b*x).*(0.012*x.^4-0.15*x.^3+0.075*x.^2+2.5*x);
z=y.^2;
w = [x' y' z']
plot(x,y,'-.pr','LineWidth',1.5,'MarkerSize',14,...
    'MarkerEdgeColor','r','MarkerFaceColor','w')
hold on
plot(x,z,'-sb','MarkerFaceColor','g')
xlabel('x'); ylabel('y, z'); legend('y','z')
hold off
```

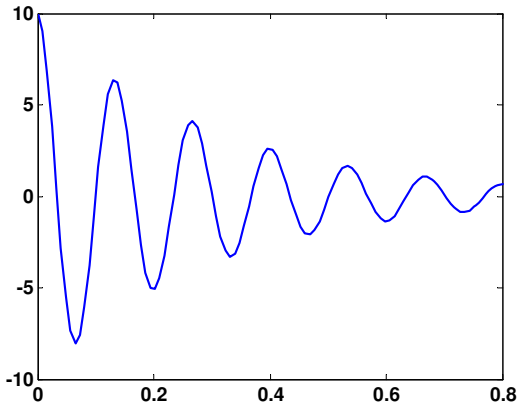
Output:

```
w =
            0            0            0
      0.07854      0.32172      0.10351
      0.15708       1.0174       1.0351
      0.23562        1.705       2.9071
      0.31416       2.1027       4.4212
       0.3927       2.0735       4.2996
      0.47124       1.6252       2.6411
      0.54978      0.87506      0.76573
      0.62832  2.7275e-016  7.4392e-032
      0.70686     -0.81663      0.66689
       0.7854       -1.427       2.0365
      0.86394      -1.7446       3.0437
      0.94248      -1.7512       3.0667
        1.021      -1.4891       2.2173
       1.0996      -1.0421       1.0859
       1.1781     -0.51272      0.26288
       1.2566 -2.9683e-016   8.811e-032
       1.3352      0.41762       0.1744
       1.4137      0.69202       0.4789
       1.4923      0.80787      0.65265
       1.5708      0.77866      0.60631
```



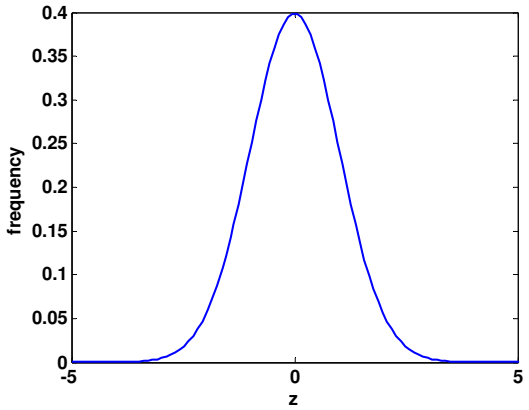
2.6

```
>> q0 = 10;R = 60;L = 9;C = 0.00005;
>> t = linspace(0,.8);
>> q = q0*exp(-R*t/(2*L)).*cos(sqrt(1/(L*C)-(R/(2*L))^2)*t);
>> plot(t,q)
```



2.7

```
>> z = linspace(-4,4);
>> f = 1/sqrt(2*pi)*exp(-z.^2/2);
>> plot(z,f)
>> xlabel('z')
>> ylabel('frequency')
```

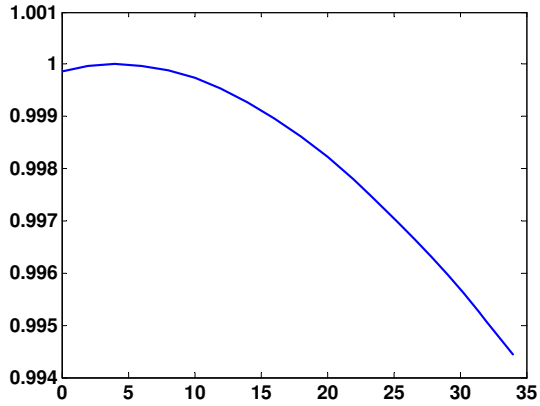


2.8

```
>> F = [14 18 8 9 13];
>> x = [0.013 0.020 0.009 0.010 0.012];
>> k = F./x
k =
  1.0e+003 *
    1.0769    0.9000    0.8889    0.9000    1.0833
>> U = .5*k.*x.^2
U =
    0.0910    0.1800    0.0360    0.0450    0.0780
>> max(U)
ans =
    0.1800
```

2.9

```
>> TF = 32:3.6:82.4;
>> 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)
```

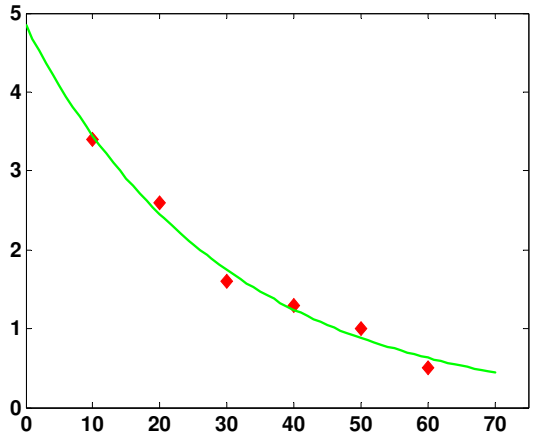


2.10

```
>> A = [.035 .0001 10 2;
0.02 0.0002 8 1;
0.015 0.001 20 1.5;
0.03 0.0007 24 3;
0.022 0.0003 15 2.5]
A =
        0.035       0.0001           10            2
         0.02       0.0002            8            1
        0.015        0.001           20          1.5
         0.03       0.0007           24            3
        0.022       0.0003           15          2.5
>> U = sqrt(A(:,2))./A(:,1).*(A(:,3).*A(:,4)./(A(:,3)+2*A(:,4))).^(2/3)
U =
      0.36241
      0.60937
       2.5167
       1.5809
       1.1971
```

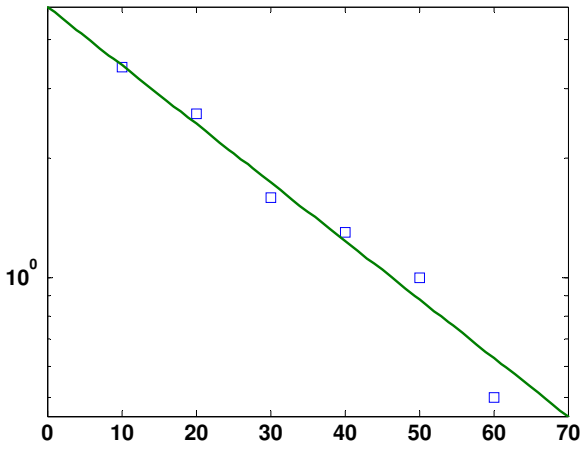
2.11

```
>> 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,'d','MarkerEdgeColor','r','MarkerFaceColor','r')
>> hold on
>> plot(tf,cf,'--g')
>> xlim([0 75])
>> hold off
```



2.12

```
>> 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,':')
```



The result is a straight line. The reason for this outcome can be understood by taking the natural (Naperian or base-e) logarithm of the function to give,

$$\ln c = \ln 4.84 + \ln e^{-0.034t}$$

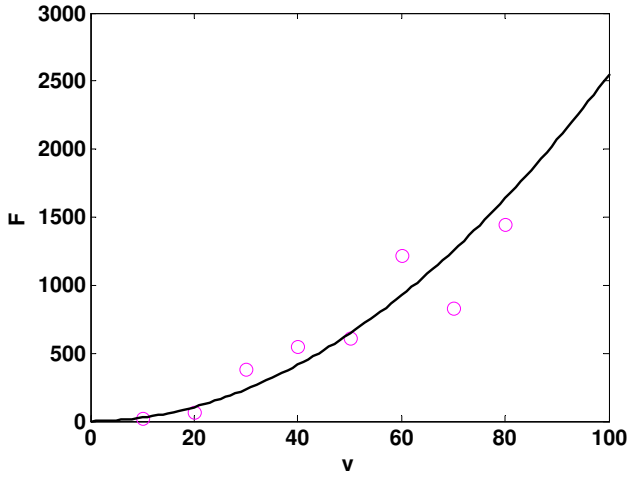
or because $\ln e^{-0.034t} = -0.034t$,

$$\ln c = \ln 4.84 - 0.034t$$

Thus, on a semi-log plot, the relationship is a straight line with an intercept of ln 4.84 and a slope of –0.034.

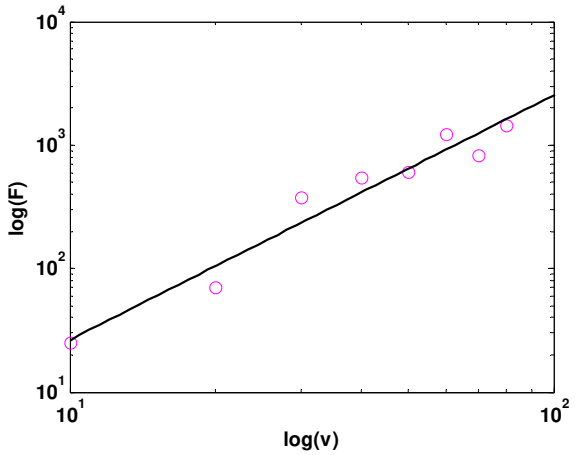
2.13

```
>> 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,'om',vf,Ff,'-.k')
>> xlabel('v');ylabel('F');
```



2.14

```
>> v = 10:10:80;
>> F = [25 70 380 550 610 1220 830 1450];
>> vf=logspace(1,2);
>> Ff = 0.2741*vf.^1.9842;
>> loglog(v,F,'om',vf,Ff,'-.k')
>> xlabel('log(v)');ylabel('log(F)');
```



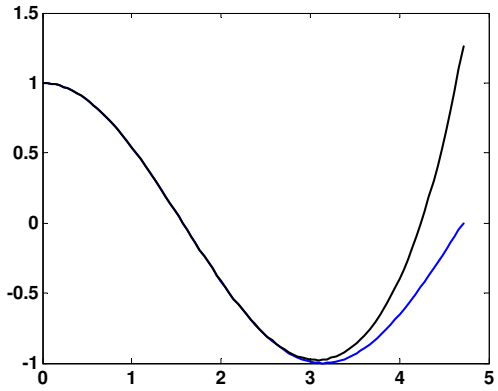
The result is a straight line. The reason for this outcome can be understood by taking the common logarithm of the function to give,

$$\log_{10} F = \log_{10} 0.2741 + 1.9842 \log_{10} v$$

Thus, on a log-log plot, the slope would be 1.9842 and the intercept would be $\log_{10}(0.2741) = -0.562$.

2.15

```
>> x = linspace(0,3*pi/2);
>> c = cos(x);
>> cf = 1-x.^2/2+x.^4/factorial(4)-x.^6/factorial(6)+x.^8/factorial(8);
>> plot(x,c,x,cf,'k--')
```



2.16 (a)

```
>> m=[83.6 60.2 72.1 91.1 92.9 65.3 80.9];
>> vt=[53.4 48.5 50.9 55.7 54 47.7 51.1];
>> g=9.81; rho=1.223;
>> A=[0.455 0.402 0.452 0.486 0.531 0.475 0.487];
>> cd=g*m./vt.^2;
>> CD=2*cd/rho./A
```

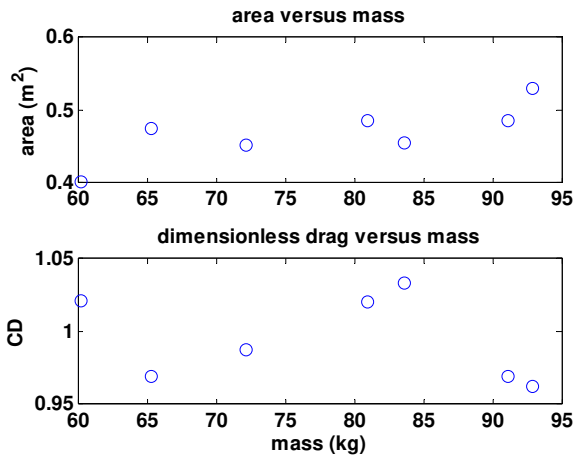
```
CD =
    1.0337    1.0213    0.9877    0.9693    0.9625    0.9693    1.0206
```

(b)

```
>> CDmin=min(CD),CDmax=max(CD),CDavg=mean(CD)
CDmin =
    0.9625
CDmax =
    1.0337
CDavg =
    0.9949
```

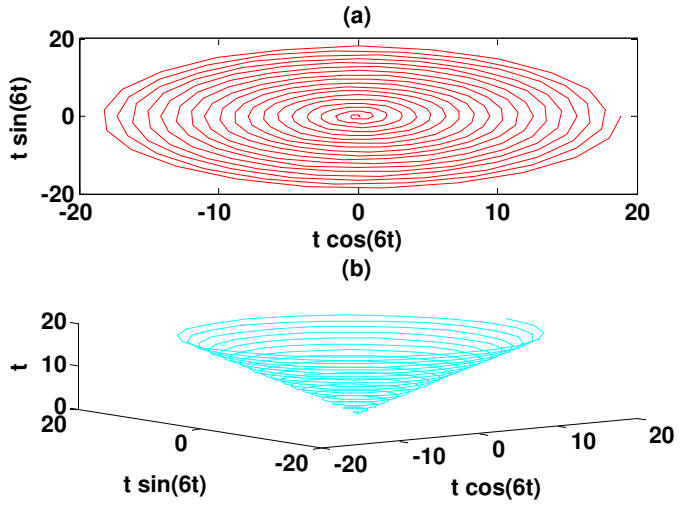
(c)

```
subplot(2,1,1);plot(m,A,'o')
ylabel('area (m^2)')
title('area versus mass')
subplot(2,1,2);plot(m,CD,'o')
xlabel('mass (kg)');ylabel('CD')
title('dimensionless drag versus mass')
```



2.17 (a)

```
t = 0:pi/64:6*pi;
subplot(2,1,1);plot(t.*cos(6*t),t.*sin(6*t),'r')
title('(a)');xlabel('t cos(6t)');ylabel('t sin(6t)')
subplot(2,1,2);plot3(t.*cos(6*t),t.*sin(6*t),t,'c')
title('(b)');xlabel('t cos(6t)');ylabel('t sin(6t)');zlabel('t')
```



2.18 (a)

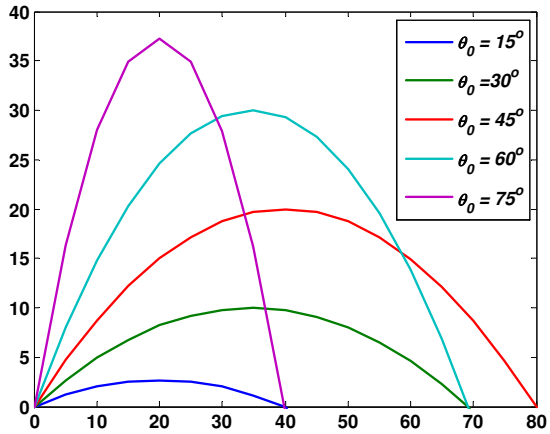
```
>> x = 5;
>> x ^ 3;
>> y = 8 - x
y =
     3
```

(b)

```
>> q = 4:2:12;
>> r = [7 8 4; 3 6 -5];
>> sum(q) * r(2,3)
q =
   -200
```

2.19

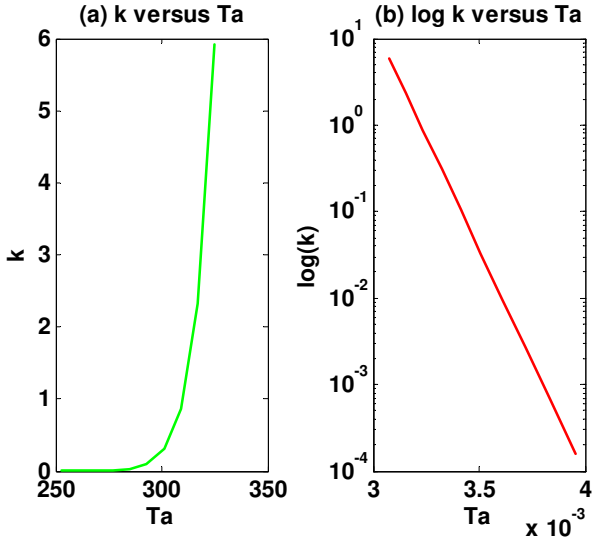
```
>> clf
>> y0=0;v0=28;g=9.81;
>> x=0:5:80;
>> theta0=15*pi/180;
>> y1=tan(theta0)*x-g/(2*v0^2*cos(theta0)^2)*x.^2+y0;
>> theta0=30*pi/180;
>> y2=tan(theta0)*x-g/(2*v0^2*cos(theta0)^2)*x.^2+y0;
>> theta0=45*pi/180;
>> y3=tan(theta0)*x-g/(2*v0^2*cos(theta0)^2)*x.^2+y0;
>> theta0=60*pi/180;
>> y4=tan(theta0)*x-g/(2*v0^2*cos(theta0)^2)*x.^2+y0;
>> theta0=75*pi/180;
>> y5=tan(theta0)*x-g/(2*v0^2*cos(theta0)^2)*x.^2+y0;
>> y=[y1' y2' y3' y4' y5'] ;
>> plot(x,y);axis([0 80 0 40])
>> legend('\it\theta_0 = 15^o','\it\theta_0 =30^o', ...
    '\it\theta_0 = 45^o','\it\theta_0 = 60^o','\it\theta_0 = 75^o')
```



2.20

```
>> clf
>> R=8.314;E=1e5;A=7E16;
>> Ta=253:8:325;
>> k=A*exp(-E./(R*Ta))
k =
  0.0002 0.00070.00270.0097 0.0328 0.1040 0.3096 0.8711 2.3265 5.9200

R=8.314;E=1e5;A=7E16;
Ta=253:8:325;
k=A*exp(-E./(R*Ta))
subplot(1,2,1);plot(Ta,k,'g')
```

```
xlabel('Ta');ylabel('k');title('(a) k versus Ta')
subplot(1,2,2);semilogy(1./Ta,k,'r')
xlabel('Ta');ylabel('log(k)');title('(b) log k versus Ta')
```



The result in **(b)** is a straight line. The reason for this outcome can be understood by taking the common logarithm of the function to give,

$$\log_{10} k = \log_{10} A - \left(\frac{E}{R}\log_{10} e\right)\frac{1}{T_a}$$

Thus, a plot of $\log_{10}k$ versus $1/T_a$ is linear with a slope of $-(E/R)\log_{10}e$ and an intercept of $\log_{10}A$.

2.21 The equations to generate the plots are

$$(a)\ y = \frac{w_0}{120EIL}\left(-x^5 + 2L^2x^3 - L^4x\right)$$

$$(b)\ \frac{dy}{dx} = \frac{w_0}{120EIL}\left(-5x^4 + 6L^2x^2 - L^4\right)$$

$$(c)\ M(x) = EI\frac{d^2y}{dx^2} = \frac{w_0}{120L}\left(-20x^3 + 12L^2x\right)$$

$$(d)\ V(x) = EI\frac{d^3y}{dx^3} = \frac{w_0}{120L}\left(-60x^2 + 12L^2\right)$$

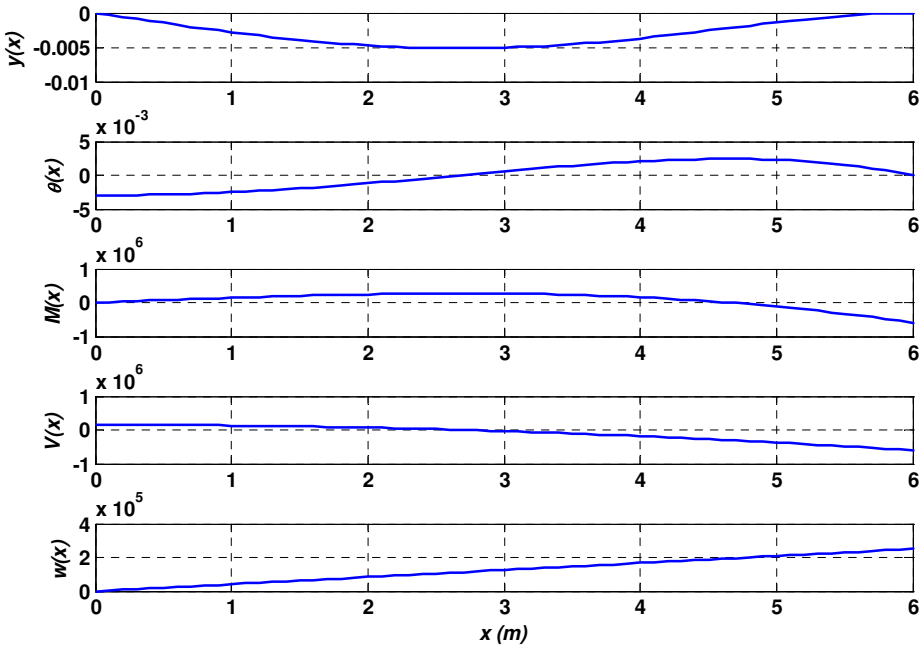
$$(e)\ w(x) = EI\frac{d^4y}{dx^4} = \frac{w_0}{L}x$$

The following MATLAB script can be developed to generate the plot:

```
format short g
E=50000*1e3*1e4;I=0.0003;w0=2.5e3*100;L=600/100;dx=10/100;
x=[0:dx:L];
clf
y=w0/(120*E*I*L)*(-x.^5+2*L^2*x.^3-L^4.*x);
theta=w0/(120*E*I*L)*(-5*x.^4+6*L^2*x.^2-L^4);
M=w0/(120*L)*(-20*x.^3+12*L^2*x);
```

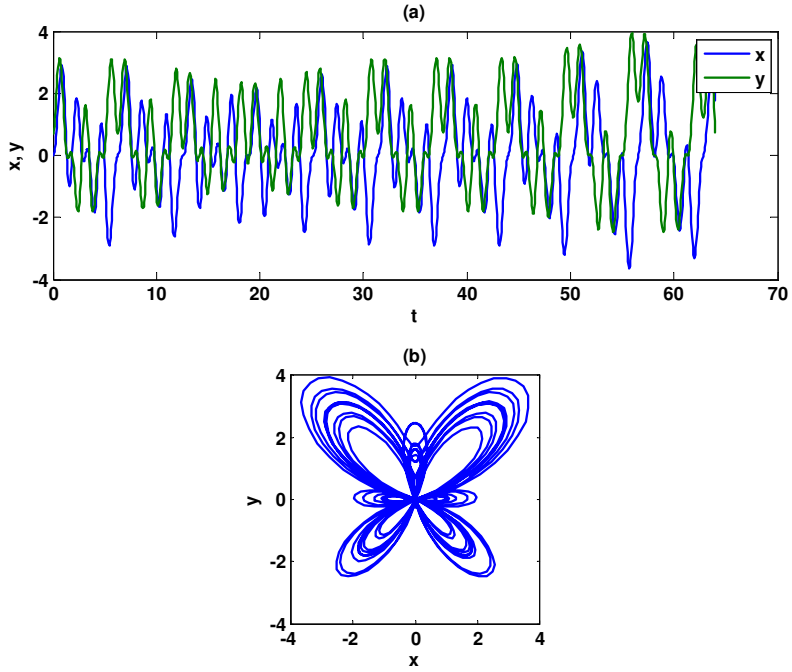
```
V=w0/(120*L)*(-60*x.^2+12*L^2);
w=w0/L*x;
subplot(5,1,1)
plot(x,y);grid;ylabel('\ity(x)')
subplot(5,1,2)
plot(x,theta);grid;ylabel('\it\theta(x)')
subplot(5,1,3)
plot(x,M);grid;ylabel('\itM(x)')
subplot(5,1,4)
plot(x,V);grid;ylabel('\itV(x)')
subplot(5,1,5)
plot(x,w);grid;ylabel('\itw(x)')
xlabel('\itx (m)')
```

The resulting plot is



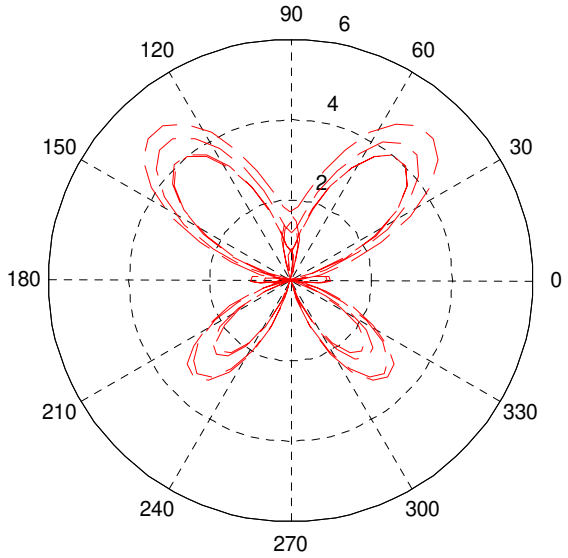
2.22

```
clf
t=[0:1/16:64];
x=sin(t).*(exp(cos(t))-2*cos(4*t)-sin(t/12).^5);
y=cos(t).*(exp(cos(t))-2*cos(4*t)-sin(t/12).^5);
subplot(2,1,1)
plot(t,x,t,y,':');title('(a)');xlabel('t');ylabel('x, y');legend('x','y')
subplot(2,1,2)
plot(x,y);axis square;title('(b)');xlabel('x');ylabel('y')
```



2.23

```
clf
t = 0:pi/32:8*pi;
polar(t,exp(sin(t))-2*cos(4*t)+sin((2*t-pi)/24).^5,'--r')
```



CHAPTER 3

3.1 The M-file can be written as

```
function Vol = tankvolume(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
  Vol = 'overtop';
end
```

This function can be used to evaluate the test cases,

```
>> tankvolume(0.9,1)
ans =
    1.0179
>> tankvolume(1.5,1.25)
ans =
    2.0453
>> tankvolume(1.3,3.8)
ans =
   15.5739
>> tankvolume(1.3,4)
ans =
overtop
```

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(100000,0.05,10)

  year   future worth
    0      100000.00
    1      105000.00
    2      110250.00
    3      115762.50
    4      121550.63
    5      127628.16
    6      134009.56
    7      140710.04
    8      147745.54
    9      155132.82
   10      162889.46
```

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   annual payment\n');
fprintf('%5d %14.2f\n',y);
```

This function can be used to evaluate the test case,

```
>> annualpayment(100000,.033,5)

  year   annual payment
    1      103300.00
    2       52488.39
    3       35557.14
    4       27095.97
    5       22022.84
```

3.4 The M-file can be written as

```
function Ta = avgtemp(Tm, Tp, ts, te)
w = 2*pi/365;
t = ts:te;
T = Tm + (Tp-Tm)*cos(w*(t-205));
Ta = mean(T);
```

This function can be used to evaluate the test cases,

```
>> avgtemp(23.1,33.6,0,59)
ans =
   13.1332
>> avgtemp(10.6,17.6,180,242)
ans =
   17.2265
```

3.5 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  %12.7f\n',i,tru,ser,er);
  i = i + 1;
end
```

This function can be used to evaluate the test case,

```
>> sincomp(0.9,8)

order  true value     approximation     error
  1    0.7833269096    0.9000000000  -14.89455921
  2    0.7833269096    0.7785000000    0.61620628
  3    0.7833269096    0.7834207500   -0.01197972
  4    0.7833269096    0.7833258498    0.00013530
  5    0.7833269096    0.7833269174   -0.00000100
```

```
  6    0.7833269096    0.7833269096    0.00000001
  7    0.7833269096    0.7833269096   -0.00000000
  8    0.7833269096    0.7833269096    0.00000000
```

3.6 The M-file can be written as

```
function [r, th] = polar(x, y)
r = sqrt(x .^ 2 + y .^ 2);
if x > 0
  th = atan(y/x);
elseif 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(2,0)
r =
     2
th =
     0
```

All the cases are summarized as

x	*y*	*r*	*θ*
2	0	2	0
2	1	2.236068	26.56505
0	3	3	90
-3	1	3.162278	161.5651
-2	0	2	180
-1	-2	2.236068	-116.565
0	0	0	0
0	-2	2	-90
2	2	2.828427	45

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
    th(i) = atan(y(i) / x(i));
  elseif 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,

```
>> x=[2 2 0 -3 -2 -1 0 0 2];
>> y=[0 1 3 1 0 -2 0 -2 2];
>> polar2(x,y)

     x        y       radius     angle
    2.00     0.00     2.0000     0.0000
    2.00     1.00     2.2361    26.5651
    0.00     3.00     3.0000    90.0000
   -3.00     1.00     3.1623   161.5651
   -2.00     0.00     2.0000   180.0000
   -1.00    -2.00     2.2361  -116.5651
    0.00     0.00     0.0000     0.0000
    0.00    -2.00     2.0000   -90.0000
    2.00     2.00     2.8284    45.0000
```

3.8 The M-file can be written as

```
function grade = lettergrade(score)
if score <0 | score>100
  error('Value must be >= 0 and <= 100')
elseif 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,

```
>> grade = lettergrade(89.9999)
grade =
```

```
B
>> grade = lettergrade(90)
grade =
A
>> grade = lettergrade(45)
grade =
F
>> grade = lettergrade(120)
??? Error using ==> lettergrade
Incorrect value entered
```

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,

```
>> A=[.036 .0001 10 2
.020 .0002 8 1
.015 .0012 20 1.5
.03 .0007 25 3
.022 .0003 15 2.6];
>> Manning(A)

    n         S          B          H          U
   0.036   0.0001      10.00       2.00     0.3523
   0.020   0.0002       8.00       1.00     0.6094
   0.015   0.0012      20.00       1.50     2.7569
   0.030   0.0007      25.00       3.00     1.5894
   0.022   0.0003      15.00       2.60     1.2207
```

3.10 The M-file can be written as

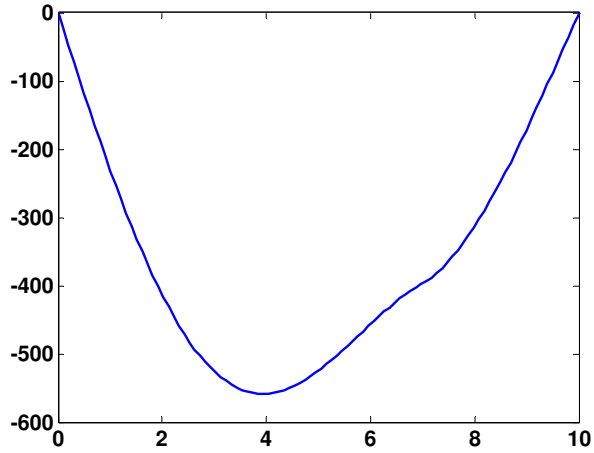
```
function beamProb(x)
xx = linspace(0,x);
n=length(xx);
for i=1:n
  uy(i) = -5/6.*(sing(xx(i),0,4)-sing(xx(i),5,4));
  uy(i) = uy(i) + 15/6.*sing(xx(i),8,3) + 75*sing(xx(i),7,2);
  uy(i) = uy(i) + 57/6.*xx(i)^3 - 238.25.*xx(i);
end
clf,plot(xx,uy,'--')
```

The M-file uses the following function

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

The plot can then be created as,

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

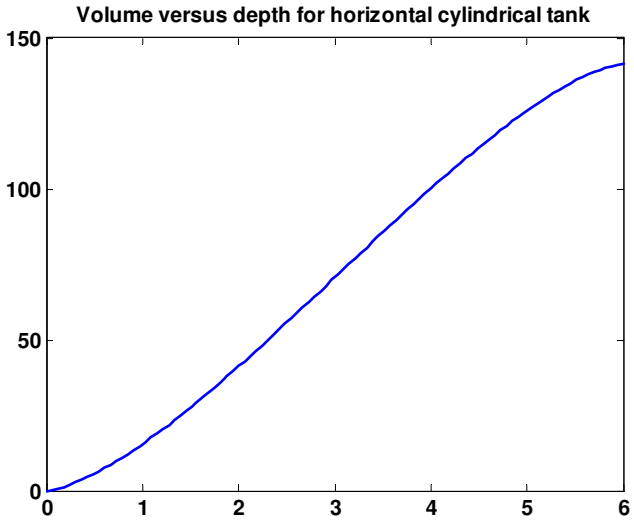


3.11 The M-file can be written as

```
function cylinder(r, L, plot_title)
% volume of horizontal cylinder
% inputs:
% r = radius
% L = length
% plot_title = string holding plot title
h = linspace(0,2*r);
V = (r^2*acos((r-h)./r)-(r-h).*sqrt(2*r*h-h.^2))*L;
clf,plot(h, V)
```

This function can be run to generate the plot,

```
>> cylinder(3,5,...
'Volume versus depth for horizontal cylindrical tank')
```



3.12 Errata for first printing: The loop should be:

```
for i=2:ni+1
  t(i)=t(i-1)+(tend-tstart)/ni;
  y(i)=12 + 6*cos(2*pi*t(i)/ ...
```

```
        (tend-tstart));
end
```

A vectorized version can be written as

```
tt=tstart:(tend-tstart)/ni:tend
yy=12+6*cos(2*pi*tt/(tend-tstart))
```

Both generate the following values for t and y:

```
t =
  0 2.5000 5.0000 7.5000 10.0000 12.5000 15.0000 17.5000 20.0000
y =
 18.0000 16.2426 12.0000 7.7574 6.0000 7.7574 12.0000 16.2426 18.0000
```

3.13

```
function s=SquareRoot(a,eps)
ind=1;
if a ~= 0
  if a < 0
    a=-a;ind=j;
  end
  x = a / 2;
  while(1)
    y = (x + a / x) / 2;
    e = abs((y - x) / y);
    x = y;
    if e < eps, break, end
  end
  s = x;
else
  s = 0;
end
s=s*ind;
```

The function can be tested:

```
>> SquareRoot(0,1e-4)
ans =
     0
>> SquareRoot(2,1e-4)
ans =
    1.4142
>> SquareRoot(10,1e-4)
ans =
    3.1623
>> SquareRoot(-4,1e-4)
ans =
        0 + 2.0000i
```

3.14 Errata: On first printing, change function for $8 \le t < 16$ to `v = 624 - 3*t;`

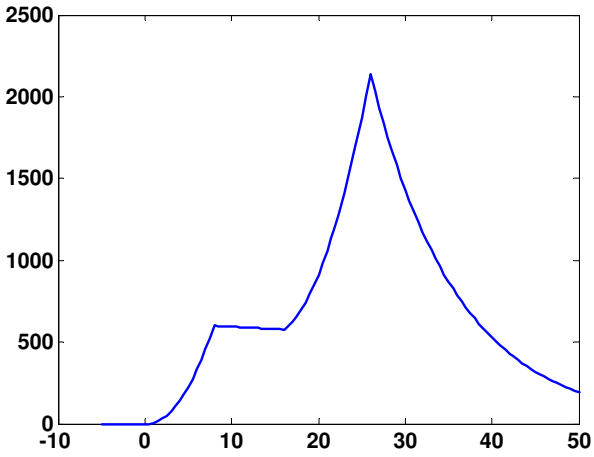
The following function implements the piecewise function:

```
function v = vpiece(t)
if t<0
  v = 0;
elseif t<8
  v = 10*t^2 - 5*t;
```

```
elseif t<16
  v = 624 - 3*t;
elseif t<26
  v = 36*t + 12*(t - 16)^2;
else
  v = 2136*exp(-0.1*(t-26));
end
```

Here is a script that uses `vpiece` to generate the plot

```
k=0;
for i = -5:.5:50
  k=k+1;
  t(k)=i;
  v(k)=vpiece(t(k));
end
plot(t,v)
```



3.15 This following function employs the round to larger method. For this approach, if the number is exactly halfway between two possible rounded values, it is always rounded to the larger number.

```
function xr=rounder(x, n)
if n < 0,error('negative number of integers illegal'),end
xr=round(x*10^n)/10^n;
```

Here are the test cases:

```
>> rounder(477.9587,2)
ans =
  477.9600
>> rounder(-477.9587,2)
ans =
 -477.9600
>> rounder(0.125,2)
ans =
    0.1300
>> rounder(0.135,2)
ans =
    0.1400
>> rounder(-0.125,2)
ans =
   -0.1300
```

```
>> rounder(-0.135,2)
ans =
   -0.1400
```

A preferable approach is called *banker's rounding* or *round to even*. In this method, a number exactly midway between two possible rounded values returns the value whose rightmost significant digit is even. Here is as function that implements banker's rounding along with the test cases that illustrate how it differs from `rounder`:

```
function xr=rounderbank(x, n)
if n < 0,error('negative number of integers illegal'),end
x=x*10^n;
if mod(floor(abs(x)),2)==0 & abs(x-floor(x))==0.5
  xr=round(x/2)*2;
else
  xr=round(x);
end
xr=xr/10^n;

>> rounder(477.9587,2)
ans =
  477.9600
>> rounder(-477.9587,2)
ans =
 -477.9600
>> rounderbank(0.125,2)
ans =
    0.1200
>> rounderbank(0.135,2)
ans =
    0.1400
>> rounderbank(-0.125,2)
ans =
   -0.1200
>> rounderbank(-0.135,2)
ans =
   -0.1400
```

3.16

```
function nd = days(mo, da, leap)
nd = 0;
for m=1:mo-1
  switch m
    case {1, 3, 5, 7, 8, 10, 12}
      nday = 31;
    case {4, 6, 9, 11}
      nday = 30;
    case 2
      nday = 28+leap;
  end
  nd=nd+nday;
end
nd = nd + da;

>> days(1,1,0)
ans =
     1
>> days(2,29,1)
```

```
ans =
    60
>> days(3,1,0)
ans =
    60
>> days(6,21,0)
ans =
   173
>> days(12,31,1)
ans =
   366
```

3.17

```
function nd = days(mo, da, year)
leap = 0;
if year / 4 - fix(year / 4) == 0, leap = 1; end
nd = 0;
for m=1:mo-1
  switch m
    case {1, 3, 5, 7, 8, 10, 12}
      nday = 31;
    case {4, 6, 9, 11}
      nday = 30;
    case 2
      nday = 28+leap;
  end
  nd=nd+nday;
end
nd = nd + da;
```

```
>> days(1,1,1997)
ans =
     1
>> days(2,29,2004)
ans =
    60
>> days(3,1,2001)
ans =
    60
>> days(6,21,2004)
ans =
   173
>> days(12,31,2008)
ans =
   366
```

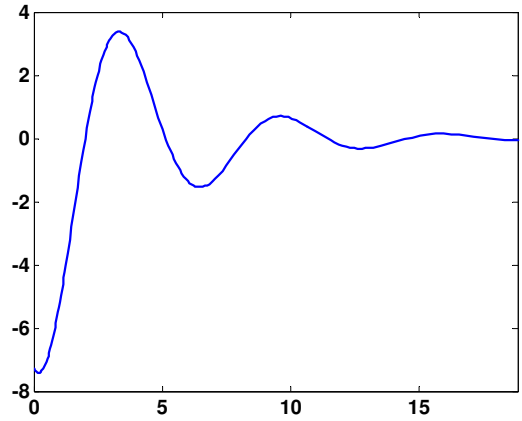
3.18

```
function fr = funcrange(f,a,b,n,varargin)
% funcrange: function range and plot
%   fr=funcrange(f,a,b,n,varargin): computes difference
%      between maximum and minimum value of function over
%      a range. In addition, generates a plot of the function.
% input:
%   f = function to be evaluated
%   a = lower bound of range
```

```
%    b = upper bound of range
%    n = number of intervals
% output:
%    fr = maximum - minimum
x = linspace(a,b,n);
y = f(x,varargin{:});
fr = max(y)-min(y);
fplot(f,[a b],varargin{:})
end
```

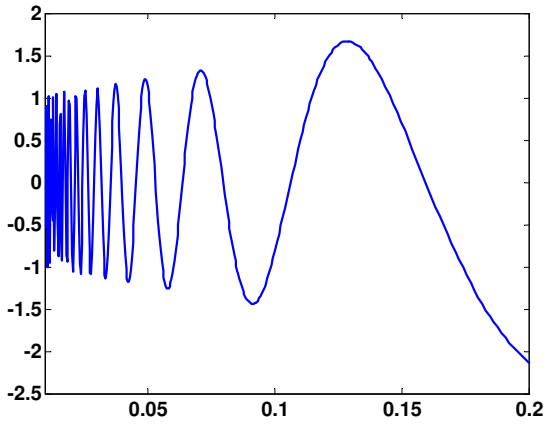
(a)

```
>> f=@(t) 8*exp(-0.25*t).*sin(t-2);
>> funcrange(f,0,6*pi,1000)
ans =
    10.7910
```



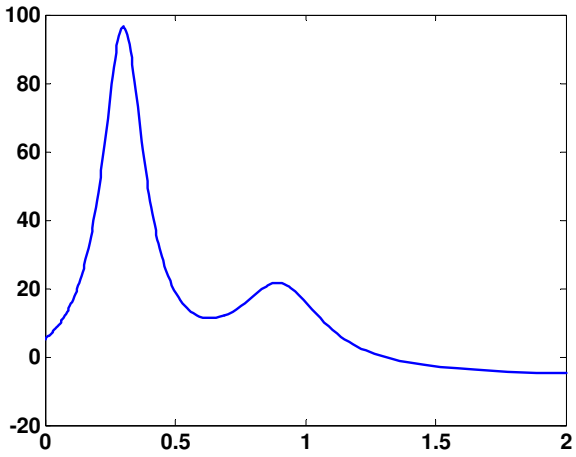
(b)

```
>> f=@(x) exp(4*x).*sin(1./x);
>> funcrange(f,0.01,0.2,1000)
ans =
     3.8018
```



(c)

```
>> funcrange(@humps,0,2,1000)
ans =
  101.3565
```



3.19

```
function yend = odesimp(dydt, dt, ti, tf, yi, varargin)
% odesimp: Euler ode solver
% yend = odesimp(dydt, dt, ti, tf, yi, varargin):
%   Euler's method solution of a single ode
% input:
%   dydt = function defining ode
%   dt = time step
%   ti = initial time
%   tf = final time
%   yi = initial value of dependent variable
% output:
%   yend = dependent variable at final time
t = ti; y = yi; h = dt;
while (1)
  if t + dt > tf, h = tf - t; end
  y = y + dydt(y,varargin{:}) * h;
  t = t + h;
  if t >= tf, break, end
end
yend = y;
```

test run:

```
>> dvdt=@(v,m,cd) 9.81-(cd/m)*v^2;
>> odesimp(dvdt,0.5,0,12,-10,70,0.23)

ans =
   51.1932
```

3.20 Here is a function to solve this problem:

```
function [theta,c,mag]=vector(a,b)
amag=norm(a);
bmag=norm(b);
adotb=dot(a,b);
theta=acos(adotb/amag/bmag)*180/pi;
c=cross(a,b);
mag=norm(c);
x1=[0 a(1)];y1=[0 a(2)];z1=[0 a(3)];
x2=[0 b(1)];y2=[0 b(2)];z2=[0 b(3)];
x3=[0 c(1)];y3=[0 c(2)];z3=[0 c(3)];
```

```
x4=[0 0];y4=[0 0];z4=[0 0];
plot3(x1,y1,z1,'--b',x2,y2,z2,'--r',x3,y3,z3,'-k',x4,y4,z4,'o')
xlabel='x';ylabel='y';zlabel='z';
```

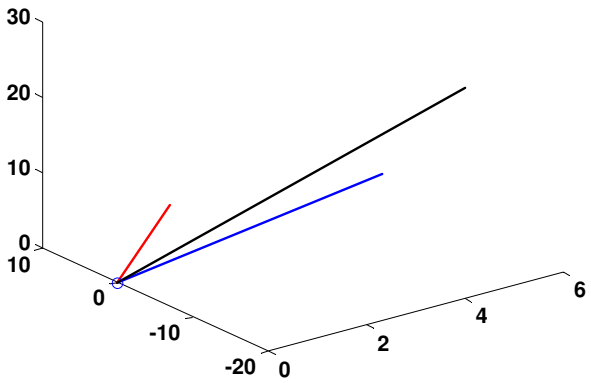
Here is a script to run the three cases

```
a = [6 4 2]; b = [2 6 4];
[th,c,m]=vector(a,b)
pause
a = [3 2 -6]; b = [4 -3 1];
[th,c,m]=vector(a,b)
pause
a = [2 -2 1]; b = [4 2 -4];
[th,c,m]=vector(a,b)
pause
a = [-1 0 0]; b = [0 -1 0];
[th,c,m]=vector(a,b)
```

When this is run, the following output is generated

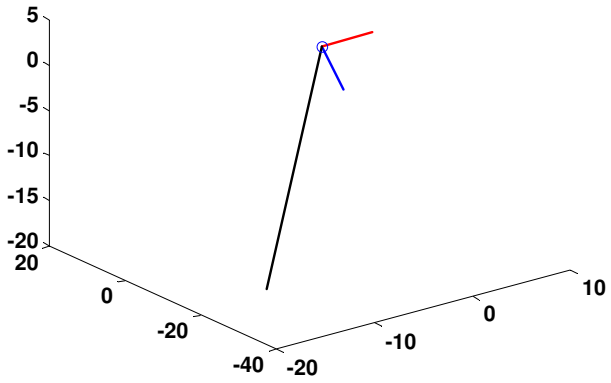
(a)

```
th =
    38.2132
c =
     4   -20    28
m =
    34.6410
```

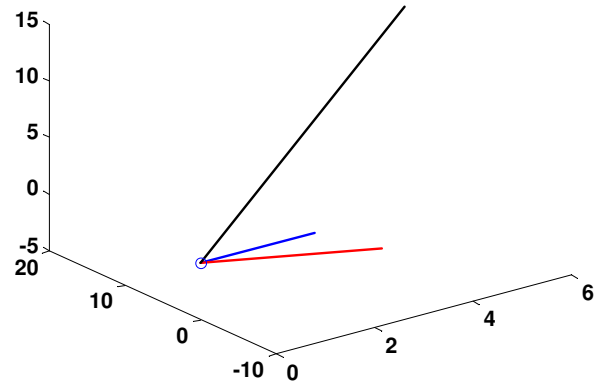


(b)

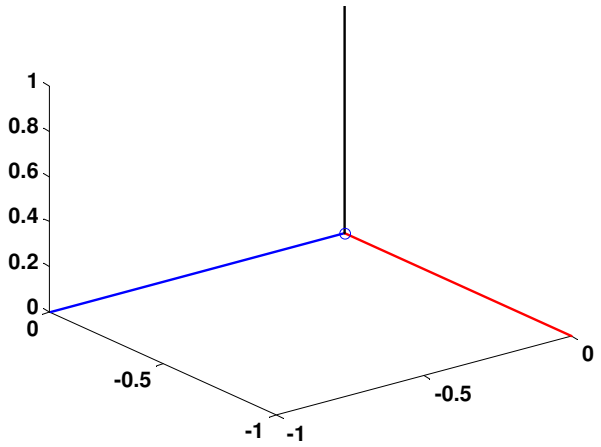
```
th =
     90
c =
   -16   -27   -17
m =
    35.6931
```




```
(c)
th =
    90
c =
     6    12    12
m =
    18
```



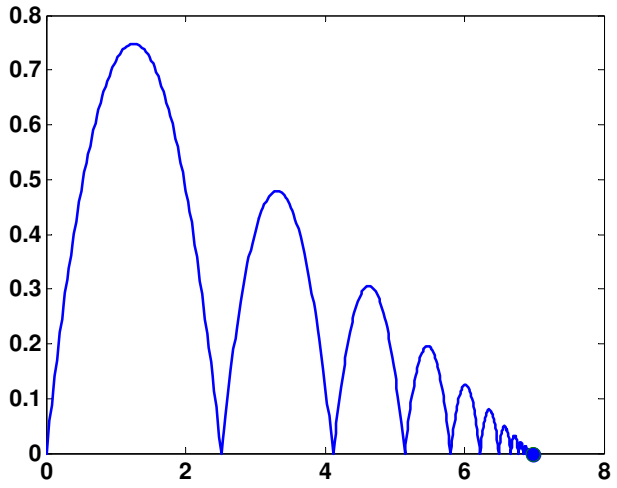

```
(d)
th =
    90
c =
     0     0     1
m =
     1
```



3.21 The script for this problem can be written as

```
clc,clf,clear
maxit=1000;
g=9.81; theta0=50*pi/180; v0=5; CR=0.83;
j=1;t(j)=0;x=0;y=0;
xx=x;yy=y;
plot(x,y,'o','MarkerFaceColor','b','MarkerSize',8)
xmax=8; axis([0 xmax 0 0.8])
M(1)=getframe;
dt=1/128;
j=1; xxx=0; iter=0;
while(1)
  tt=0;
  timpact=2*v0*sin(theta0)/g;
  ximpact=v0*cos(theta0)*timpact;
  while(1)
    j=j+1;
    h=dt;
    if tt+h>timpact,h=timpact-tt;end
    t(j)=t(j-1)+h;
    tt=tt+h;
    x=xxx+v0*cos(theta0)*tt;
    y=v0*sin(theta0)*tt-0.5*g*tt^2;
    xx=[xx x];yy=[yy y];
    plot(xx,yy,':',x,y,'o','MarkerFaceColor','b','MarkerSize',8)
    axis([0 xmax 0 0.8])
    M(j)=getframe;
    iter=iter+1;
    if tt>=timpact, break, end
  end
  v0=CR*v0;
  xxx=x;
  if x>=xmax|iter>=maxit,break,end
end
pause
clf
axis([0 xmax 0 0.8])
movie(M,1,36)
```

Here's the plot that will be generated:

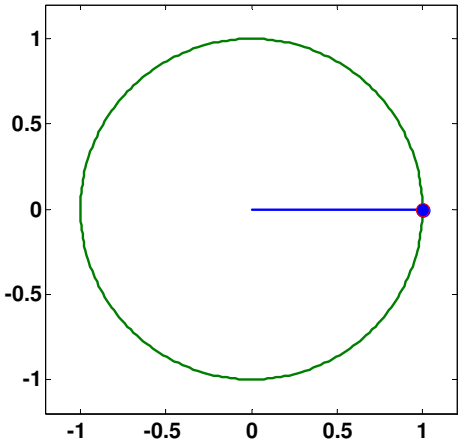


3.22 The function for this problem can be written as

```
function phasor(r, nt, nm)
% function to show the orbit of a phasor
% r = radius
% nt = number of increments for theta
% nm = number of movies
clc;clf
dtheta=2*pi/nt;
th=0;
fac=1.2;
xx=r;yy=0;
for i=1:nt+1
  x=r*cos(th);y=r*sin(th);
  xx=[xx x];yy=[yy y];
  plot([0 x],[0 y],xx,yy,':',...
        x,y,'o','MarkerFaceColor','b','MarkerSize',8)
  axis([-fac*r fac*r -fac*r fac*r]);
  axis square
  M(i)=getframe;
  th=th+dtheta;
end
pause
clf
axis([-fac*r fac*r -fac*r fac*r]);
axis square
movie(M,1,36)
```

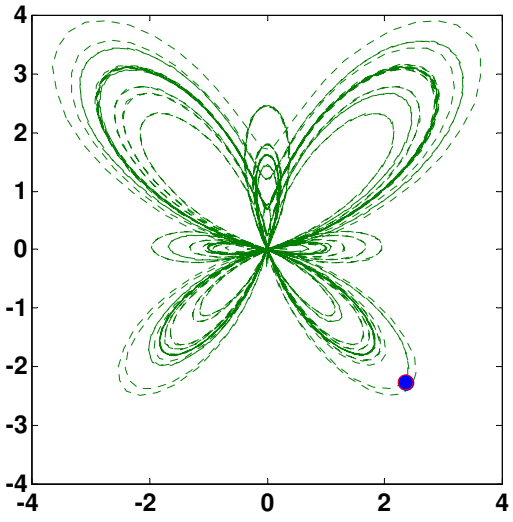
When it is run, the result is

```
>> phasor(1, 256, 10)
```



3.23 A script to solve this problem based on the parametric equations described in Prob. 2.22:

```
clc;clf
t=[0:1/16:128];
x(1)=sin(t(1)).*(exp(cos(t(1)))-2*cos(4*t(1))-sin(t(1)/12).^5);
y(1)=cos(t(1)).*(exp(cos(t(1)))-2*cos(4*t(1))-sin(t(1)/12).^5);
xx=x;yy=y;
plot(x,y,xx,yy,':',x,y,'o','MarkerFaceColor','b','MarkerSize',8)
axis([-4 4 -4 4]); axis square
M(1)=getframe;
for i = 2:length(t)
  x=sin(t(i)).*(exp(cos(t(i)))-2*cos(4*t(i))-sin(t(i)/12).^5);
  y=cos(t(i)).*(exp(cos(t(i)))-2*cos(4*t(i))-sin(t(i)/12).^5);
  xx=[xx x];yy=[yy y];
  plot(x,y,xx,yy,':',x,y,'o','MarkerFaceColor','b','MarkerSize',8)
  axis([-4 4 -4 4]); axis square
  M(i)=getframe;
end
```



CHAPTER 4

4.1 The function can be developed as

```
function [fx,ea,iter] = SquareRoot(a,es,maxit)
% Divide and average method for evaluating square roots
%   [fx,ea,iter] = SquareRoot(a,es,maxit)
% input:
%   a = value for which square root is to be computed
%   es = stopping criterion (default = 0.0001)
%   maxit = maximum iterations (default = 50)
% output:
%   fx = estimated value
%   ea = approximate relative error (%)
%   iter = number of iterations

% defaults:
if nargin<2|isempty(es),es=0.0001;end
if nargin<3|isempty(maxit),maxit=50;end
if a<= 0,error('value must be positive'),end
% initialization
iter = 1; sol = a/2; ea = 100;
% iterative calculation
while (1)
  solold = sol;
  sol = (sol + a/sol)/2;
  iter = iter + 1;
  if sol~=0
    ea=abs((sol - solold)/sol)*100;
  end
  if ea<=es | iter>=maxit,break,end
end
fx = sol;
end
```

It can be tested for the following cases:

```
>> format long
>> [fx,ea,iter] = SquareRoot(25)

fx =
     5
ea =
    3.355538069627073e-010
iter =
     7

>> [fx,ea,iter] = SquareRoot(25,0.001)

fx =
   5.000000000016778
ea =
    2.590606381316551e-004
iter =
     6

>> [fx,ea,iter] = SquareRoot(0)

??? Error using ==> SquareRoot
value must be positive
```

4.2 (a)

$$(1011001)_2 = (1\times 2^6)+(0\times 2^5)+(1\times 2^4)+(1\times 2^3)+(0\times 2^2)+(0\times 2^1)+(1\times 2^0)$$
$$= 1(64)+0(32)+1(16)+1(8)+0(4)+0(2)+1(1) = 89$$

(b)

$$(0.01011)_2 = (0\times 2^{-1})+(1\times 2^{-2})+(0\times 2^{-3})+(1\times 2^{-4})+(1\times 2^{-5})$$
$$= 0(0.5)+1(0.25)+0(0.125)+1(0.0625)+1(0.03125)$$
$$= 0+0.25+0+0.0625+0.03125 = 0.34375$$

(c)

$$(110.01001)_2 = (1\times 2^2)+(1\times 2^1)+(0\times 2^0)+(0\times 2^{-1})+(1\times 2^{-2})+(0\times 2^{-3})+(0\times 2^{-4})+(1\times 2^{-5})$$
$$= 1(4)+1(2)+0(1)+0(0.5)+1(0.25)+0(0.125)+0(0.0625)+1(0.03125)$$
$$= 4+2+0+0+0.25+0+0+0.03125 = 6.28125$$

4.3

$$(61565)_8 = (6\times 8^4)+(1\times 8^3)+(5\times 8^2)+(6\times 8^1)+(5\times 8^0)$$
$$= 6(4096)+1(512)+5(64)+6(8)+5(1)$$
$$= 24576+512+320+48+5 = 25,461$$

$$(2.71)_8 = (2\times 8^0)+(7\times 8^{-1})+(1\times 8^{-2}) = 2(1)+7(0.125)+1(0.015625) = 2.890625$$

4.4

```
function ep = macheps
% determines the machine epsilon
e = 1;
while (1)
  if e+1<=1, break, end
  e = e/2;
end
ep = 2*e;

>> macheps
ans =
  2.2204e-016

>> eps
ans =
  2.2204e-016
```

4.5

```
function s = small
% determines the smallest number
sm = 1;
while (1)
  s=sm/2;
  if s==0,break,end
  sm = s;
end
s = sm;
```

This function can be run to give

```
>> s=small
s =
  4.9407e-324
```

This result differs from the one obtained with the built-in `realmin` function,

```
>> s=realmin
s =
  2.2251e-308
```

Challenge question: We can take the base-2 logarithm of both results,

```
>> log2(small)
ans =
       -1074
>> log2(realmin)
ans =
       -1022
```

Thus, the result of our function is 2^{-1074} ($\cong 4.9407\times10^{-324}$), whereas `realmin` gives 2^{-1022} ($\cong 2.2251\times10^{-308}$). Therefore, the function actually gives a smaller value that is equivalent to

`small` = $2^{-52}\times$`realmin`

Recall that machine epsilon is equal to 2^{-52}. Therefore,

`small` = `eps`×`realmin`

Such numbers, which are called *denormal* or *subnormal*, arise because the math coprocessor employs a different strategy for representing the significand and the exponent.

4.6 Because the exponent ranges from –126 to 127, the smallest positive number can be represented in binary as

$$\text{smallest value} = +1.0000\ldots0000\cdot 2^{-126}$$

where the 23 bits in the mantissa are all 0. This value can be translated into a base-10 value of $2^{-126} = 1.1755\times10^{-38}$. The largest positive number can be represented in binary as

$$\text{largest value} = +1.1111\ldots1111\cdot 2^{+127}$$

where the 23 bits in the mantissa are all 1. Since the significand is approximately 2 (it is actually $2 - 2^{-23}$), the largest value is therefore $2^{+128} = 3.4028\times10^{38}$. The machine epsilon in single precision would be $2^{-23} = 1.1921\times10^{-7}$.

These results can be verified using built-in MATLAB functions,

```
>> realmax('single')
ans =
   3.4028e+038

>> realmin('single')
ans =
   1.1755e-038

>> eps('single')
ans =
   1.1921e-007
```

4.7 For computers that use truncation, the machine epsilon is the positive distance from $|x|$ to the next larger in magnitude floating point number of the same precision as *x*.

$$1.1\times10^{0} - 1.0\times10^{0} = 1.0\times10^{-1}$$

For computers that hold intermediate results in a larger-sized word before returning a rounded result, the machine epsilon would be 0.05.

4.8 The true value can be computed as

$$f'(1.22) = \frac{6(0.577)}{(1-3\times0.577^2)^2} = 2,352,911$$

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.

4.9 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.

4.10 (*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:

$$f(1) = 0.778801 - 0.778801(0.75) + 0.778801\frac{0.75^2}{2} - 0.778801\frac{0.75^3}{6} = 0.358978$$
$$\varepsilon_t = \left|\frac{0.367879 - 0.358978}{0.367879}\right|100\% = 2.42\%$$

4.11 Use $\varepsilon_s = 0.5\times10^{2-2} = 0.5\%$. 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:

$$\cos\left(\frac{\pi}{4}\right) \cong 1 - \frac{(\pi/4)^2}{2} = 0.691575$$
$$\varepsilon_t = \left|\frac{0.707107 - 0.691575}{0.707107}\right|100\% = 2.19\% \qquad \varepsilon_a = \left|\frac{0.691575 - 1}{0.691575}\right|100\% = 44.6\%$$

second-order:

$$\cos\left(\frac{\pi}{4}\right) \cong 0.691575 + \frac{(\pi/4)^4}{24} = 0.707429$$
$$\varepsilon_t = \left|\frac{0.707107 - 0.707429}{0.707107}\right|100\% = 0.456\% \qquad \varepsilon_a = \left|\frac{0.707429 - 0.691575}{0.707429}\right|100\% = 2.24\%$$

third-order:

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

$$\varepsilon_t = \left|\frac{0.707107 - 0.707103}{0.707107}\right| 100\% = 0.0005\% \qquad \varepsilon_a = \left|\frac{0.707103 - 0.707429}{0.707103}\right| 100\% = 0.046\%$$

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

4.12 Use $\varepsilon_s = 0.5\times 10^{2-2} = 0.5\%$. 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:

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

$$\varepsilon_t = \left|\frac{0.707107 - 0.704653}{0.707107}\right| 100\% = 0.347\% \qquad \varepsilon_a = \left|\frac{0.704653 - 0.785398}{0.704653}\right| 100\% = 11.46\%$$

second-order:

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

$$\varepsilon_t = \left|\frac{0.707107 - 0.707143}{0.707107}\right| 100\% = 0.0051\% \qquad \varepsilon_a = \left|\frac{0.707143 - 0.704653}{0.707143}\right| 100\% = 0.352\%$$

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

4.13 The true value is $f(3) = 554$.

zero order:

$$f(3) \cong f(1) = -62 \qquad \varepsilon_t = \left|\frac{554 - (-62)}{554}\right| 100\% = 111.19\%$$

first order:

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

$$f(3) \cong -62 + 70(2) = 78 \qquad \varepsilon_t = \left|\frac{554 - 78}{554}\right| 100\% = 85.92\%$$

second order:

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

$$f(3) \cong 78 + \frac{138}{2}(2)^2 = 354 \qquad \varepsilon_t = \left|\frac{554 - 354}{554}\right| 100\% = 36.10\%$$

third order:

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

$$f(3) = 354 + \frac{150}{6}(2)^3 = 554 \qquad \varepsilon_t = \left|\frac{554 - 554}{554}\right| 100\% = 0.0\%$$

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.

4.14

$$f(x) = ax^2 + bx + c$$
$$f'(x) = 2ax + b$$
$$f''(x) = 2a$$

Substitute these relationships into Eq. (4.11),

$$ax_{i+1}^2 + bx_{i+1} + c = ax_i^2 + bx_i + c + (2ax_i + b)(x_{i+1} - x_i) + \frac{2a}{2!}(x_{i+1}^2 - 2x_{i+1}x_i + x_i^2)$$

Collect terms

$$ax_{i+1}^2 + bx_{i+1} + c = ax_i^2 + 2ax_i(x_{i+1} - x_i) + a(x_{i+1}^2 - 2x_{i+1}x_i + x_i^2) + bx_i + b(x_{i+1} - x_i) + c$$
$$ax_{i+1}^2 + bx_{i+1} + c = ax_i^2 + 2ax_ix_{i+1} - 2ax_i^2 + ax_{i+1}^2 - 2ax_{i+1}x_i + ax_i^2 + bx_i + bx_{i+1} - bx_i + c$$
$$ax_{i+1}^2 + bx_{i+1} + c = (ax_i^2 - 2ax_i^2 + ax_i^2) + ax_{i+1}^2 + (2ax_ix_{i+1} - 2ax_{i+1}x_i) + (bx_i - bx_i) + bx_{i+1} + c$$
$$ax_{i+1}^2 + bx_{i+1} + c = ax_{i+1}^2 + bx_{i+1} + c$$

4.15 The true value is ln(2) = 0.693147...

zero order:

$$f(2) \cong f(1) = 0 \qquad \varepsilon_t = \left|\frac{0.693147 - 0}{0.693147}\right| 100\% = 100\%$$

first order:

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

$$f(2) \cong 0 + 1(1) = 1 \qquad \varepsilon_t = \left|\frac{0.693147 - 1}{0.693147}\right| 100\% = 44.27\%$$

second order:

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

$$f(2) = 1 - 1\frac{1^2}{2} = 0.5 \qquad \varepsilon_t = \left|\frac{0.693147 - 0.5}{0.693147}\right| 100\% = 27.87\%$$

third order:

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

$$f(2) \cong 0.5 + 2\frac{1^3}{6} = 0.833333 \qquad \varepsilon_t = \left|\frac{0.693147 - 0.833333}{0.693147}\right| 100\% = 20.22\%$$

fourth order:

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

$$f(2) \cong 0.833333 - 6\frac{1^4}{24} = 0.583333 \qquad \varepsilon_t = \left|\frac{0.693147 - 0.583333}{0.693147}\right| 100\% = 15.84\%$$

The series is converging, but at a slow rate. A smaller step size is required to obtain more rapid convergence.

4.16 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.

4.16 True value:

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

$\underline{h = 0.25}$:

$$f''(2) = \frac{f(2.25) - 2f(2) + f(1.75)}{0.25^2} = \frac{182.1406 - 2(102) + 39.85938}{0.25^2} = 288$$

$\underline{h = 0.125}$:

$$f''(2) = \frac{f(2.125) - 2f(2) + f(1.875)}{0.125^2} = \frac{139.6738 - 2(102) + 68.82617}{0.125^2} = 288$$

Both results are exact because the errors are a function of 4th and higher derivatives which are zero for a 3rd-order polynomial.

4.17 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.

4.18 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:

$$\frac{1}{1-0.1} \cong 1 + 0.1 = 1.1$$
$$\varepsilon_t = \left|\frac{1.11111 - 1.1}{1.11111}\right| 100\% = 1\% \qquad \varepsilon_a = \left|\frac{1.1 - 1}{1.1}\right| 100\% = 9.0909\%$$

second-order:

$$\frac{1}{1-0.1} \cong 1 + 0.1 + 0.01 = 1.11$$
$$\varepsilon_t = \left|\frac{1.11111 - 1.11}{1.11111}\right| 100\% = 0.1\% \qquad \varepsilon_a = \left|\frac{1.11 - 1.1}{1.11}\right| 100\% = 0.9009\%$$

third-order:

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

$$\varepsilon_t = \left|\frac{1.11111-1.111}{1.11111}\right|100\% = 0.01\% \qquad \varepsilon_a = \left|\frac{1.111-1.11}{1.111}\right|100\% = 0.090009\%$$

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

4.19 Here are the function and its derivatives

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

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

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

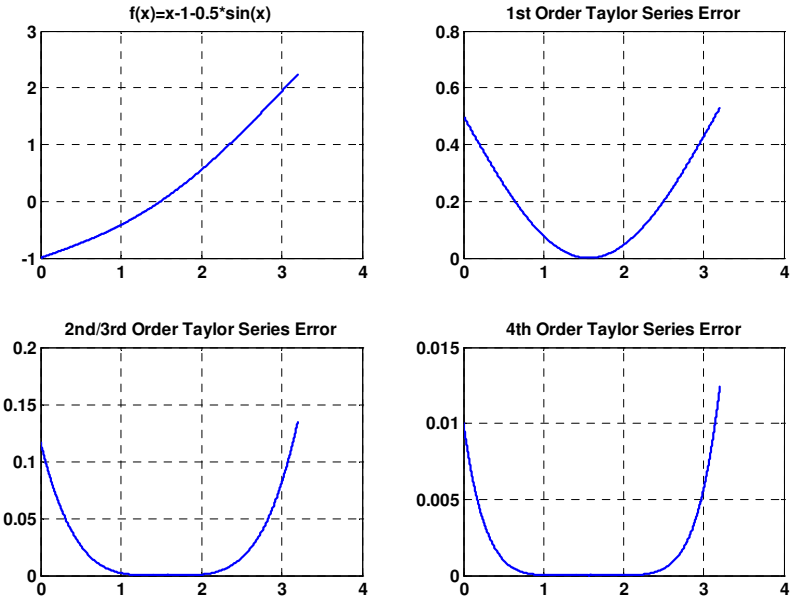
$$f^{(3)}(x) = \frac{1}{2}\cos x$$

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

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.

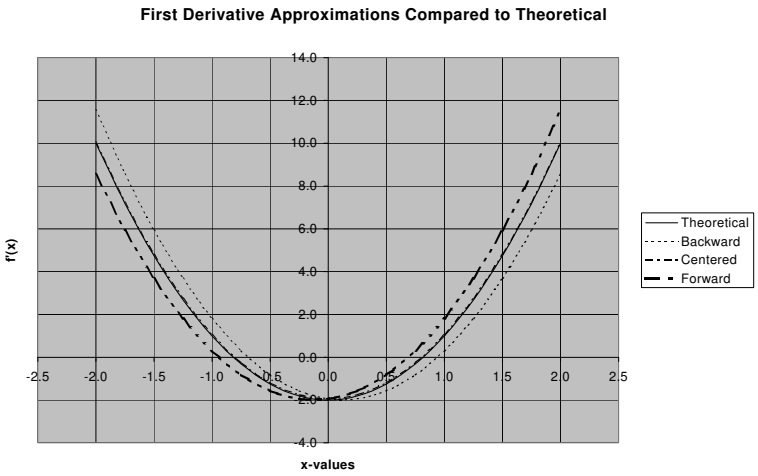
From the plots below, we see that the answer is the 4th Order Taylor Series expansion.

```
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
```

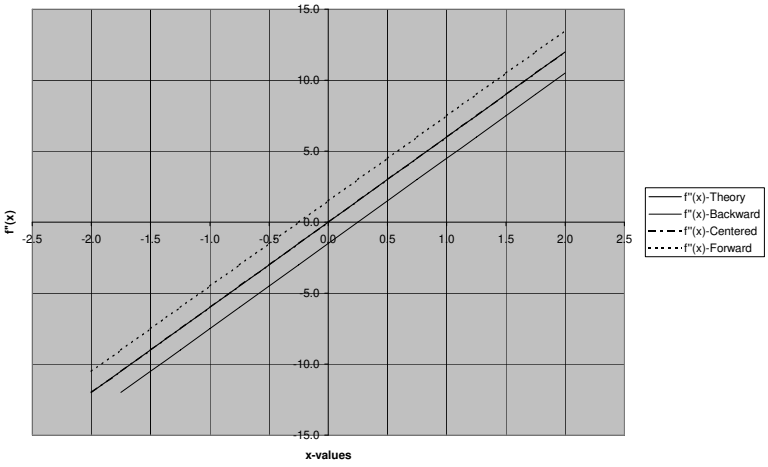


4.20

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	-6.625	3.625	-12.000	-13.500	-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



4.21 We want to find the value of h that results in an optimum of

$$E = \frac{\varepsilon}{h} + \frac{h^2 M}{6}$$

Differentiating gives

$$\frac{dE}{dh} = -\frac{\varepsilon}{h^2} + \frac{M}{3}h$$

This result can be set to zero and solved for

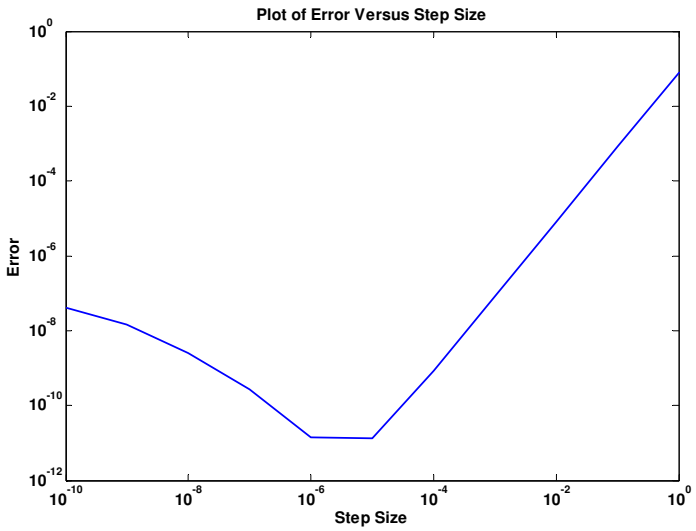
$$h^3 = \frac{3\varepsilon}{M}$$

Taking the cube root gives

$$h_{opt} = \sqrt[3]{\frac{3\varepsilon}{M}}$$

4.22 Using the same function as in Example 4.5:

```
>> ff=@(x) cos(x);
>> df=@(x) -sin(x);
>> diffex(ff,df,pi/6,11)

    step size  finite difference     true error
  1.0000000000 -0.42073549240395  0.0792645075961
  0.1000000000 -0.49916708323414  0.0008329167659
  0.0100000000 -0.49999166670833  0.0000083332917
  0.0010000000 -0.49999991666672  0.0000000833333
  0.0001000000 -0.49999999916672  0.0000000008333
  0.0000100000 -0.49999999998662  0.0000000000134
  0.0000010000 -0.50000000001438  0.0000000000144
  0.0000001000 -0.49999999973682  0.0000000002632
  0.0000000100 -0.50000000251238  0.0000000025124
  0.0000000010 -0.49999998585903  0.0000000141410
  0.0000000001 -0.50000004137019  0.0000000413702
```

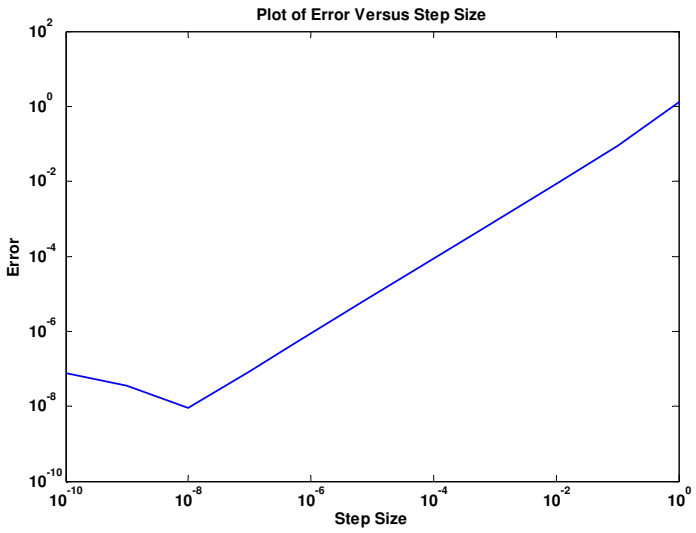


4.23 First, we must develop a function like the one in Example 4.5, but to evaluate a forward difference:

```
function prob0423(func,dfunc,x,n)
format long
dftrue=dfunc(x);
h=1;
H(1)=h;
D(1)=(func(x+h)-func(x))/h;
E(1)=abs(dftrue-D(1));
for i=2:n
  h=h/10;
  H(i)=h;
  D(i)=(func(x+h)-func(x))/h;
  E(i)=abs(dftrue-D(i));
end
L=[H' D' E'];
fprintf('   step size   finite difference    true error\n');
fprintf('%14.10f %16.14f %16.13f\n',L);
loglog(H,E),xlabel('Step Size'),ylabel('Error')
title('Plot of Error Versus Step Size')
format short
```

We can then use it to evaluate the same case as in Example 4.5:

```
>> ff=@(x) -0.1*x^4-0.15*x^3-0.5*x^2-0.25*x+1.2;
>> df=@(x) -0.4*x^3-0.45*x^2-x-0.25;
>> prob0423(ff,df,0.5,11)
   step size   finite difference    true error
  1.0000000000 -2.23750000000000  1.3250000000000
  0.1000000000 -1.00360000000000  0.0911000000000
  0.0100000000 -0.92128509999999  0.0087851000000
  0.0010000000 -0.91337535009994  0.0008753500999
  0.0001000000 -0.91258750349987  0.0000875034999
  0.0000100000 -0.91250875002835  0.0000087500284
  0.0000010000 -0.91250087497219  0.0000008749722
  0.0000001000 -0.91250008660282  0.0000000866028
  0.0000000100 -0.91250000888721  0.0000000088872
  0.0000000010 -0.91249996447829  0.0000000355217
  0.0000000001 -0.91250007550059  0.0000000755006
```



4.24 First we can evaluate the exact values using the standard formula with double-precision arithmetic as

$$\begin{matrix} x_1 \\ x_2 \end{matrix} = \frac{5{,}000.002 \pm \sqrt{(5{,}000.002)^2 - 4(1)10}}{2(1)} = \frac{5{,}000}{0.002}$$

We can then determine the square root term with 5-digit arithmetic and chopping

$$\sqrt{(5{,}000.0)^2 - 4(1)10} = \sqrt{2{,}500{,}000 - 4(1)10} = \sqrt{24{,}999{,}960} \xrightarrow{\text{chopping}} \sqrt{24{,}999{,}000}$$
$$= 4{,}999.996 \xrightarrow{\text{chopping}} 4{,}999.9$$

Standard quadratic formula:

$$x_1 = \frac{5{,}000.0 + 4{,}999.9}{2} = \frac{9{,}999.9}{2} = 4{,}999.95 \xrightarrow{\text{chopping}} 4{,}999.9$$
$$x_2 = \frac{5{,}000.0 - 4{,}999.9}{2} = \frac{0.1}{2} = 0.05$$

Thus, although the first root is reasonably close to the true value ($\varepsilon_t = 0.002\%$), the second is considerably off ($\varepsilon_t = 2400\%$) due primarily to subtractive cancellation.

Equation (3.13):

$$x_1 = \frac{-2(10)}{-5{,}000.0 + 4{,}999.9} = \frac{-20}{-0.1} = 200$$
$$x_2 = \frac{-2(10)}{-5{,}000.0 - 4{,}999.9} = \frac{-20}{-9{,}999.9} = 0.002$$

For this case, the second root is well approximated, whereas the first is considerably off ($\varepsilon_t = 96\%$). Again, the culprit is the subtraction of two nearly equal numbers.

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,fx,Ea,n] = bisectnew(func,xl,xu,Ead,varargin)
% bisection roots zero
% [root,fx,Ea,n] = bisectnew(func,xl,xu,Ead,varargin)
%   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)
```

```
%   p1,p2,... = additional parameters used by func
% output:
%   root = real root
%   fx = function value at root
%   Ea = absolute error
%   n = iterations

if func(xl,varargin{:})*func(xu,varargin{:})>0  %if guesses do not bracket a
sign change
  disp('no bracket')    %display an error message
  return                %and terminate
end
% if necessary, assign default values
if nargin<4|isempty(Ead),Ead=0.000001;end  %if Ead blank set to 0.000001
xr = xl;
% compute n and round up to next highest integer
n = round(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
  Ea = abs(xr - xrold);
  test = func(xl,varargin{:})*func(xr,varargin{:});
  if test < 0
    xu = xr;
  elseif test > 0
    xl = xr;
  else
    Ea = 0;
  end
end
root = xr; fx = func(xr,varargin{:});
```

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

```
clear,clc,format long
fcd=@(cd,m,t,v) sqrt(9.81*m/cd)*tanh(sqrt(9.81*cd/m)*t)-v;
[root,fx,Ea,n] =bisectnew(fcd,0.1,0.2,0.0001,80,4,36)
```

The results for Prob. 5.1 are

```
root =
   0.14013671875000
fx =
   -4.063639939673180e-004
Ea =
    9.765625000002220e-005
n =
    10
```

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

$$x_r = 0.2 - \frac{-1.19738(0.1-0.2)}{0.860291-(-1.19738)} = 0.141809$$

$$f(0.1)f(0.141809) = 0.860291(-0.0352109) = -0.03029$$

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

$$x_r = 0.141809 - \frac{-0.0352109(0.1-0.141809)}{0.860291-(-0.0352109)} = 0.140165$$

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

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,fx,ea,iter]=falsepos(func,xl,xu,es,maxit,varargin)
% falsepos: root location zeroes
%   [root,fx,ea,iter]=falsepos(func,xl,xu,es,maxit,p1,p2,...):
%      uses false position to find the root of func
% input:
%   func = name of function
%   xl, xu = lower and upper guesses
%   es = desired relative error (default = 0.0001%)
%   maxit = maximum allowable iterations (default = 50)
%   p1,p2,... = additional parameters used by function
% output:
%   root = real root
%   fx = function value at root
%   ea = approximate relative error (%)
%   iter = number of iterations

if nargin<3,error('at least 3 input arguments required'),end
test = func(xl,varargin{:})*func(xu,varargin{:});
if test>0,error('no sign change'),end
if nargin<4|es<=0, es=0.0001;end
if nargin<5|maxit<=0, maxit=50;end
iter = 0; xr = xl;
while (1)
  xrold = xr;
  fl=func(xl,varargin{:});
  fu=func(xu,varargin{:});
  xr = xu - fu*(xl - xu)/(fl - fu);
  iter = iter + 1;
  if xr ~= 0,ea = abs((xr - xrold)/xr) * 100;end
  test = fl*func(xr,varargin{:});
  if test < 0
    xu = xr;
  elseif test > 0
    xl = xr;
  else
    ea = 0;
  end
  if ea <= es | iter >= maxit,break,end
end
root = xr; fx = func(xr,varargin{:});
```

The following script uses the function to solve Prob. 5.1:

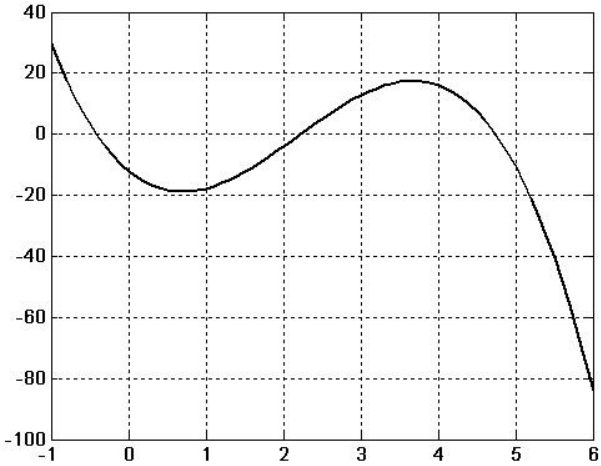
```
clear,clc,format long
fcd=@(cd) sqrt(9.81*80/cd)*tanh(sqrt(9.81*cd/80)*4)-36;
[root,fx,ea,iter]=falsepos(fcd,0.1,0.2,2)
```

The results for Prob. 5.1 are

```
root =
   0.14016503741282
fx =
   -9.964474382826438e-004
ea =
   1.17284536190266
iter =
     2
```

5.5 **(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

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

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

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

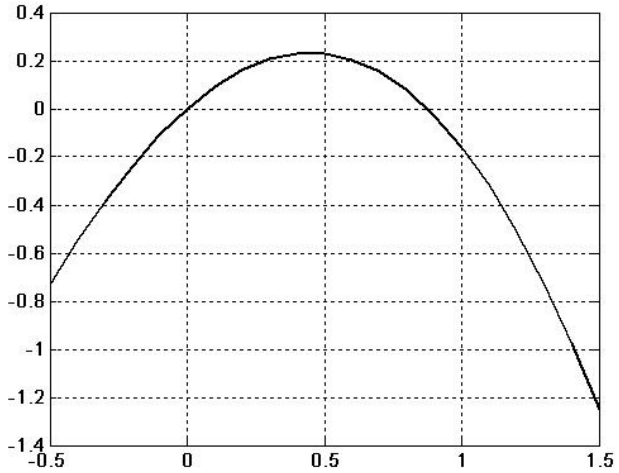
Therefore, the root is in the first interval and the upper guess is redefined as $x_u = -0.379449$. The remainder of the iterations are displayed in the following table:

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%.

5.6 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

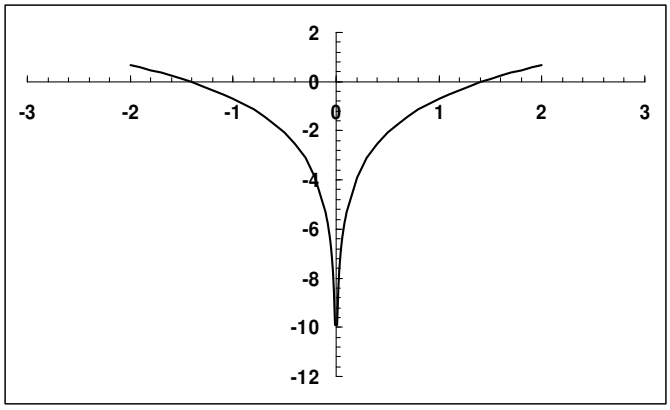
$$x_r = \frac{0.75+1}{2} = 0.875$$
$$\varepsilon_a = \left|\frac{0.875-0.75}{0.875}\right|100\% = 14.29\%$$
$$f(0.75)f(0.875) = 0.119139(0.0019185) = 0.000229$$

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%.

5.7 **(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%.

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

$$n = \log_2\left(\frac{\Delta x^0}{E_{a,d}}\right) = \log_2\left(\frac{35}{0.05}\right) = 9.4512$$

which can be rounded up to 10 iterations.

(b) Here is an M-file that evaluates the temperature in ^oC using 10 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 = 1:10
  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 =
   26.7627

>> TempEval(10)
ans =
   15.4150

>> TempEval(14)
ans =
    1.5381
```

These results correspond to absolute approximate errors of E_a = 0.034. The true errors are all less than the desired value of 0.05: E_t = 0.018, 0.027, and 0.017, respectively.

5.9 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) $M = 265x - 5.5555556x^3$

3<x<6

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

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

6<x<10

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

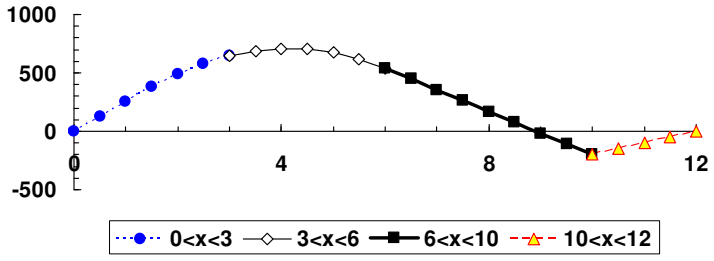
(3) $M = -185x + 1650$

10<x<12

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

(4) $M = 100x - 1200$

A plot of these equations can be generated:



Combining Equations:

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

$M = -185x + 1650$

Set $x_L = 6; x_U = 10$

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

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

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

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

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

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

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

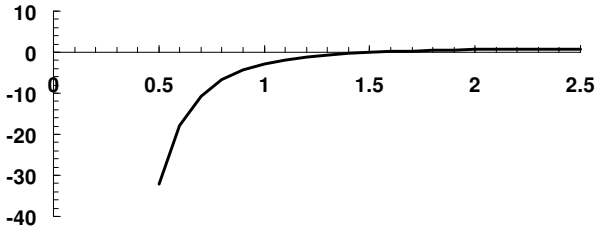
$M(x_L) = 2.34375$
$M(x_U) = -3.4375$ $\qquad x_r = \dfrac{8.90625+8.9375}{2} = 8.921875$
$M(x_R) = -0.546875 \rightarrow$ *replaces* x_U

$M(x_L) = 2.34375$
$M(x_U) = -0.546875$ $\qquad x_r = \dfrac{8.90625+8.921875}{2} = 8.9140625$
$M(x_R) = 0.8984$ $\qquad$ **Therefore,** $x = 8.91$ feet

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

$$x_r = \frac{1.5 + 2.5}{2} = 2 \qquad \varepsilon_a = \left|\frac{2 - 1.5}{2}\right| 100\% = 25\%$$
$$f(1.5)f(2) = -0.030946(0.601809) = -0.018624$$

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

i	x_l	$f(x_l)$	x_u	$f(x_u)$	x_r	$f(x_r)$	ε_a
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

$$x_r = 2.45083 - \frac{0.79987(0.5 - 2.45083)}{-32.25821 - 0.79987} = 2.40363 \qquad \varepsilon_a = \left|\frac{2.40363 - 2.45083}{2.40363}\right| 100\% = 1.96\%$$
$$f(0.5)f(2.40363) = -32.2582(0.78612) = -25.35893$$

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

i	x_l	$f(x_l)$	x_u	$f(x_u)$	x_r	$f(x_r)$	ε_a
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.

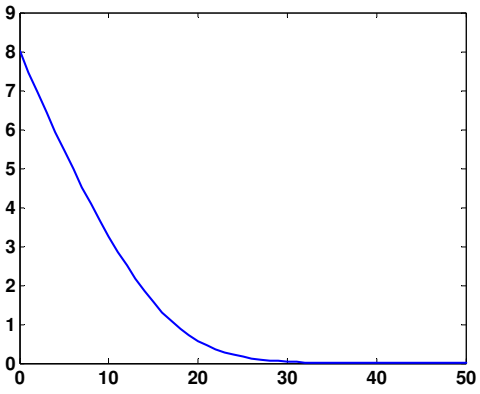
Discussion: This is a classic example of a case where false position performs poorly and is inferior to bisection. Insight into these results can be gained by examining the plot that was developed in part **(a)**. This function violates the premise upon which false position was based–that is, if $f(x_u)$ is much closer to zero than $f(x_l)$, then the root is closer to x_u than to x_l (recall Figs. 5.8 and 5.9). Because of the shape of the present function, the opposite is true.

5.11 The problem amounts to determining the root of

$$f(S) = S_0 - v_m t + k_s \ln(S_0 / S) - S$$

The following script calls the `bisect` function (Fig. 5.7) for various values of t in order to generate the solution.

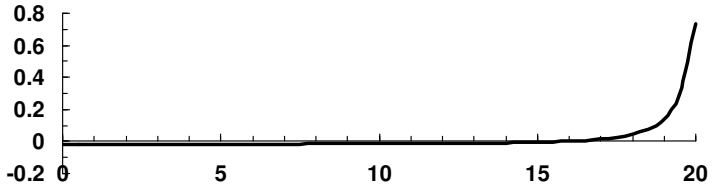
```
clear,clc,clf
S0=8;vm=0.7;ks=2.5;
f = @(S,t) S0 - vm * t + ks * log(S0 / S) - S;
t=0:50;S=0:50;
n=length(t);
for i = 1:n
  S(i)=bisect(f,0.00001,10.01,1e-6,100,t(i));
end
plot(t,S)
```



5.12 The function to be solved is

$$f(x) = \frac{(4+x)}{(42-2x)^2(28-x)} - 0.016 = 0$$

(a) A plot of the function indicates a root at about x = 16.



(b) The shape of the function indicates that false position would be a poor choice (recall Fig. 5.9). Bisection with initial guesses of 0 and 20 can be used to determine a root of 15.85938 after 8 iterations with ε_a = 0.493%. Note that false position would have required 68 iterations to attain comparable accuracy.

i	x_l	x_u	x_r	$f(x_l)$	$f(x_r)$	$f(x_l)\times f(x_r)$	ε_a
1	0	20	10	-0.01592	-0.01439	0.000229	100.000%
2	10	20	15	-0.01439	-0.00585	8.42×10^{-5}	33.333%
3	15	20	17.5	-0.00585	0.025788	-0.00015	14.286%
4	15	17.5	16.25	-0.00585	0.003096	-1.8×10^{-5}	7.692%
5	15	16.25	15.625	-0.00585	-0.00228	1.33×10^{-5}	4.000%
6	15.625	16.25	15.9375	-0.00228	0.000123	-2.8×10^{-7}	1.961%
7	15.625	15.9375	15.78125	-0.00228	-0.00114	2.59×10^{-6}	0.990%
8	15.78125	15.9375	15.85938	-0.00114	-0.00052	5.98×10^{-7}	0.493%

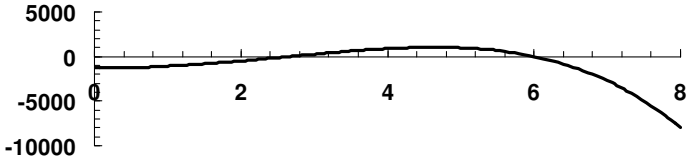
5.13 This problem can be solved by determining the root of the derivative of the elastic curve

$$\frac{dy}{dx} = 0 = \frac{w_0}{120EIL}\left(-5x^4 + 6L^2x^2 - L^4\right)$$

Therefore, after substituting the parameter values (in meter-kilogram-second units), we must determine the root of

$$f(x) = -5x^4 + 216x^2 - 1,296 = 0$$

A plot of the function indicates a root at about x = 2.6 m.



Using initial guesses of 0 and 5 m, bisection can be used to determine the root. Here are the first few iterations:

i	x_l	x_u	x_r	$f(x_l)$	$f(x_r)$	$f(x_l)\times f(x_r)$	ε_a
1	0	5	2.5	-1296.00	-141.31	183141	
2	2.5	5	3.75	-141.31	752.73	-106370	33.33%
3	2.5	3.75	3.125	-141.31	336.54	-47557	20.00%
4	2.5	3.125	2.8125	-141.31	99.74	-14095	11.11%
5	2.5	2.8125	2.65625	-141.31	-20.89	2952	5.88%

After 30 iterations, the root is determined as x = 2.683282 m. This value can be substituted into Eq. (P5.13) to compute the maximum deflection as

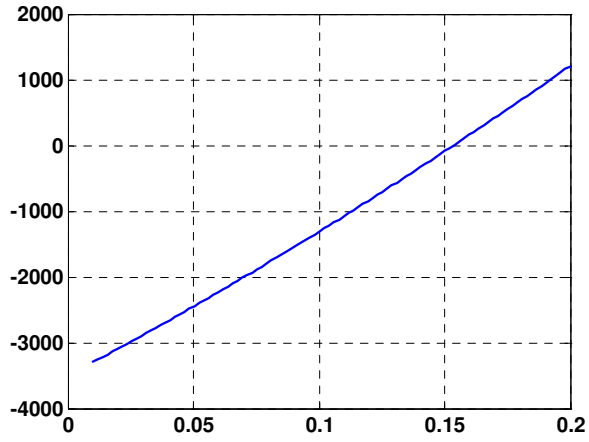
$$y = \frac{250,000}{120(5000\times10^9)0.0003(6)}(-(2.683282)^5 + 216(2.683282)^3 - 1,296(2.683282)) = -0.00515 \text{ m}$$

5.14 The solution can be formulated as

$$f(i) = 35,000\frac{i(1+i)^7}{(1+i)^7 - 1} - 8,500$$

A script can be developed to generate a plot of the function and obtain the solution with the `bisect` function:

```
clear,clc,clf,format short g
P=35000;A=8500;n=7;
f = @(irate) P*irate.*(1+irate).^n./((1+irate).^n-1)-A;
iplot=linspace(0.01,.2)';
fplot=f(iplot);
plot(iplot,fplot),grid
[xr,fx,ea,iter]=bisect(f,0.01,.3,0.00005)
```



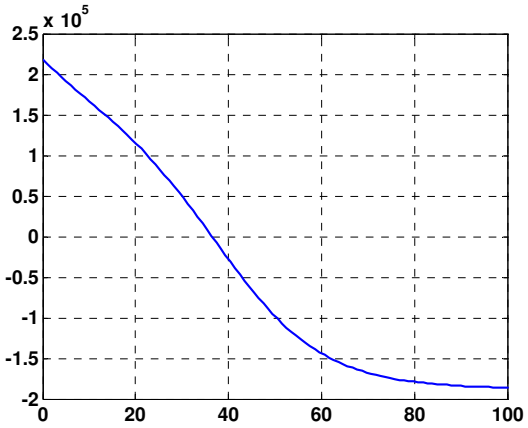

```
xr =
      0.15346
fx =
  -0.00026251
ea =
  4.5056e-005
iter =
    22
```

5.15 (a) The solution can be formulated as

$$f(t) = 1.2\left(80,000e^{-0.05t} + 110,000\right) - \frac{320,000}{1+31e^{-0.09t}}$$

A plot of this function can be generated with a script to suggest a root at about 36 years:

```
clear,clc,clf,format short g
Pumax=80000;ku=0.05;Pumin=110000;
Psmax=320000;P0=10000;ks=0.09;
f = @(t) 1.2*(Pumax*exp(-ku*t)+Pumin)-Psmax./(1+(Psmax/P0-1)*exp(-ks*t));
iplot=linspace(0,100)';
fplot=f(iplot);
plot(iplot,fplot),grid
```



(b) The false-position method can be implemented with the results summarized as

i	t_l	t_u	$f(t_l)$	$f(t_u)$	t_r	$f(t_r)$	$f(t_l)\times f(t_r)$	ε_a
1	0	100.0000	218000	-186134	**53.9426**	-119280	-2.600E+10	
2	0	53.9426	218000	-119280	**34.8656**	12309.95	2.684E+09	54.716%
3	34.8656	53.9426	12310	-119280	**36.6502**	-1817.89	-2.238E+07	4.869%
4	34.8656	36.6502	12310	-1817.89	**36.4206**	4.081795	5.025E+04	0.631%
5	36.4206	36.6502	4	-1817.89	**36.4211**	0.000548	2.236E-03	0.001%

The root is determined to be $t = 36.4211$. At this time, the ratio of the suburban to the urban population is 147,538/122,948 = 1.2.

5.16 The solution can be formulated as

$$f(N) = 0 = \frac{2}{q\left(N + \sqrt{N^2 + 4n_i^2}\right)\mu} - \rho$$

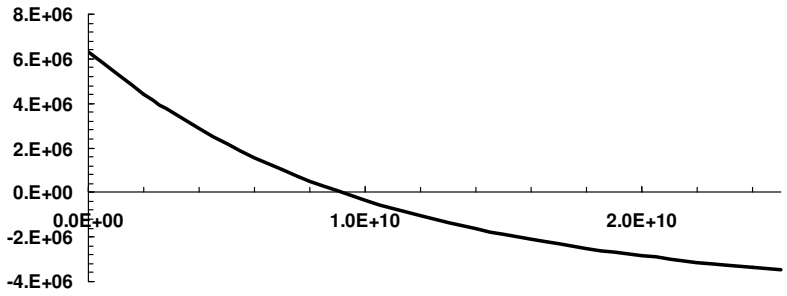
where

$$\mu = 1360\left(\frac{1000}{300}\right)^{-2.42} = 73.81971$$

Substituting this value along with the other parameters gives

$$f(N) = 0 = \frac{2}{1.25494\times10^{-17}\left(N + \sqrt{N^2 + 1.54256\times10^{20}}\right)} - 6.5\times10^6$$

A plot of this function indicates a root at about $N = 9\times10^9$.



(a) The bisection method can be implemented with the results for the first 5 iterations summarized as

i	N_l	N_u	N_r	$f(N_l)$	$f(N_r)$	$f(N_l)\times f(N_r)$	ε_a
1	0.000×10^{0}	2.500×10^{10}	1.250×10^{10}	6.33×10^{6}	-1.21×10^{6}	-7.7×10^{12}	100.000%
2	0.000×10^{0}	1.250×10^{10}	6.250×10^{9}	6.33×10^{6}	1.41×10^{6}	8.91×10^{12}	100.000%
3	6.250×10^{9}	1.250×10^{10}	9.375×10^{9}	1.41×10^{6}	-1.09×10^{5}	-1.5×10^{11}	33.333%
4	6.250×10^{9}	9.375×10^{9}	7.813×10^{9}	1.41×10^{6}	5.88×10^{5}	8.27×10^{11}	20.000%
5	7.813×10^{9}	9.375×10^{9}	8.594×10^{9}	5.88×10^{5}	2.25×10^{5}	1.32×10^{11}	9.091%

After 16 iterations, the root is 9.114×10^{9} with an approximate relative error of 0.004%.

(b) The false position method can be implemented with the results for the first 5 iterations summarized as

i	N_l	$f(N_l)$	N_u	$f(N_u)$	N_r	$f(N_r)$	$f(N_l)*f(N_r)$	ε_a
1	0.000×10^{0}	6.33×10^{6}	2.500×10^{10}	-3.49×10^{6}	1.612×10^{10}	-2.13×10^{6}	-1.3×10^{13}	
2	0.000×10^{0}	6.33×10^{6}	1.612×10^{10}	-2.13×10^{6}	1.206×10^{10}	-1.07×10^{6}	-6.8×10^{12}	33.639%
3	0.000×10^{0}	6.33×10^{6}	1.206×10^{10}	-1.07×10^{6}	1.031×10^{10}	-4.76×10^{5}	-3×10^{12}	16.973%
4	0.000×10^{0}	6.33×10^{6}	1.031×10^{10}	-4.76×10^{6}	9.591×10^{9}	-1.97×10^{5}	-1.2×10^{12}	7.513%
5	0.000×10^{0}	6.33×10^{6}	9.591×10^{9}	-1.97×10^{6}	9.302×10^{9}	-7.89×10^{4}	-5×10^{11}	3.106%

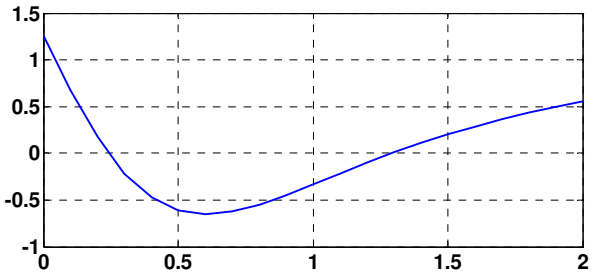
After 12 iterations, the root is 9.114×10^{9} with a relative error of 0.005%.

5.17 Using the given values, the roots problem to be solved is

$$f(x) = 0 = 1.25 - 3.576516\frac{x}{(x^2 + 0.7225)^{3/2}}$$

A script can be developed to generate a plot:

```
format short g
e0=8.9e-12;F=1.25;q=2e-5;Q=2e-5;a=0.85;
f = @(x) F-1/(4*pi*e0)*q*Q*x./(x.^2+a^2).^(3/2);
xx=[0:0.1:2];
ff=f(xx);
plot(xx,ff),grid
```



The plot indicates roots at about 0.25 and 1.3. The bisect function can be used to determine these roots as

```
>> [x,fx,ea,iter]=bisect(f,0,0.5)
x =
       0.24104
fx =
 -4.6805e-007
ea =
  9.8911e-005
iter =
    21

>> [x,fx,ea,iter]=bisect(f,0.5,2)
x =
       1.2913
fx =
 -3.7927e-009
ea =
  5.5391e-005
iter =
    21
```

Therefore, the roots are 0.24104 and 1.2913.

5.18 The solution can be formulated as

$$f(f) = 4\log_{10}\left(\mathrm{Re}\sqrt{f}\right) - 0.4 - \frac{1}{\sqrt{f}}$$

We want our program to work for Reynolds numbers between 2,500 and 1,000,000. Therefore, we must determine the friction factors corresponding to these limits. This can be done with any root location method to yield 0.011525 and 0.002913. Therefore, we can set our initial guesses as $x_l = 0.0028$ and $x_u = 0.012$. Equation (5.6) can be used to determine the number of bisection iterations required to attain an absolute error less than 0.000005,

$$n = \log_2\left(\frac{\Delta x^0}{E_{a,d}}\right) = \log_2\left(\frac{0.012 - 0.0028}{0.000005}\right) = 10.8454$$

which can be rounded up to 11 iterations. Here is a MATLAB function that is set up to implement 11 iterations of bisection to solve the problem.

```
function f=Fanning(func,xl,xu,varargin)
test = func(xl,varargin{:})*func(xu,varargin{:});
if test>0,error('no sign change'),end
for i = 1:11
```

```
  xr = (xl + xu)/2;
  test = func(xl,varargin{:})*func(xr,varargin{:});
  if test < 0
    xu = xr;
  elseif test > 0
    xl = xr;
  else
    break
  end
end
f=xr;
```

This can be implemented in MATLAB. For example,

```
>> vk=@(f,Re) 4*log10(Re*sqrt(f))-0.4-1/sqrt(f);
>> format long
>> f=Fanning(vk,0.0028,0.012,2500)
f =
   0.01152832031250
```

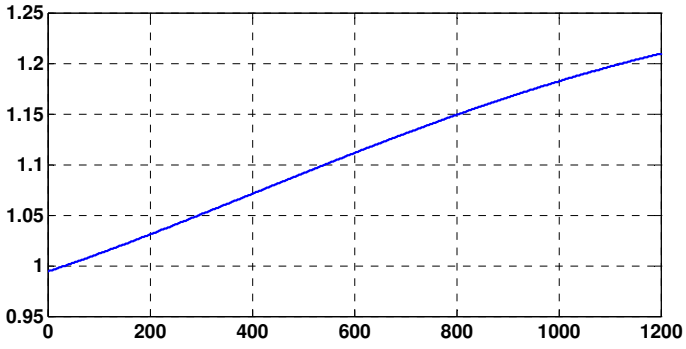
Here are additional results for a number of values within the desired range. We have included the true value and the resulting error to verify that the results are within the desired error criterion of $E_a < 5\times10^{-6}$.

Re	Root	Truth	E_t
2500	0.0115283203125	0.0115247638118	$3.56\text{x}10^{-6}$
3000	0.0108904296875	0.0108902285840	$2.01\text{x}10^{-7}$
10000	0.0077279296875	0.0077271274071	$8.02\text{x}10^{-7}$
30000	0.0058771484375	0.0058750482511	$2.10\text{x}10^{-6}$
100000	0.0045025390625	0.0045003757287	$2.16\text{x}10^{-6}$
300000	0.0036220703125	0.0036178949673	$4.18\text{x}10^{-6}$
1000000	0.0029123046875	0.0029128191460	$5.14\text{x}10^{-7}$

5.19 A script can be developed to generate the plot and the solution with the book's MATLAB function `bisect`:

```
clear,clc,clf,format short g
cp = @(T) 1.952e-14*T.^4-9.5838e-11*T.^3+9.7215e-8*T.^2+1.671e-4*T+0.99403;
TT=[0:1200];
cpp=cp(TT);
plot(TT,cpp),grid
cpr = @(T) 1.952e-14*T^4-9.5838e-11*T^3+9.7215e-8*T^2+1.671e-4*T-0.10597;
[root,fx,ea,iter] = bisect(cpr,0,1200)
```

The results indicate a root at about $T = 544.09$.



```
root =
       544.09
fx =
 -4.4688e-008
ea =
  5.2584e-005
iter =
    22
```

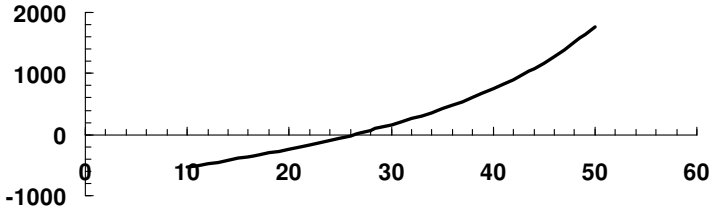
5.20 The solution can be formulated as

$$f(t) = u \ln \frac{m_0}{m_0 - qt} - gt - v$$

Substituting the parameter values gives

$$f(t) = 1{,}800 \ln \frac{160{,}000}{160{,}000 - 2{,}600t} - 9.81t - 750$$

A plot of this function indicates a root at about $t = 21$.



Because two initial guesses are given, a bracketing method like bisection can be used to determine the root,

i	t_l	t_u	t_r	$f(t_l)$	$f(t_r)$	$f(t_l) \times f(t_r)$	ε_a
1	10	50	30	-528.899	158.9182	-84051.7	
2	10	30	20	-528.899	-238.723	126260.5	50.00%
3	20	30	25	-238.723	-56.9155	13587.07	20.00%
4	25	30	27.5	-56.9155	46.13327	-2625.7	9.09%
5	25	27.5	26.25	-56.9155	-6.52111	371.1524	4.76%
6	26.25	27.5	26.875	-6.52111	19.51345	-127.249	2.33%
7	26.25	26.875	26.5625	-6.52111	6.424319	-41.8937	1.18%
8	26.25	26.5625	26.40625	-6.52111	-0.0662	0.431679	0.59%

Thus, after 8 iterations, the approximate error falls below 1% with a result of $t = 26.40625$. Note that if the computation is continued, the root can be determined as 26.40784796.

5.21 Eqs. (5.11), (5.14) and (5.16) can be substituted into Eq. (5.13) to yield

$$0 = \frac{K_1}{10^6[\mathrm{H}^+]} K_H p_{CO_2} + 2\frac{K_2 K_1}{10^6[\mathrm{H}^+]^2} K_H p_{CO_2} + \frac{K_w}{[\mathrm{H}^+]} - [\mathrm{H}^+] - Alk$$

The function then becomes

```
function f = fpHAlk(pH,pCO2,Alk)
K1=10^-6.3;K2=10^-10.3;Kw=10^-14;
KH=10^-1.46;
H=10^-pH;
f=K1/(1e6*H)*KH*pCO2+2*K2*K1/(1e6*H)*KH*pCO2+Kw/H-H-Alk;
```

For 2008, the resulting pH can be determined with

```
>> [pH2008 fx ea iter]=bisect(@fpHAlk,2,12,1e-8,50,386,0.4e-3)

pH2008 =
    7.7748
fx =
 -2.3932e-013
ea =
  7.4867e-009
iter =
    34
```

Notice how the presence of alkalinity has significantly raised the pH.

5.22 Archimedes principle says that

$$\rho_s V_s g = \rho_w V_w g$$

where V_s = the total sphere volume (m^3) and V_w = the below-water volume (m^3). The below-water volume is equal to the total volume minus the above-water volume,

$$V_w = \frac{4\pi r^3}{3} - \frac{\pi h^2}{3}(3r - h)$$

Therefore, *Archimedes principle* amounts to finding the root of

$$0 = \rho_w \left[\frac{4\pi r^3}{3} - \frac{\pi h^2}{3}(3r - h) \right] g - \rho_s \frac{4\pi r^3}{3} g$$

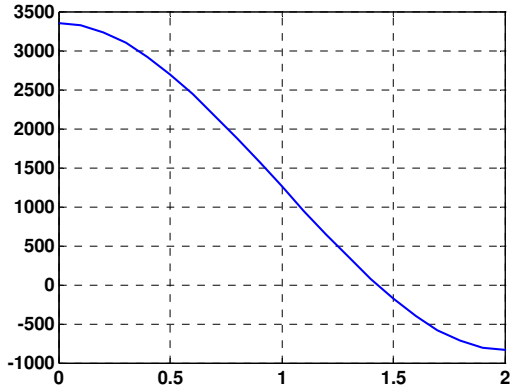
or collecting terms

$$0 = \left[\frac{4\pi r^3}{3}(\rho_w - \rho_s) - \rho_w \frac{\pi h^2}{3}(3r - h) \right]$$

Given the problem parameters, the following script can be employed to determine the root with the `bisect` program from the text (Fig. 5.7):

```
clear,clc,clf,format short g
r=1;rhos=200;rhow=1000;
fh=@(h) 4/3*r^3*pi*(rhow-rhos)-rhow*pi*h.^2/3.*(3*r-h);
h=[0:2*r/20:2*r];fhh=fh(h);
plot(h,fhh),grid
[height f ea iter]=bisect(fh,0,2*r)
```

The result is




```
height =
    1.4257
f =
   -0.0018
ea =
  6.6891e-005
iter =
    21
```

5.23 Archimedes principle says that

$$\rho_s V_f g = \rho_w V_w g$$

where V_f = the total frustrum volume (m^3) and V_w = the below-water volume (m3). The total frustrum volume is simply

$$V_f = \frac{\pi h}{3}(r_1^2 + r_2^2 + r_1 r_2)$$

In order to determine the below-water volume, we must first relate the radius to the height above water as in

$$r = r_1 + \frac{r_2 - r_1}{h} h_1$$

The below water volume is

$$V_w = \frac{\pi (h - h_1)}{3}\left[r_2^2 + \left(r_1 + \frac{r_2 - r_1}{h} h_1 \right)^2 + r_2 \left(r_1 + \frac{r_2 - r_1}{h} h_1 \right) \right]$$

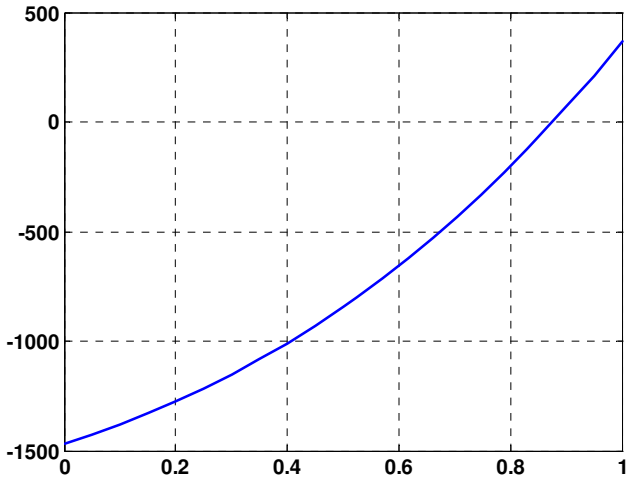
Therefore, the solution amounts to determining the value of h_1 that satisfies

$$\rho_s \left(\frac{\pi h}{3}(r_1^2 + r_2^2 + r_1 r_2) \right) - \rho_w \left(\frac{\pi (h - h_1)}{3}\left[\left(r_1 + \frac{r_2 - r_1}{h} h_1 \right)^2 + r_2^2 + \left(r_1 + \frac{r_2 - r_1}{h} h_1 \right) r_2 \right] \right) = 0$$

Given the problem parameters, the following script can be employed to determine the root with the `bisect` program from the text (Fig. 5.7):

```
r1=0.5;r2=1;h=1;rhof=200;rhow=1000;
fh1=@(h1) rhos*pi*h/3*(r1^2+r2^2+r1*r2)-...
    rhow*pi*(h-h1)/3.*((r1+(r2-r1)/h*h1).^2+r2^2+(r1+(r2-r1)/h*h1)*r2);
h1=[0:h/20:h];fhh=fh1(h1);
plot(h,fhh)
[height f ea iter]=bisect(fh1,0,1)
```

The result is




```
height =
      0.87578
f =
   0.00073518
ea =
  5.4447e-005
iter =
=
     21
```

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:

i	x_i	$\lvert\varepsilon_a\rvert$	E_t	$E_{t,i} / E_{t,i-1}$
0	0.500000		0.268648	
1	0.649637	23.0339%	0.119011	0.44300
2	0.721524	9.9632%	0.047124	0.39596
3	0.750901	3.9123%	0.017747	0.37660
4	0.762097	1.4691%	0.006551	0.36914
5	0.766248	0.5418%	0.002400	0.36632
6	0.767772	0.1984%	0.000876	0.36514
7	0.768329	0.0725%	0.000319	0.36432
8	0.768532	0.0265%	0.000116	0.36297
9	0.768606	0.0097%	0.000042	0.35956

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

To confirm that the scheme is linearly convergent, according to the book, the ratio of the errors between iterations should be

$$\frac{E_{i+1}}{E_i} = g'(\xi) = \frac{1}{2\sqrt{\xi}}\cos\left(\sqrt{\xi}\right)$$

Substituting the root for ξ gives a value of 0.365 which is close to the values in the last column of the table.

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:

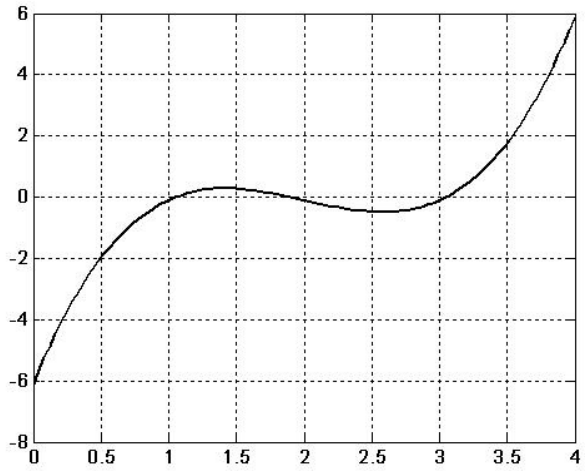
iteration	x_i	$f(x_i)$	$f'(x_i)$	$\|\varepsilon_a\|$
0	5	−11.5	−7.3	
1	3.424658	−2.23353	−4.46438	46.0000%
2	2.924357	−0.22527	−3.56384	17.1081%
3	2.861147	−0.00360	−3.45006	2.2093%
4	2.860105	−9.8E−07	−3.44819	0.0364%
5	2.860104	−7.2E−14	−3.44819	0.0000%

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:

$x_0 = 3.5$ $\quad f(x_0) = 1.775$
$x_0 + \delta x_0 = 3.535$ $\quad f(x_0 + \delta x_0) = 1.981805$

$$x_1 = 3.5 - \frac{0.01(3.5)1.775}{1.981805-1.775} = 3.199597$$
$$|\varepsilon_a| = \left|\frac{3.199597-3.5}{3.199597}\right| \times 100\% = 9.389\%$$

Second iteration:

$x_1 = 3.199597$ $\quad f(x_1) = 0.426661904$
$x_1 + \delta x_1 = 3.271725$ $\quad f(x_1 + \delta x_1) = 0.536512631$

$$x_2 = 3.199597 - \frac{0.01(3.199597)0.426661904}{0.536513-0.426661904} = 3.075324$$
$$|\varepsilon_a| = \left|\frac{3.075324-3.199597}{3.075324}\right| \times 100\% = 4.041\%$$

Third iteration:

$x_2 = 3.075324$ $\quad f(x_2) = 0.068096$
$x_2 + \delta x_2 = 3.143675$ $\quad f(x_2 + \delta x_2) = 0.147105$

$$x_3 = 3.075324 - \frac{0.01(3.075324)0.068096}{0.147105 - 0.068096} = 3.048818$$

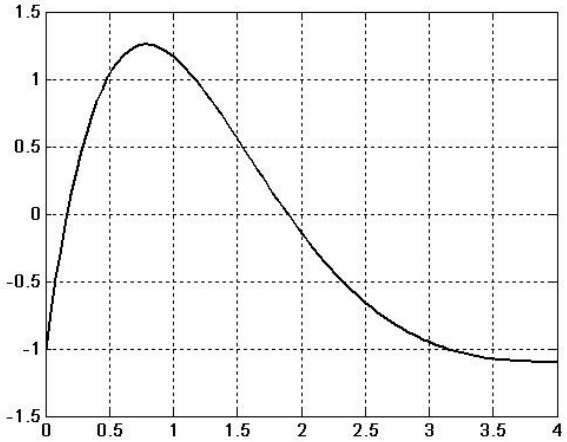
$$|\varepsilon_a| = \left|\frac{3.048818 - 3.075324}{3.048818}\right| \times 100\% = 0.869\%$$

(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 0.3, 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:

$x_{-1} = 0.5$ $\quad f(x_{-1}) = 1.03550$
$x_0 = 0.4$ $\quad f(x_0) = 0.827244$

$$x_1 = 0.4 - \frac{0.827244(0.5 - 0.4)}{1.03550 - 0.827244} = 0.002782$$

$$|\varepsilon_a| = \left|\frac{0.002782 - 0.4}{0.002782}\right| \times 100\% = 14{,}278\%$$

Second iteration:

$x_0 = 0.4$ $\quad f(x_0) = 0.827244$
$x_1 = 0.002782$ $\quad f(x_1) = -0.980580$

$$x_2 = 0.002782 - \frac{-0.98058(0.4 - 0.002782)}{0.827244 - (-0.98058)} = 0.218237$$

$$|\varepsilon_a| = \left|\frac{0.218237 - 0.002782}{0.218237}\right| \times 100\% = 98.725\%$$

Third iteration:

$x_1 = 0.002782$ $\quad f(x_1) = -0.980580$
$x_2 = 0.218237$ $\quad f(x_2) = 0.218411$

$$x_3 = 0.218237 - \frac{0.218411(0.002782 - 0.218237)}{-0.98058 - 0.218411} = 0.178989$$

$$|\varepsilon_a| = \left|\frac{0.178989 - 0.218237}{0.178989}\right| \times 100\% = 21.927\%$$

(d) For the modified secant method:

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:

$x_1 = 0.14369799$ $\quad f(x_1) = -0.13175$

$x_1 + \delta x_1 = 0.14513497$ $\quad f(x_1 + \delta x_1) = -0.12439$

$$x_2 = 0.143698 - \frac{0.01(0.143698)(-0.13175)}{-0.12439 - (-0.13175)} = 0.169412$$

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

Third iteration:

$x_2 = 0.169411504$ $\quad f(x_2) = -0.00371$

$x_2 + \delta x_2 = 0.17110562$ $\quad f(x_2 + \delta x_2) = 0.004456$

$$x_3 = 0.169411504 - \frac{0.01(0.169411504)(-0.00371)}{0.004456 - (-0.00371)} = 0.170180853$$

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

Errata: In the first printing, the problem specified five iterations.

Fourth iteration:

$x_3 = 0.170180853$ $\quad f(x_3) = 4.14 \times 10^{-6}$

$x_3 + \delta x_3 = 0.17188266$ $\quad f(x_3 + \delta x_3) = 0.008189$

$$x_4 = 0.170180853 - \frac{0.01(0.170180853)(4.14 \times 10^{-6})}{0.008189 - 4.14 \times 10^{-6}} = 0.170179992$$

$$|\varepsilon_a| = \left|\frac{0.170179992 - 0.170180853}{0.170179992}\right| \times 100\% = 0.001\%$$

Fifth iteration:

$x_3 = 0.170179992$ $\quad f(x_3) = -8.5 \times 10^{-9}$

$x_3 + \delta x_3 = 0.17188179$ $\quad f(x_3 + \delta x_3) = 0.008185$

$$x_4 = 0.170179992 - \frac{0.01(0.170179992)(-8.5 \times 10^{-9})}{0.008185 - (-8.5 \times 10^{-9})} = 0.170179994$$

$$|\varepsilon_a| = \left|\frac{0.170179994 - 0.170179992}{0.170179994}\right| \times 100\% = 0.000\%$$

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	$f(x_i)$	$f'(x_i)$	$\|\varepsilon_a\|$
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:

First iteration:

$x_0 = 0.5825$ $\quad f(x_0) = 50.06217$

$x_0 + \delta x_0 = 0.611625$ $\quad f(x_0 + \delta x_0) = 49.15724$

$$x_1 = 0.5825 - \frac{0.05(0.5825)50.06217}{49.15724 - 50.06217} = 2.193735$$
$$|\varepsilon_a| = \left|\frac{2.193735 - 0.5825}{2.193735}\right| \times 100\% = 73.447\%$$

Second iteration:

$x_1 = 2.193735$ $\quad f(x_1) = -21.1969$

$x_1 + \delta x_1 = 2.303422$ $\quad f(x_1 + \delta x_1) = -21.5448$

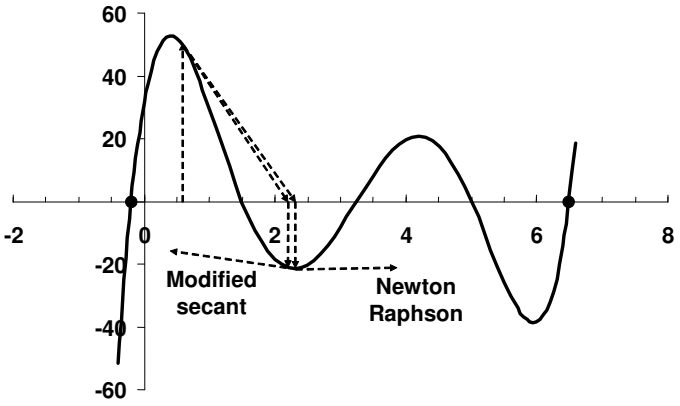
$$x_2 = 2.193735 - \frac{0.05(2.193735)(-21.1969)}{-21.5448 - (-21.1969)} = -4.48891$$

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

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:

```
format long
f=@(x) x^3-6*x^2+11*x-6.1;
secant(f,2.5,3.5)
ans =
   3.046680527126298
```

6.7

```
function root = modsec(func,xr,delta,es,maxit)
% modsec(func,xr,delta,es,maxit):
%   uses 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:

```
format long
f=@(x) x^3-6*x^2+11*x-6.1;
modsec(f,3.5,0.02)
ans =
   3.046682670215557
```

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%

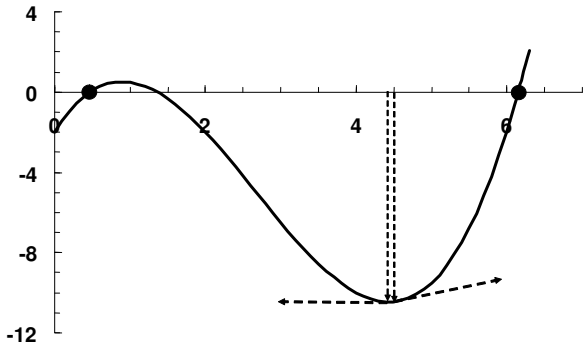
Thus, after an initial jump, the computation eventually settles down and converges on a root at x = 6.156325.

(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%
.				
.				
.				
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

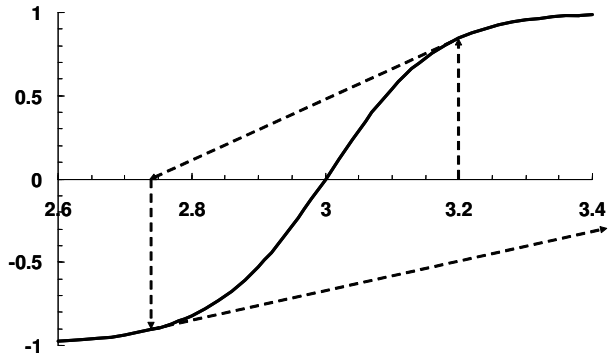
$$x_{i+1} = x_i - \frac{\tanh\left(x_i^2 - 9\right)}{2x_i \operatorname{sech}^2\left(x_i^2 - 9\right)}$$

Using an initial guess of 3.2, the iterations proceed as

iteration	x_i	$f(x_i)$	$f'(x_i)$	ε_a
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%

Note that on the fourth iteration, the computation should go unstable.

(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.



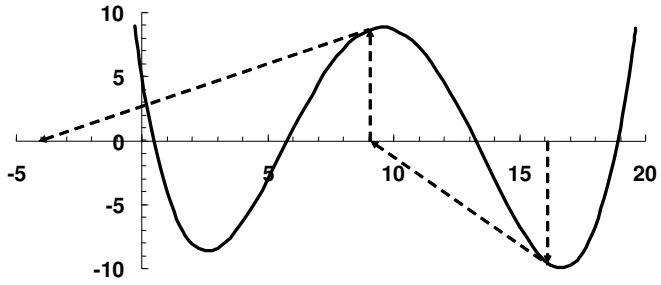
6.12 The formula for Newton-Raphson is

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

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.

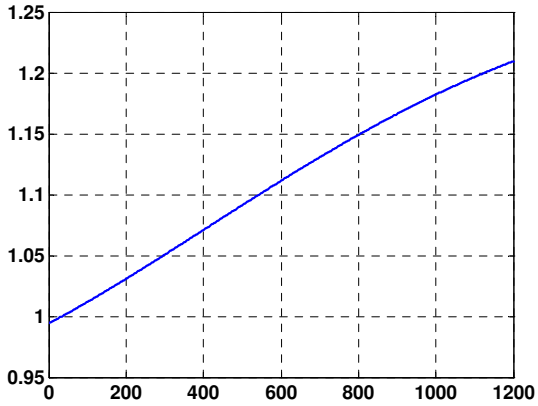


6.13 The solution can be formulated as

$$f(T) = 0 = -0.10597 + 1.671\times10^{-4}T + 9.7215\times10^{-8}T^2 - 9.5838\times10^{-11}T^3 + 1.9520\times10^{-14}T^4$$

A MATLAB script can be used to generate the plot and determine all the roots of this polynomial,

```
clear,clc,clf,format long g
cp=[1.952e-14 -9.5838e-11 9.7215e-8 1.671e-4 0.99403];
T=[0:1200];
cp_plot=polyval(cp,T);
plot(T,cp_plot),grid
x=[1.952e-14 -9.5838e-11 9.7215e-8 1.671e-4 -0.10597];
roots(x)

ans =
       2748.3 +     1126.3i
       2748.3 -     1126.3i
        -1131
       544.09
```



The only realistic value is 544.09. This value can be checked using the `polyval` function,

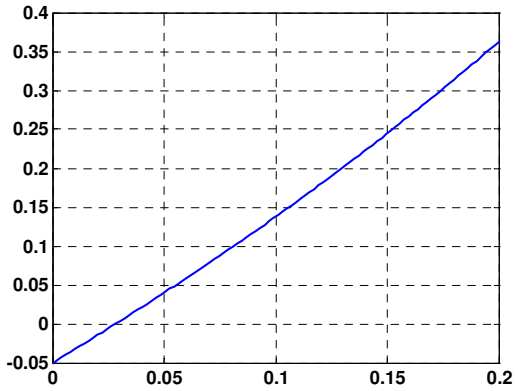
```
>> polyval(x,544.09)
    ans =
      4.9333e-007
```

6.14 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=@(x) x./(1-x).*sqrt(6./(2+x))-0.05;
x = linspace(0,.2);
y = f(x);
plot(x,y),grid
```



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

```
format long
fzero(f,0.03)

ans =
   0.028249441148471
```

6.15 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

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

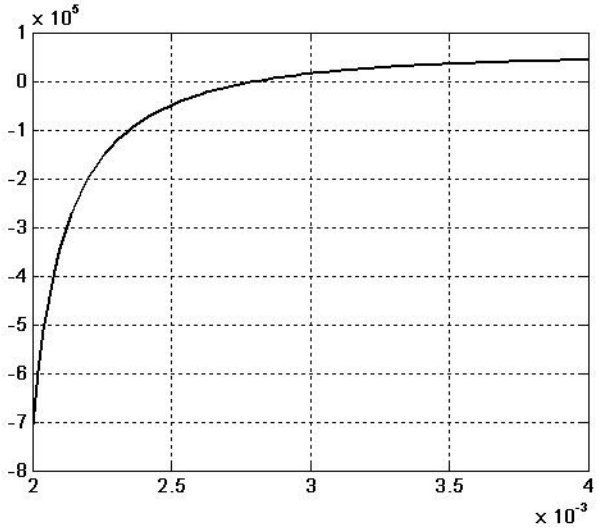
MATLAB can be used to generate a plot of the function and to solve for the root. One way to do this is to develop an M-file for the function,

```
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

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

6.16 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
```

6.17 (a) The function to be evaluated is

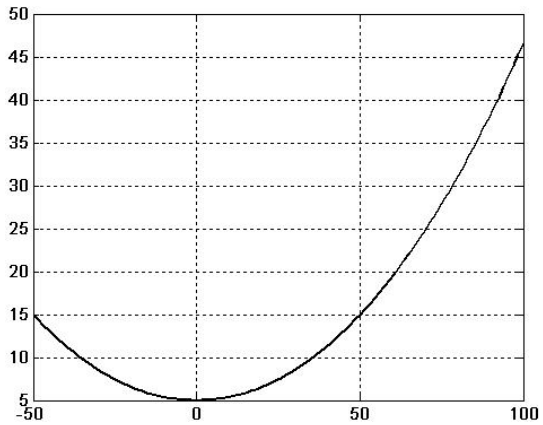
$$f(T_A) = 10 - \frac{T_A}{10}\cosh\left(\frac{500}{T_A}\right) + \frac{T_A}{10}$$

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
```

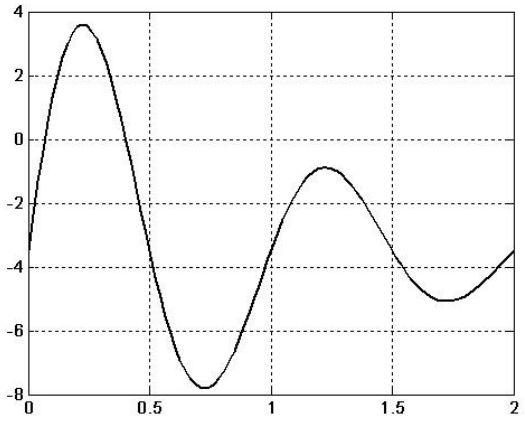


6.18 The function to be evaluated is

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

A plot can be generated with MATLAB,

```
>> t = linspace(0,2);
>> ft = @(t) 9*exp(-t) .* sin(2*pi*t) - 3.5;
>> y=ft(t);
>> 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(ft,[0 0.2])
t =
   0.06835432096851

>> t = fzero(ft,[0.2 0.8])
t =
   0.40134369265980
```

6.19 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

$$f(\omega) = 0.01 - \sqrt{\frac{1}{50625} + \left(0.6\times10^{-6}\omega - \frac{2}{\omega}\right)^2}$$

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

```
>> fzero('0.01-sqrt(1/50625+(0.6e-6*x-2./x).^2)',[1 1000])
ans =
  220.0202
```

6.20 The following script uses the `fzero` function can be used to determine the root as

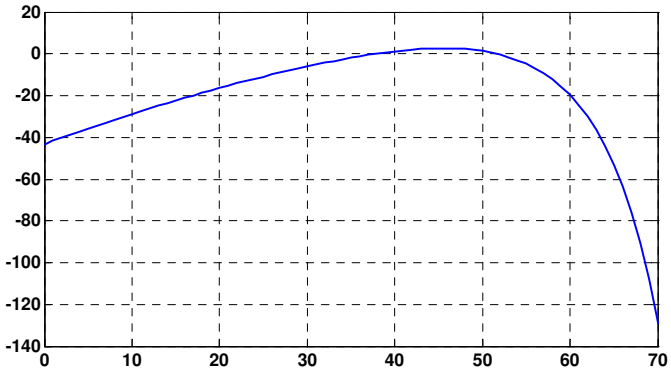
```
format long
k1=40000;k2=40;m=95;g=9.81;h=0.43;
fd=@(d) 2*k2*d^(5/2)/5+0.5*k1*d^2-m*g*d-m*g*h;
fzero(fd,1)

ans =
   0.166723562437785
```

6.21 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

$$f(\theta) = 0.8 + 90\tan\left(\frac{\pi}{180}\theta_0\right) - \frac{44.145}{\cos^2(\pi\theta_0/180)}$$

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

```
format long
g=9.81;v0=30;y0=1.8;
fth=@(th) 0.8+90*tan(pi*th/180)-44.1./cos(pi*th/180).^2;
thplot=[0:70];fplot=fth(thplot);
plot(thplot,fplot),grid
```



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

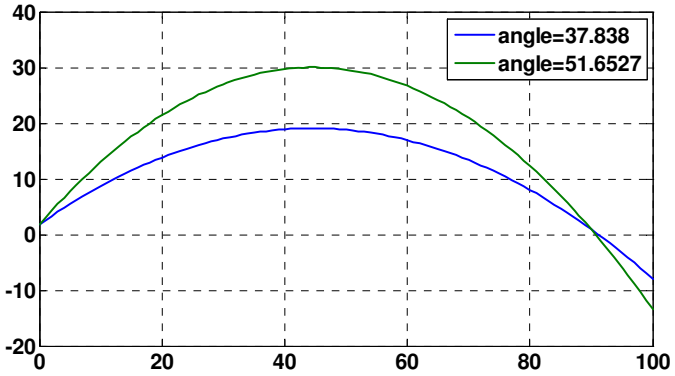
```
theta1 = fzero(fth,[30 45])
theta2 = fzero(fth,[45 60])
```

with the results

```
theta1 =
  37.837972746140331
theta2 =
  51.652744848882158
```

Therefore, two angles result in the desired outcome. We can develop plots of the two trajectories:

```
yth=@(x,th) tan(th*pi/180)*x-g/2/v0^2/cos(th*pi/180)^2*x.^2+y0;
xplot=[0:xdist];yplot=yth(xplot,theta1);yplot2=yth(xplot,theta2);
plot(xplot,yplot,xplot,yplot2,'--')
grid;legend(['angle=' num2str(theta1)],['angle=' num2str(theta2)])
```



Note that the lower angle would probably be preferred as the ball would arrive at the catcher sooner.

6.22 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,

$$x_{i+1} = x_i - \frac{\pi(3) x_i^2 - \left(\frac{\pi}{3}\right) x_i^3 - 30}{2\pi(3) x_i - \pi x_i^2}$$

The iterations can be summarized as

iteration	x_i	$f(x_i)$	$f'(x_i)$	$\lvert\varepsilon_a\rvert$
0	3	26.54867	28.27433	
1	2.061033	0.866921	25.50452	45.558%
2	2.027042	0.003449	25.30035	1.677%
3	2.026906	5.68E-08	25.29952	0.007%

Thus, after only three iterations, the root is determined to be 2.026906 with an approximate relative error of 0.007%.

6.23

```
>> format short g
>> r = [-2 -5 6 4 8];
>> a = poly(r)
a =
     1      -11      -12      356     -304    -1920
>> polyval(a,1)
ans =
       -1890
```

```
>> b = poly([-2 -5])
b =
     1     7    10
>> [q,r] = deconv(a,b)
q =
     1   -18   104  -192
r =
     0     0     0     0     0     0
>> x = roots(q)
x =
     8
     6
     4
>> a = conv(q,b)
a =
     1    -11  -12   356  -304  -1920
>> x = roots(a)
x =
     8
     6
     4
    -5
    -2
>> a = poly(x)
a =
        1      -11      -12      356     -304    -1920
```

6.24

```
>> 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

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

6.25 The equation can be rearranged so it can be solved with fixed-point iteration as

$$H_{i+1} = \frac{(Qn)^{3/5}(B+2H_i)^{2/5}}{BS^{3/10}}$$

Substituting the parameters gives,

$$H_{i+1} = 0.2062129(20+2H_i)^{2/5}$$

This formula can be applied iteratively to solve for H. For example, using and initial guess of $H_0 = 0$, the first iteration gives

$$H_1 = 0.2062129(20 + 2(0))^{2/5} = 0.683483$$

Subsequent iterations yield

i	H	ε_a
0	0.000000	
1	0.683483	100.000%
2	0.701799	2.610%
3	0.702280	0.068%
4	0.702293	0.002%

Thus, the process converges on a depth of 0.7023. We can prove that the scheme converges for all initial guesses greater than or equal to zero by differentiating the equation to give

$$g' = \frac{0.16497}{(20 + 2H)^{3/5}}$$

This function will always be less than one for $H \geq 0$. For example, if $H = 0$, $g' = 0.027339$. Because H is in the denominator, all values greater than zero yield even smaller values. Thus, the convergence criterion that $|g'| < 1$ always holds.

6.26 This problem can be solved in a number of ways. One approach involves using the modified secant method. This approach is feasible because the Swamee-Jain equation provides a sufficiently good initial guess that the method is always convergent for the specified parameter bounds. The following functions implement the approach:

```
function ffact = prob0626(eD,ReN)
% prob0626: friction factor with Colebrook equation
%   ffact = prob0626(eD,ReN):
%      uses modified secant equation to determine the friction factor
%      with the Colebrook equation
% input:
%   eD = e/D
%   ReN = Reynolds number
% output:
%   ffact = friction factor
maxit=100;es=1e-8;delta=1e-5;
iter = 0;
% Swamee-Jain equation:
xr = 1.325 / (log(eD / 3.7 + 5.74 / ReN ^ 0.9)) ^ 2;
% modified secant method
while (1)
  xrold = xr;
  xr = xr - delta*xr*func(xr,eD,ReN)/(func(xr+delta*xr,eD,ReN)...
                                          -func(xr,eD,ReN));
  iter = iter + 1;
  if xr ~= 0, ea = abs((xr - xrold)/xr) * 100; end
  if ea <= es | iter >= maxit, break, end
end
ffact = xr;

function ff=func(f,eD,ReN)
ff = 1/sqrt(f) + 2*log10(eD/3.7 + 2.51/ReN/sqrt(f));
```

Here are implementations for the extremes of the parameter range:

```
>> prob0626(0.00001,4000)
ans =
    0.0399
>> prob0626(0.05,4000)
ans =
    0.0770
>> prob0626(0.00001,1e7)
ans =
    0.0090
>> prob0626(0.05,1e7)
ans =
    0.0716
```

6.27 The Newton-Raphson method can be set up as

$$x_{i+1} = x_i - \frac{e^{-0.5x_i}(4 - x_i) - 2}{-e^{-0.5x_i}(3 - 0.5x_i)}$$

(a)

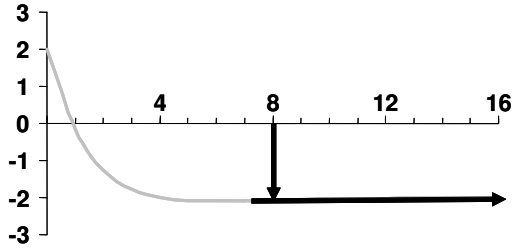
i	x	$f(x)$	$f'(x)$	$\lvert\varepsilon_a\rvert$
0	2	-1.26424	-0.73576	
1	0.281718	1.229743	-2.48348	609.93%
2	0.776887	0.18563	-1.77093	63.74%
3	0.881708	0.006579	-1.64678	11.89%
4	0.885703	9.13E-06	-1.64221	0.45%
5	0.885709	1.77E-11	-1.6422	0.00%
6	0.885709	0	-1.6422	0.00%

(b) The case does not work because the derivative is zero at $x_0 = 6$.

(c)

i	x	$f(x)$	$f'(x)$
0	8	-2.07326	0.018316
1	121.1963	-2	2.77E-25
2	7.21E+24	-2	0

This guess breaks down because, as depicted in the following plot, the near zero, positive slope sends the method away from the root.



6.28 The optimization problem involves determining the root of the derivative of the function. The derivative is the following function,

$$f'(x) = -12x^5 - 6x^3 + 10$$

The Newton-Raphson method is a good choice for this problem because

- The function is easy to differentiate
- It converges very rapidly

The Newton-Raphson method can be set up as

$$x_{i+1} = x_i - \frac{-12x_i^5 - 6x_i^3 + 10}{-60x_i^4 - 18x_i^2}$$

First iteration:

$$x_1 = x_0 - \frac{-12(1)^5 - 6(1)^3 + 10}{-60(1)^4 - 18(1)^2} = 0.897436$$

$$\varepsilon_a = \left|\frac{0.897436 - 1}{0.897436}\right| \times 100\% = 11.43\%$$

Second iteration:

$$x_1 = x_0 - \frac{-12(0.897436)^5 - 6(0.897436)^3 + 10}{-60(0.897436)^4 - 18(0.897436)^2} = 0.872682$$

$$\varepsilon_a = \left|\frac{0.872682 - 0.897436}{0.872682}\right| \times 100\% = 2.84\%$$

Since ε_a < 5%, the solution can be terminated.

6.29 Newton-Raphson is the best choice because:

- You know that the solution will converge. Thus, divergence is not an issue.
- Newton-Raphson is generally considered the fastest method
- You only require one guess
- The function is easily differentiable

To set up the Newton-Raphson first formulate the function as a roots problem and then differentiate it

$$f(x) = e^{0.5x} - 5 + 5x$$

$$f'(x) = 0.5e^{0.5x} + 5$$

These can be substituted into the Newton-Raphson formula

$$x_{i+1} = x_i - \frac{e^{0.5x_i} - 5 + 5x_i}{0.5e^{0.5x_i} + 5}$$

First iteration:

$$x_1 = 0.7 - \frac{e^{0.5(0.7)} - 5 + 5(0.7)}{0.5e^{0.5(0.7)} + 5} = 0.7 - \frac{-0.08093}{5.7095} = 0.714175$$

$$\varepsilon_a = \left|\frac{0.714175 - 0.7}{0.714175}\right| \times 100\% = 1.98\%$$

Therefore, only one iteration is required.

6.30 (a)

```
function [b,fb] = fzeronew(f,xl,xu,varargin)
% fzeronew: Brent root location zeroes
% [b,fb] = fzeronew(f,xl,xu,p1,p2,...):
%   uses Brent's method to find the root of f
% input:
%   f = name of function
%   xl, xu = lower and upper guesses
%   p1,p2,... = additional parameters used by f
% output:
%   b = real root
%   fb = function value at root
if nargin<3,error('at least 3 input arguments required'),end
a = xl; b = xu; fa = f(a,varargin{:}); fb = f(b,varargin{:});
c = a; fc = fa; d = b - c; e = d;
while (1)
  if fb == 0, break, end
  if sign(fa) == sign(fb)              %If necessary, rearrange points
    a = c; fa = fc; d = b - c; e = d;
  end
  if abs(fa) < abs(fb)
    c = b; b = a; a = c;
    fc = fb; fb = fa; fa = fc;
  end
  m = 0.5 * (a - b);      %Termination test and possible exit
  tol = 2 * eps * max(abs(b), 1);
  if abs(m) <= tol | fb == 0.
    break
  end
  %Choose open methods or bisection
  if abs(e) >= tol & abs(fc) > abs(fb)
    s = fb / fc;
    if a == c                              %Secant method
      p = 2 * m * s; q = 1 - s;
    else                     %Inverse quadratic interpolation
      q = fc / fa; r = fb / fa;
      p = s * (2 * m * q * (q - r) - (b - c) * (r - 1));
      q = (q - 1) * (r - 1) * (s - 1);
    end
    if p > 0, q = -q; else p = -p; end;
    if 2 * p < 3 * m * q - abs(tol * q) & p < abs(0.5 * e * q)
      e = d; d = p / q;
    else
      d = m; e = m;
    end
  else                                              %Bisection
    d = m; e = m;
  end
  c = b; fc = fb;
  if abs(d) > tol, b = b + d; else b = b - sign(b - a) * tol; end
  fb = f(b,varargin{:});
end
```

(b)

```
>> [x,fx] = fzeronew(@(x,n) x^n-1,0,1.3,10)
x =
     1
fx =
     0
```

CHAPTER 7

7.1 Equation (E7.1.1) can be differentiated to give

$$\frac{d^2z}{dt^2} = -\left(g + \frac{c}{m}v_0\right)e^{-(c/m)t}$$

Therefore, the Newton-Raphson method can be written as

$$t_{i+1} = t_i + \frac{v_0 e^{-(c/m)t} - \frac{mg}{c}\left(1 - e^{-(c/m)t}\right)}{\left(g + \frac{c}{m}v_0\right)e^{-(c/m)t}}$$

or simplifying

$$t_{i+1} = t_i + \frac{v_0 + \frac{mg}{c} - \frac{mg}{c}e^{(c/m)t}}{g + \frac{c}{m}v_0}$$

and substituting parameters

$$t_{i+1} = t_i + \frac{107.32 - 52.32e^{0.1875t_i}}{20.1225}$$

The first iteration gives

$$t_1 = 3 + \frac{107.32 - 52.32e^{0.1875(3)}}{20.1225} = 3.7706$$

After three iterations, the process converges on the correct result.

i	*t*	*f*	*f'*
0	3	8.829093	-11.4655
1	3.77006	0.607799	-9.92396
2	3.831306	0.003477	-9.81065
3	3.83166	1.15E-07	-9.81

Also notice how the derivative of the velocity (acceleration) converges on *g* as the velocity is driven to zero

7.2 (a) The function can be differentiated to give

$$f'(x) = -2x + 8$$

This function can be set equal to zero and solved for $x = 8/2 = 4$. The derivative can be differentiated to give the second derivative

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

Because this is negative, it indicates that the function has a maximum at $x = 4$.

(b) Using Eq. 7.10

$x_1 = 0 \quad f(x_1) = -12$
$x_2 = 2 \quad f(x_2) = 0$
$x_3 = 6 \quad f(x_3) = 0$

$$x_4 = 2 - \frac{1}{2}\frac{(2-0)^2[0-0]-(2-6)^2[0+12]}{(2-0)[0-0]-(2-6)[0+12]} = 4$$

7.3 Differentiating the function and setting the result equal to zero results in the following roots problem to locate the minimum

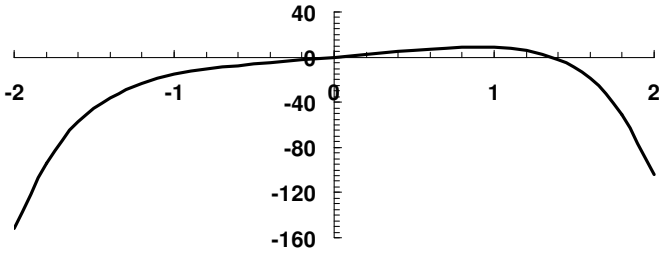
$$f'(x) = 6 + 10x + 9x^2 + 16x^3$$

Bisection can be employed to determine the root. Here are the first few iterations:

iteration	x_l	x_u	x_r	$f(x_l)$	$f(x_r)$	$f(x_l)\times f(x_r)$	ε_a
1	-2.0000	1.0000	-0.5000	-106.000	1.2500	-132.500	300.00%
2	-2.0000	-0.5000	-1.2500	-106.000	-23.6875	2510.875	60.00%
3	-1.2500	-0.5000	-0.8750	-23.6875	-6.5781	155.819	42.86%
4	-0.8750	-0.5000	-0.6875	-6.5781	-1.8203	11.974	27.27%
5	-0.6875	-0.5000	-0.5938	-1.8203	-0.1138	0.207	15.79%
6	-0.5938	-0.5000	-0.5469	-0.1138	0.6060	-0.0689	8.57%
7	-0.5938	-0.5469	-0.5703	-0.1138	0.2562	-0.0291	4.11%
8	-0.5938	-0.5703	-0.5820	-0.1138	0.0738	-0.0084	2.01%
9	-0.5938	-0.5820	-0.5879	-0.1138	-0.0193	0.0022	1.00%
10	-0.5879	-0.5820	-0.5850	-0.0193	0.0274	-0.0005	0.50%

The approach can be continued to yield a result of $x = -0.5867$.

7.4 (a) The function can be plotted



(b) The function can be differentiated twice to give

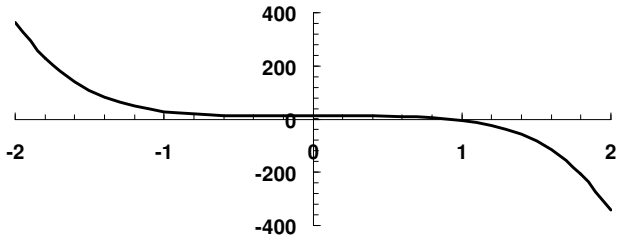
$$f''(x) = -45x^4 - 24x^2$$

Thus, the second derivative will always be negative and hence the function is concave for all values of x.

(c) Differentiating the function and setting the result equal to zero results in the following roots problem to locate the maximum

$$f'(x) = 0 = -9x^5 - 8x^3 + 12$$

A plot of this function can be developed



A technique such as bisection can be employed to determine the root. Here are the first few iterations:

iteration	x_l	x_u	x_r	$f(x_l)$	$f(x_r)$	$f(x_l)\times f(x_r)$	ε_a
1	0.00000	2.00000	1.00000	12	-5	-60.0000	
2	0.00000	1.00000	0.50000	12	10.71875	128.6250	100.00%
3	0.50000	1.00000	0.75000	10.71875	6.489258	69.5567	33.33%
4	0.75000	1.00000	0.87500	6.489258	2.024445	13.1371	14.29%
5	0.87500	1.00000	0.93750	2.024445	-1.10956	-2.2463	6.67%

The approach can be continued to yield a result of $x = 0.91692$.

7.5 First, the golden ratio can be used to create the interior points,

$$d = \frac{\sqrt{5}-1}{2}(2-0) = 1.2361$$

$$x_1 = 0 + 1.2361 = 1.2361 \qquad x_2 = 2 - 1.2361 = 0.7639$$

The function can be evaluated at the interior points

$$f(x_2) = f(0.7639) = 8.1879 \qquad f(x_1) = f(1.2361) = 4.8142$$

Because $f(x_2) > f(x_1)$, the maximum is in the interval defined by x_l, x_2, and x_1.where x_2 is the optimum. The error at this point can be computed as

$$\varepsilon_a = (1 - 0.61803)\left|\frac{2-0}{0.7639}\right| \times 100\% = 100\%$$

For the second iteration, $x_l = 0$ and $x_u = 1.2361$. The former x_2 value becomes the new x_1, that is, $x_1 = 0.7639$ and $f(x_1) = 8.1879$. The new values of d and x_2 can be computed as

$$d = \frac{\sqrt{5}-1}{2}(1.2361-0) = 0.7639 \qquad x_2 = 1.2361 - 0.7639 = 0.4721$$

The function evaluation at $f(x_2) = 5.5496$. Since this value is less than the function value at x_1, the maximum is in the interval prescribed by x_2, x_1 and x_u. The process can be repeated and all three iterations summarized as

i	x_l	$f(x_l)$	x_2	$f(x_2)$	x_1	$f(x_1)$	x_u	$f(x_u)$	d	x_{opt}	ε_a
1	0.0000	0.0000	0.7639	8.1879	1.2361	4.8142	2.0000	-104.0000	1.2361	0.7639	100.00%
2	0.0000	0.0000	0.4721	5.5496	0.7639	8.1879	1.2361	4.8142	0.7639	0.7639	61.80%
3	0.4721	5.5496	0.7639	8.1879	0.9443	8.6778	1.2361	4.8142	0.4721	0.9443	30.90%

The approach can be continued to yield a result of $x = 0.91692$.

7.6 First, the function values at the initial values can be evaluated

$f(x_1) = f(0) = 0$
$f(x_2) = f(1) = 8.5$
$f(x_3) = f(2) = -104$

and substituted into Eq. (7.10) to give,

$$x_4 = 1 - \frac{1}{2}\frac{(1-0)^2[8.5+104] - (1-2)^2[8.5-0)]}{(1-0)[8.5+104] - (1-2)[8.5-0)]} = 0.570248$$

which has a function value of $f(0.570248) = 6.5799$. Because the function value for the new point is lower than for the intermediate point (x_2) and the new x value is to the left of the intermediate point, the lower guess (x_1) is discarded. Therefore, for the next iteration,

$f(x_1) = f(0.570248) = 6.6799$
$f(x_2) = f(1) = 8.5$
$f(x_3) = f(2) = -104$

which can be substituted into Eq. (7.10) to give $x_4 = 0.812431$, which has a function value of $f(0.812431) = 8.446523$. At this point, an approximate error can be computed as

$$\varepsilon_a = \left|\frac{0.81243 - 0.570248}{0.81243}\right| \times 100\% = 29.81\%$$

The process can be repeated, with the results tabulated below:

i	x_1	$f(x_1)$	x_2	$f(x_2)$	x_3	$f(x_3)$	x_4	$f(x_4)$	ε_a
1	0.00000	0.00000	1.00000	8.50000	2.0000	-104	0.57025	6.57991	
2	0.57025	6.57991	1.00000	8.50000	2.0000	-104	0.81243	8.44652	29.81%
3	0.81243	8.44652	1.00000	8.50000	2.0000	-104	0.90772	8.69575	10.50%

Thus, after 3 iterations, the result is converging on the true value of $f(x) = 8.69793$ at $x = 0.91692$.

7.7 (a) First, the golden ratio can be used to create the interior points,

$$d = \frac{\sqrt{5}-1}{2}(4-(-2)) = 3.7082$$
$$x_1 = -2 + 3.7082 = 1.7082$$
$$x_2 = 4 - 3.7082 = 0.2918$$

The function can be evaluated at the interior points

$$f(x_2) = f(0.2918) = 1.04156$$
$$f(x_1) = f(1.7082) = 5.00750$$

Because $f(x_1) > f(x_2)$, the maximum is in the interval defined by x_2, x_1 and x_u where x_1 is the optimum. The error at this point can be computed as

$$\varepsilon_a = (1 - 0.61803)\left|\frac{4-(-2)}{1.7082}\right| \times 100\% = 134.16\%$$

The process can be repeated and all the iterations summarized as

i	x_l	$f(x_l)$	x_2	$f(x_2)$	x_1	$f(x_1)$	x_u	$f(x_u)$	d	x_{opt}	ε_a
1	-2.0000	-29.6000	0.2918	1.0416	1.7082	5.0075	4.0000	-12.8000	3.7082	1.7082	134.16%
2	0.2918	1.0416	1.7082	5.0075	2.5836	5.6474	4.0000	-12.8000	2.2918	2.5836	54.82%
3	1.7082	5.0075	2.5836	5.6474	3.1246	2.9361	4.0000	-12.8000	1.4164	2.5836	33.88%
4	1.7082	5.0075	2.2492	5.8672	2.5836	5.6474	3.1246	2.9361	0.8754	2.2492	24.05%
5	1.7082	5.0075	2.0426	5.6648	2.2492	5.8672	2.5836	5.6474	0.5410	2.2492	14.87%
6	2.0426	5.6648	2.2492	5.8672	2.3769	5.8770	2.5836	5.6474	0.3344	2.3769	8.69%
7	2.2492	5.8672	2.3769	5.8770	2.4559	5.8287	2.5836	5.6474	0.2067	2.3769	5.37%
8	2.2492	5.8672	2.3282	5.8853	2.3769	5.8770	2.4559	5.8287	0.1277	2.3282	3.39%
9	2.2492	5.8672	2.2980	5.8828	2.3282	5.8853	2.3769	5.8770	0.0789	2.3282	2.10%
10	2.2980	5.8828	2.3282	5.8853	2.3468	5.8840	2.3769	5.8770	0.0488	2.3282	1.30%
11	2.2980	5.8828	2.3166	5.8850	2.3282	5.8853	2.3468	5.8840	0.0301	2.3282	0.80%

(b) First, the function values at the initial values can be evaluated

$f(x_1) = f(1.75) = 5.1051$
$f(x_2) = f(2) = 5.6$
$f(x_3) = f(2.5) = 5.7813$

and substituted into Eq. (7.10) to give $x_4 = 2.3341$, which has a function value of $f(2.3341) = 5.8852$. Because the function value for the new point is higher than for the intermediate point (x_2) and the new x value is to the right of the intermediate point, the lower guess (x_1) is discarded. Therefore, for the next iteration,

$f(x_1) = f(2) = 5.6$
$f(x_2) = f(2.3341) = 5.8852$
$f(x_3) = f(2.5) = 5.7813$

which can be substituted into Eq. (7.10) to give $x_4 = 2.3112$, which has a function value of $f(2.3112) = 5.8846$. At this point, an approximate error can be computed as

$$\varepsilon_a = \left|\frac{2.3112 - 2.3341}{2.3112}\right| \times 100\% = 0.99\%$$

The process can be repeated, with the results tabulated below:

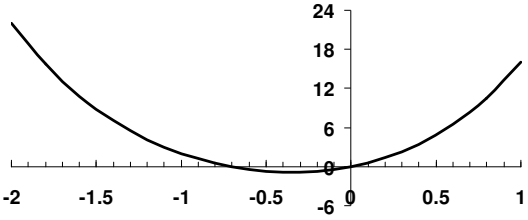
i	x_1	$f(x_1)$	x_2	$f(x_2)$	x_3	$f(x_3)$	x_4	$f(x_4)$	ε_a
1	1.7500	5.1051	2.0000	5.6000	2.5000	5.7813	2.3341	5.8852	
2	2.0000	5.6000	2.3341	5.8852	2.5000	5.7813	2.3112	5.8846	0.99%
3	2.0000	5.6000	2.3112	5.8846	2.3341	5.8852	2.3270	5.8853	0.68%
4	2.3112	5.8846	2.3270	5.8853	2.3341	5.8852	2.3263	5.8853	0.03%
5	2.3112	5.8846	2.3263	5.8853	2.3270	5.8853	2.3264	5.8853	0.00%

Thus, after 5 iterations, the result is converging rapidly on the true value of $f(x) = 5.8853$ at $x = 2.3264$.

7.8 The function can be differentiated twice to give

$$f'(x) = 4x^3 + 6x^2 + 16x = 5$$
$$f''(x) = 12x^2 + 12x + 16$$

which is positive for $-2 \le x \le 1$. This suggests that an optimum in the interval would be a minimum. A graph of the original function shows a maximum at about $x = -0.35$.



7.9 (a) First, the golden ratio can be used to create the interior points,

$$d = \frac{\sqrt{5}-1}{2}(1-(-2)) = 1.8541$$
$$x_1 = -2 + 1.8541 = -0.1459$$
$$x_2 = 1 - 1.8541 = -0.8541$$

The function can be evaluated at the interior points

$$f(x_2) = f(-0.8541) = 0.8514$$
$$f(x_1) = f(-0.1459) = -0.5650$$

Because $f(x_1) < f(x_2)$, the minimum is in the interval defined by x_2, x_1 and x_u where x_2 is the optimum. The error at this point can be computed as

$$\varepsilon_a = (1-0.61803)\left|\frac{1-(-2)}{-0.1459}\right| \times 100\% = 785.41\%$$

The process can be repeated and all the iterations summarized as

i	x_l	$f(x_l)$	x_2	$f(x_2)$	x_1	$f(x_1)$	x_u	$f(x_u)$	d	x_{opt}	ε_a
1	-2	22	-0.8541	0.851	-0.1459	-0.565	1	16.000	1.8541	-0.1459	785.41%
2	-0.8541	0.851	-0.1459	-0.565	0.2918	2.197	1	16.000	1.1459	-0.1459	485.41%
3	-0.8541	0.851	-0.4164	-0.809	-0.1459	-0.565	0	2.197	0.7082	-0.4164	105.11%
4	-0.8541	0.851	-0.5836	-0.475	-0.4164	-0.809	0	-0.565	0.4377	-0.4164	64.96%
5	-0.5836	-0.475	-0.4164	-0.809	-0.3131	-0.833	0	-0.565	0.2705	-0.3131	53.40%
6	-0.4164	-0.809	-0.3131	-0.833	-0.2492	-0.776	0	-0.565	0.1672	-0.3131	33.00%
7	-0.4164	-0.809	-0.3525	-0.841	-0.3131	-0.833	0	-0.776	0.1033	-0.3525	18.11%
8	-0.4164	-0.809	-0.3769	-0.835	-0.3525	-0.841	0	-0.833	0.0639	-0.3525	11.19%
9	-0.3769	-0.835	-0.3525	-0.841	-0.3375	-0.840	0	-0.833	0.0395	-0.3525	6.92%
10	-0.3769	-0.835	-0.3619	-0.839	-0.3525	-0.841	0	-0.840	0.0244	-0.3525	4.28%
11	-0.3619	-0.839	-0.3525	-0.841	-0.3468	-0.841	0	-0.840	0.0151	-0.3468	2.69%
12	-0.3525	-0.841	-0.3468	-0.841	-0.3432	-0.841	0	-0.840	0.0093	-0.3468	1.66%
13	-0.3525	-0.841	-0.3490	-0.841	-0.3468	-0.841	0	-0.841	0.0058	-0.3468	1.03%
14	-0.3490	-0.841	-0.3468	-0.841	-0.3454	-0.841	0	-0.841	0.0036	-0.3468	0.63%

(b) First, the function values at the initial values can be evaluated

$f(x_1) = f(-2) = 22$
$f(x_2) = f(-1) = 2$
$f(x_3) = f(1) = 16$

and substituted into Eq. (7.10) to give $x_4 = -0.38889$, which has a function value of $f(-0.38889) = -0.829323$. Because the function value for the new point is lower than for the intermediate point (x_2) and the new x value is to the right of the intermediate point, the lower guess (x_1) is discarded. Therefore, for the next iteration,

$f(x_1) = f(-1) = 2$
$f(x_2) = f(-0.38889) = -0.829323$
$f(x_3) = f(1) = 16$

which can be substituted into Eq. (7.10) to give $x_4 = -0.41799$, which has a function value equal to $f(-0.41799) = -0.80776$. At this point, an approximate error can be computed as

$$\varepsilon_a = \left|\frac{-0.41799 - (-0.38889)}{-0.41799}\right| \times 100\% = 6.96\%$$

The process can be repeated, with the results tabulated below:

i	x_1	$f(x_1)$	x_2	$f(x_2)$	x_3	$f(x_3)$	x_4	$f(x_4)$	ε_a
1	-2	22	-1.0000	2	1.0000	16	-0.38889	-0.82932	
2	-1.0000	2	-0.3889	-0.82932	1.0000	16	-0.41799	-0.80776	6.96%
3	-0.4180	-0.80776	-0.3889	-0.82932	1.0000	16	-0.36258	-0.83924	15.28%
4	-0.3889	-0.82932	-0.3626	-0.839236	1.0000	16	-0.35519	-0.84038	2.08%
5	-0.3626	-0.83924	-0.3552	-0.840376	1.0000	16	-0.35053	-0.84072	1.33%

After 5 iterations, the result is converging on the true value of $f(x) = -0.8408$ at $x = -0.34725$.

7.10 First, the function values at the initial values can be evaluated

$f(x_1) = f(0.1) = 30.2$
$f(x_2) = f(0.5) = 7$
$f(x_3) = f(5) = 10.6$

and substituted into Eq. (7.10) to give $x_4 = 2.7167$, which has a function value of $f(2.7167) = 6.5376$. Because the function value for the new point is lower than for the intermediate point (x_2) and the new x value is to the right of the intermediate point, the lower guess (x_1) is discarded. Therefore, for the next iteration,

$f(x_1) = f(0.5) = 7$
$f(x_2) = f(2.7167) = 6.5376$
$f(x_3) = f(5) = 10.6$

which can be substituted into Eq. (7.10) to give $x_4 = 1.8444$, which has a function value of $f(1.8444) = 5.3154$. At this point, an approximate error can be computed as

$$\varepsilon_a = \left|\frac{1.8444 - 2.7167}{1.8444}\right| \times 100\% = 47.29\%$$

The process can be repeated, with the results tabulated below:

i	x_1	$f(x_1)$	x_2	$f(x_2)$	x_3	$f(x_3)$	x_4	$f(x_4)$	ε_a
1	0.1	30.2	0.5000	7.0000	5.0000	10.6000	2.7167	6.5376	
2	0.5000	7	2.7167	6.5376	5.0000	10.6000	1.8444	5.3154	47.29%
3	0.5000	7	1.8444	5.3154	2.7167	6.5376	1.6954	5.1603	8.79%
4	0.5000	7	1.6954	5.1603	1.8444	5.3154	1.4987	4.9992	13.12%

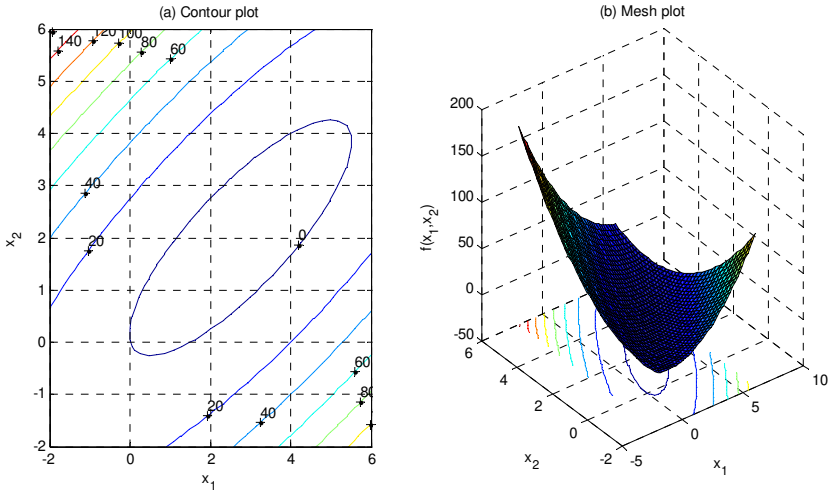
5	0.5000	7	1.4987	4.9992	1.6954	5.1603	1.4236	4.9545	5.28%
6	0.5000	7	1.4236	4.9545	1.4987	4.9992	1.3556	4.9242	5.02%
7	0.5000	7	1.3556	4.9242	1.4236	4.9545	1.3179	4.9122	2.85%
8	0.5000	7	1.3179	4.9122	1.3556	4.9242	1.2890	4.9054	2.25%
9	0.5000	7	1.2890	4.9054	1.3179	4.9122	1.2703	4.9023	1.47%
10	0.5000	7	1.2703	4.9023	1.2890	4.9054	1.2568	4.9006	1.08%

Thus, after 10 iterations, the result is converging very slowly on the true value of $f(x) = 4.8990$ at $x = 1.2247$.

7.11 The script can be written as

```
x=linspace(-2,6,40);y=linspace(-2,6,40);
[X,Y] = meshgrid(x,y);
Z=2*X.^2+3*Y.^2-4*X.*Y-Y-3*X;
subplot(1,2,1);
cs=contour(X,Y,Z);clabel(cs);
xlabel('x_1');ylabel('x_2');
title('(a) Contour plot');grid;
subplot(1,2,2);
cs=surfc(X,Y,Z);
zmin=floor(min(Z));
zmax=ceil(max(Z));
xlabel('x_1');ylabel('x_2');zlabel('f(x_1,x_2)');
title('(b) Mesh plot');
pause
f=@(x) 2*x(1)^2+3*x(2)^2-4*x(1)*x(2)-x(2)-3*x(1);
[x,fval]=fminsearch(f,[4,4])
```

When it is run, the following plot and solution are generated:



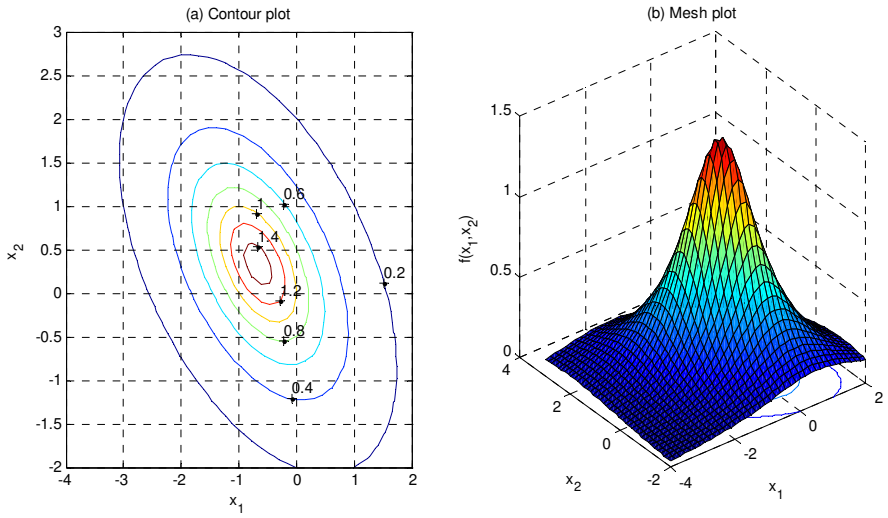

```
x =
    2.7500    2.0000
fval =
   -5.1250
```

7.12 The script can be written as

```
x=linspace(-4,2,40);y=linspace(-2,3,40);
[X,Y] = meshgrid(x,y);
Z=1./(1+X.^2+Y.^2+X+X.*Y);
```

```
subplot(1,2,1);
cs=contour(X,Y,Z);clabel(cs);
xlabel('x_1');ylabel('x_2');
title('(a) Contour plot');grid;
subplot(1,2,2);
cs=surfc(X,Y,Z);
zmin=floor(min(Z));
zmax=ceil(max(Z));
xlabel('x_1');ylabel('x_2');zlabel('f(x_1,x_2)');
title('(b) Mesh plot');
pause
f=@(x) -1/(1+x(1).^2+x(2).^2+x(1)+x(1)*x(2));
[x,fval]=fminsearch(f,[4,4])
```

Notice that because we are looking for a maximum, we enter the negative of the function for `fminsearch`. When it is run, the following plot and solution are generated:




```
x =
   -0.6667    0.3333
fval =
   -1.5000
```

Thus, the solution is 1.5 at location (–0.6667, 0.3333).

7.13 This problem can be approached by developing the following equation for the potential energy of the bracketing system,

$$V(x,y) = \frac{EA}{\ell}\left(\frac{w}{2\ell}\right)^2 x^2 + \frac{EA}{\ell}\left(\frac{h}{\ell}\right)^2 y^2 - Fx\cos\theta - Fy\sin\theta$$

Solving for an individual angle is straightforward. For $\theta = 30^o$, the given parameter values can be substituted to yield

$$V(x, y) = 5{,}512{,}026x^2 + 28{,}471{,}210y^2 - 8{,}600x - 5{,}000y$$

The minimum of this function can be determined in a number of ways. For example, a function can be set up to compute the potential energy

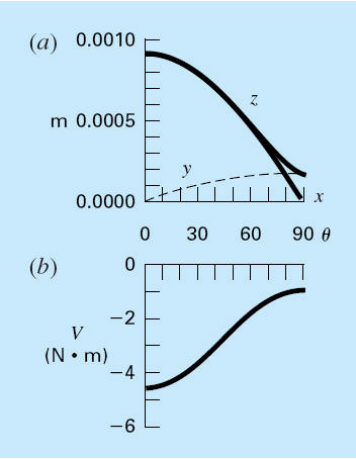
```
function Vxy = prob0713f(x,th)
E=2e11;A=0.0001;w=0.44;l=0.56;h=0.5;F=10000;
Vxy=E*A/l*(w/2/l)^2*x(1)^2+E*A/l*(h/l)^2*x(2)^2-F*x(1)*cos(th)-F*x(2)*sin(th);
```

Then, the `fminsearch` function can be used to evaluate the minimum potential energy at a particular angle. For the following case, we use $\theta = 30^o$,

```
>> format short g
>> th=30*pi/180;
>> options = optimset('TolFun',1e-6,'TolX',1e-6);
>> [x fxy] = fminsearch(@prob0713f,[0 0],options,th)

x =
   0.00078585  8.7771e-005
fxy =
      -3.6212
```

Thus, the minimum potential energy is −3.6212 with deflections of $x = 0.00078585$ and $y = 0.000087771$ m.

Of course, this calculation could be implemented repeatedly for different values of θ to assess how the solution changes as the angle changes. If this is done, the results are displayed in the following figure:



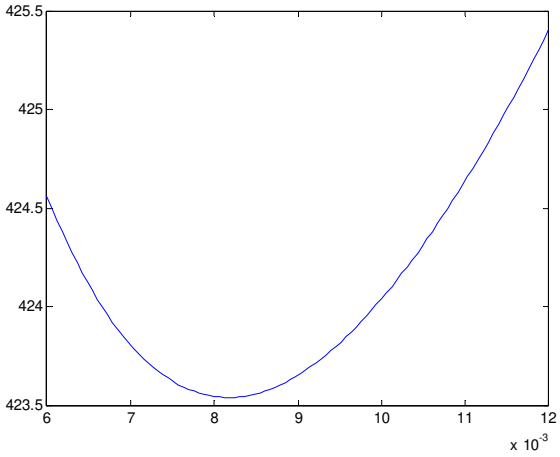
(*a*) The impact of different angles on the deflections (note that *z* is the resultant of the *x* and *y* components) and (*b*) potential energy.

As might be expected, the x deflection is at a maximum when the load is pointed in the x direction ($\theta = 0^o$) and the y deflection is at a maximum when the load is pointed in the y direction ($\theta = 90^o$). However, notice that the x deflection is much more pronounced than that in the y direction. This is also manifest in part (*b*), where the potential energy is higher at low angles. Both results are due to the geometry of the strut. If w were made bigger, the deflections would be more uniform.

7.14 The following script generates a plot of wire temperature versus insulation thickness. Then, it employs `fminbnd` to locate the thickness of insulation that minimizes the wire's temperature.

```
clear, clc, clf, format short g
q=75;rw=6e-3;k=0.17;h=12;Tair=293;
rplot=linspace(rw,rw*2);
T=@(ri) Tair+q/(2*pi)*(1/k*log((rw+ri)/rw)+1/h*1./(rw+ri));
Tplot=T(rplot);
plot(rplot,Tplot)
```

```
[rimin Tmin]=fminbnd(T,rw,2*rw)
rtotal=rimin+rw
```




```
rimin =
    0.0081725
Tmin =
       423.54
rtotal =
     0.014173
```

Thus, an insulation thickness of 8.1725 mm yields a minimum wire temperature of 423.54 K. The total radius of the wire and insulation is 14.1725 mm.

7.15 The algorithm from Fig. 7.7 can be converted to locate a maximum by merely changing the sense of the `if` statement highlighted below:

```
function [x,fx,ea,iter]=goldmax(f,xl,xu,es,maxit,varargin)
% goldmax: maximization golden section search
%   [xopt,fopt,ea,iter]=goldmax(f,xl,xu,es,maxit,p1,p2,...):
%      uses golden section search to find the maximum of f
% input:
%   f = name of function
%   xl, xu = lower and upper guesses
%   es = desired relative error (default = 0.0001%)
%   maxit = maximum allowable iterations (default = 50)
%   p1,p2,... = additional parameters used by f
% output:
%   x = location of maximum
%   fx = maximum function value
%   ea = approximate relative error (%)
%   iter = number of iterations

if nargin<3,error('at least 3 input arguments required'),end
if nargin<4|isempty(es), es=0.0001;end
if nargin<5|isempty(maxit), maxit=50;end
phi=(1+sqrt(5))/2;
iter=0;
while(1)
  d = (phi-1)*(xu - xl);
  x1 = xl + d;
  x2 = xu - d;
  if f(x1,varargin{:}) > f(x2,varargin{:})
    xopt = x1;
    xl = x2;
```

```
  else
    xopt = x2;
    xu = x1;
  end
  iter=iter+1;
  if xopt~=0, ea = (2 - phi) * abs((xu - xl) / xopt) * 100;end
  if ea <= es | iter >= maxit,break,end
end
x=xopt;fx=f(xopt,varargin{:});
```

An application to solve Example 7.1 can be developed as

```
>> g=9.81;z0=100;v0=55;m=80;c=15;
>> z=@(t) z0+m/c*(v0+m*g/c)*(1-exp(-c/m*t))-m*g/c*t;
>> [xmax,fmax,ea,iter]=goldmax(z,0,8)

xmax =
    3.8317
fmax =
  192.8609
ea =
  6.9356e-005
iter =
    29
```

7.16 Each iteration of the golden-section search reduces the interval by a factor of $1/\phi$. Hence, if we start with an interval, $E_a^0 = \Delta x_0 = x_u - x_l$, the interval after the nth iteration will be

$$E_a^n = \frac{\Delta x^0}{\phi^n}$$

Each iteration of the golden-section search reduces the interval by a factor of $1/\phi$. Hence, if we start with an interval, $\Delta x^0 = x_u - x_l$, the interval after the nth iteration will be

$$\Delta x^n = \frac{\Delta x^0}{\phi^n}$$

In addition, on p. 191, we illustrate that the maximum error is $(2 - \phi)$ times the interval width. If we define the desired error as $E_{a,d}$,

$$E_{a,d} = (2-\phi)\frac{\Delta x^0}{\phi^n}$$

This equation can be solved for

$$n = \frac{\log_2\left[(2-\varphi)\dfrac{\Delta x^0}{E_{a,d}}\right]}{\log_2 \varphi}$$

or through further simplification,

$$n = \frac{\log_2\left(\frac{\Delta x^0}{E_{a,d}}\right)}{\log_2 \varphi} - 2$$

The following function implements n iterations of the golden section search:

```
function [x,fx]=prob0716(f,xl,xu,Ead,varargin)
% prob0716: minimization golden section search
%   [x,fx]=prob0716(f,xl,xu,Ead,p1,p2,...):
%      uses golden section search to find the minimum of f
%      within a prescribed tolerance
% input:
%   f = name of function
%   xl, xu = lower and upper guesses
%   Ead = desired absolute error (default = 0.000001)
%   p1,p2,... = additional parameters used by f
% output:
%   x = location of minimum
%   fx = minimum function value

if nargin<3,error('at least 3 input arguments required'),end
if nargin<4|isempty(Ead), Ead=0.000001;end
phi=(1+sqrt(5))/2;
n=ceil(log2((xu-xl)/Ead)/log2(phi)-2);
if n<1,n=1;end
for i = 1:n
  d = (phi-1)*(xu - xl);
  x1 = xl + d; x2 = xu - d;
  if f(x1,varargin{:}) < f(x2,varargin{:})
    xopt = x1;
    xl = x2;
  else
    xopt = x2;
    xu = x1;
  end
end
x=xopt;fx=f(xopt,varargin{:});
```

Here is a session that uses this function to solve the minimization from Example 7.2.

```
>> format long
>> f=@(x) x^2/10-2*sin(x);
>> [x,fx]=prob0716(f,0,4,0.0001)

x =
   1.42752749558362
fx =
  -1.77572565250482
```

7.17

```
function [xopt,fopt]=prob0713(func,xlow,xhigh,es,maxit,varargin)
% prob0717: minimization parabolic interpolation
%   [xopt,fopt]=prob0717(func,xlow,xhigh,es,maxit,p1,p2,...):
%      uses parabolic interpolation to find the minimum of f
% input:
%   f = name of function
%   xl, xu = lower and upper guesses
%   es = desired relative error (default = 0.0001%)
%   maxit = maximum allowable iterations (default = 50)
```

```
%   p1,p2,... = additional parameters used by f
% output:
%   xopt = location of minimum
%   fopt = minimum function value

if nargin<3,error('at least 3 input arguments required'),end
if nargin<4|isempty(es), es=0.0001;end
if nargin<5|isempty(maxit), maxit=50;end
iter = 0;
x1 = xlow;   x3 = xhigh;
x2 = (x1 + x3) / 2;
f1 = func(x1,varargin{:});
f2 = func(x2,varargin{:});
f3 = func(x3,varargin{:});
if f2<f1 & f2<f3
  xoptold = x2;
  while(1)
    xopt=x2-0.5*((x2-x1)^2*(f2-f3)-(x2-x3)^2*(f2-f1))/((x2-x1)...
                                              *(f2-f3)-(x2-x3)*(f2-f1));
    fopt = func(xopt,varargin{:});
    iter = iter + 1;
    if xopt > x2
      x1 = x2;
      f1 = f2;
    else
      x3 = x2;
      f3 = f2;
    end
    x2 = xopt; f2 = fopt;
    if xopt~=0,ea=abs((xopt - xoptold) / xopt) * 100;end
    xoptold = xopt;
    if ea<=es | iter>=maxit,break,end
  end
else
  error('bracket does not contain minimum')
end

>> f=@(x) x^2/10-2*sin(x);
>> [x,fx]=prob0717(f,0,4)
x =
    1.4276
fx =
   -1.7757
```

7.18 The first iteration of the golden-section search can be implemented as

$$d = \frac{\sqrt{5}-1}{2}(4-2) = 1.2361$$
$$x_1 = 2 + 1.2361 = 3.2361$$
$$x_2 = 4 - 1.2361 = 2.7639$$
$$f(x_2) = f(2.7639) = -6.1303$$
$$f(x_1) = f(3.2361) = -5.8317$$

Because $f(x_2) < f(x_1)$, the minimum is in the interval defined by x_l, x_2, and x_1 where x_2 is the optimum. The error at this point can be computed as

$$\varepsilon_a = (1 - 0.61803)\left|\frac{4-2}{2.7639}\right| \times 100\% = 27.64\%$$

The process can be repeated and all the iterations summarized as

i	x_l	$f(x_l)$	x_2	$f(x_2)$	x_1	$f(x_1)$	x_u	$f(x_u)$	d	x_{opt}	ε_a
1	2	-3.8608	2.7639	-6.1303	3.2361	-5.8317	4	-2.7867	1.2361	2.7639	27.64%
2	2	-3.8608	2.4721	-5.6358	2.7639	-6.1303	3.2361	-5.8317	0.7639	2.7639	17.08%
3	2.4721	-5.6358	2.7639	-6.1303	2.9443	-6.1776	3.2361	-5.8317	0.4721	2.9443	9.91%
4	2.7639	-6.1303	2.9443	-6.1776	3.0557	-6.1065	3.2361	-5.8317	0.2918	2.9443	6.13%

After four iterations, the process is converging on the true minimum at $x = 2.8966$ where the function has a value of $f(x) = -6.1847$.

7.19 The first iteration of the golden-section search can be implemented as

$$d = \frac{\sqrt{5}-1}{2}(60 - 0) = 37.0820$$
$$x_1 = 0 + 37.0820 = 37.0820$$
$$x_2 = 60 - 37.0820 = 22.9180$$
$$f(x_2) = f(22.9180) = 18.336$$
$$f(x_1) = f(37.0820) = 19.074$$

Because $f(x_1) > f(x_2)$, the maximum is in the interval defined by x_2, x_1, and x_u where x_1 is the optimum. The error at this point can be computed as

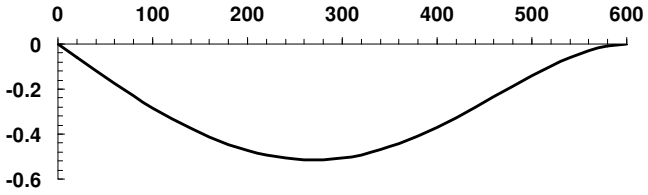
$$\varepsilon_a = (1 - 0.61803)\left|\frac{60-0}{37.0820}\right| \times 100\% = 61.80\%$$

The process can be repeated and all the iterations summarized as

i	x_l	$f(x_l)$	x_2	$f(x_2)$	x_1	$f(x_1)$	x_u	$f(x_u)$	d	x_{opt}	ε_a
1	0	1	22.9180	18.336	37.0820	19.074	60	4.126	37.0820	37.0820	61.80%
2	22.9180	18.336	37.0820	19.074	45.8359	15.719	60	4.126	22.9180	37.0820	38.20%
3	22.9180	18.336	31.6718	19.692	37.0820	19.074	45.8359	15.719	14.1641	31.6718	27.64%
4	22.9180	18.336	28.3282	19.518	31.6718	19.692	37.0820	19.074	8.7539	31.6718	17.08%
5	28.3282	19.518	31.6718	19.692	33.7384	19.587	37.0820	19.074	5.4102	31.6718	10.56%
6	28.3282	19.518	30.3947	19.675	31.6718	19.692	33.7384	19.587	3.3437	31.6718	6.52%
7	30.3947	19.675	31.6718	19.692	32.4612	19.671	33.7384	19.587	2.0665	31.6718	4.03%
8	30.3947	19.675	31.1840	19.693	31.6718	19.692	32.4612	19.671	1.2772	31.1840	2.53%
9	30.3947	19.675	30.8825	19.689	31.1840	19.693	31.6718	19.692	0.7893	31.1840	1.56%
10	30.8825	19.689	31.1840	19.693	31.3703	19.693	31.6718	19.692	0.4878	31.3703	0.96%

After ten iterations, the process falls below the stopping criterion and the result is converging on the true maximum at $x = 31.3713$ where the function has a value of $y = 19.6934$.

7.20 (a) A graph indicates the minimum at about $x = 270$.



(b) Golden section search,

$$d = \frac{\sqrt{5}-1}{2}(600-0) = 370.820$$
$$x_1 = 0 + 370.820 = 370.820$$
$$x_2 = 600 - 370.820 = 229.180$$
$$f(x_2) = f(229.180) = -0.5016$$
$$f(x_1) = f(370.820) = -0.4249$$

Because $f(x_2) < f(x_1)$, the minimum is in the interval defined by x_l, x_2, and x_1 where x_2 is the optimum. The error at this point can be computed as

$$\varepsilon_a = (1-0.61803)\left|\frac{600-0}{370.820}\right| \times 100\% = 100\%$$

The process can be repeated and all the iterations summarized as

i	x_l	$f(x_l)$	x_2	$f(x_2)$	x_1	$f(x_1)$	x_u	$f(x_u)$	d	x_{opt}	ε_a
1	0	0	229.180	-0.5016	370.820	-0.4249	600	0	370.820	229.1796	100.00%
2	0	0	141.641	-0.3789	229.180	-0.5016	370.820	-0.4249	229.180	229.1796	61.80%
3	141.641	-0.3789	229.180	-0.5016	283.282	-0.5132	370.820	-0.4249	141.641	283.2816	30.90%
4	229.180	-0.5016	283.282	-0.5132	316.718	-0.4944	370.820	-0.4249	87.539	283.2816	19.10%
5	229.180	-0.5016	262.616	-0.5149	283.282	-0.5132	316.718	-0.4944	54.102	262.6165	12.73%
6	229.180	-0.5016	249.845	-0.5121	262.616	-0.5149	283.282	-0.5132	33.437	262.6165	7.87%
7	249.845	-0.5121	262.616	-0.5149	270.510	-0.5151	283.282	-0.5132	20.665	270.5098	4.72%
8	262.616	-0.5149	270.510	-0.5151	275.388	-0.5147	283.282	-0.5132	12.772	270.5098	2.92%
9	262.616	-0.5149	267.495	-0.5152	270.510	-0.5151	275.388	-0.5147	7.893	267.4948	1.82%
10	262.616	-0.5149	265.631	-0.5151	267.495	-0.5152	270.510	-0.5151	4.878	267.4948	1.13%
11	265.631	-0.5151	267.495	-0.5152	268.646	-0.5152	270.510	-0.5151	3.015	268.6465	0.69%

After eleven iterations, the process falls below the stopping criterion and the result is converging on the true minimum at $x = 268.3281$ where the function has a value of $y = -0.51519$.

7.21 The velocity of a falling object with an initial velocity and first-order drag can be computed as

$$v = v_0 e^{-(c/m)t} + \frac{mg}{c}\left(1 - e^{-(c/m)t}\right)$$

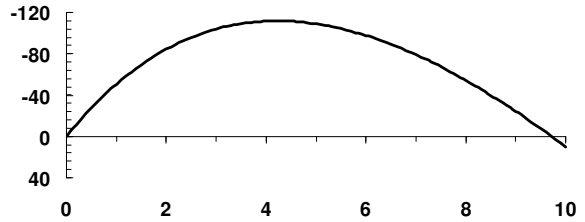
The vertical distance traveled can be determined by integration

$$z = z_0 + \int_0^t v_0 e^{-(c/m)t} + \frac{mg}{c}\left(1 - e^{-(c/m)t}\right) dt$$

where negative z is distance upwards. Assuming that $z_0 = 0$, evaluating the integral yields

$$z = \frac{mg}{c}t + \frac{m}{c}\left(v_0 - \frac{mg}{c}\right)\left(1 - e^{-(c/m)t}\right)$$

Therefore the solution to this problem amounts to determining the minimum of this function (since the most negative value of z corresponds to the maximum height). This function can be plotted using the given parameter values. As in the following graph (note that the ordinate values are plotted in reverse), the maximum height occurs after about 4.2 s and appears to be about 110 m.



Here is the result of using the golden-section search to determine the maximum height

i	x_l	$f(x_l)$	x_2	$f(x_2)$	x_1	$f(x_1)$	x_u	$f(x_u)$	d	x_{opt}	ε_a
1	0	0	1.9098	-82.008	3.0902	-105.171	5	-108.922	3.0902	3.0902	61.80%
2	1.9098	-82.008	3.0902	-105.171	3.8197	-111.014	5	-108.922	1.9098	3.8197	30.90%
3	3.0902	-105.171	3.8197	-111.014	4.2705	-111.790	5.0000	-108.922	1.1803	4.2705	17.08%
4	3.8197	-111.014	4.2705	-111.790	4.5492	-111.272	5.0000	-108.922	0.7295	4.2705	10.56%
5	3.8197	-111.014	4.0983	-111.735	4.2705	-111.790	4.5492	-111.272	0.4508	4.2705	6.52%
6	4.0983	-111.735	4.2705	-111.790	4.3769	-111.680	4.5492	-111.272	0.2786	4.2705	4.03%
7	4.0983	-111.735	4.2047	-111.804	4.2705	-111.790	4.3769	-111.680	0.1722	4.2047	2.53%
8	4.0983	-111.735	4.1641	-111.791	4.2047	-111.804	4.2705	-111.790	0.1064	4.2047	1.56%
9	4.1641	-111.791	4.2047	-111.804	4.2299	-111.804	4.2705	-111.790	0.0658	4.2047	0.97%
10	4.1641	-111.791	4.1892	-111.801	4.2047	-111.804	4.2299	-111.804	0.0407	4.2047	0.60%

After ten iterations, the process falls below a stopping of 1% criterion and the result is converging on the true minimum at $x = 4.2167$ where the function has a value of $y = -111.805$.

7.22 The inflection point corresponds to the point at which the derivative of the normal distribution is a minimum. The derivative can be evaluated as

$$\frac{dy}{dx} = -2xe^{-x^2}$$

Starting with initial guesses of $x_l = 0$ and $x_u = 2$, the golden-section search method can be implemented as

$$d = \frac{\sqrt{5}-1}{2}(2-0) = 1.2361$$

$$x_1 = 0 + 1.2361 = 1.2361 \qquad x_2 = 2 - 1.2361 = 0.7639$$

$$f(x_2) = f(0.7639) = -2(0.7639)e^{-(0.7639)^2} = -0.8524$$

$$f(x_1) = f(1.2361) = -2(1.2361)e^{-(1.2361)^2} = -0.5365$$

Because $f(x_2) < f(x_1)$, the minimum is in the interval defined by x_l, x_2, and x_1 where x_2 is the optimum. The error at this point can be computed as

$$\varepsilon_a = (1 - 0.61803)\left|\frac{2-0}{0.7639}\right| \times 100\% = 100\%$$

The process can be repeated and all the iterations summarized as

i	x_l	$f(x_l)$	x_2	$f(x_2)$	x_1	$f(x_1)$	x_u	$f(x_u)$	d	x_{opt}	ε_a
1	0	0.0000	0.7639	-0.8524	1.2361	-0.5365	2	-0.0733	1.2361	0.7639	100.00%
2	0	0.0000	0.4721	-0.7556	0.7639	-0.8524	1.2361	-0.5365	0.7639	0.7639	61.80%
3	0.4721	-0.7556	0.7639	-0.8524	0.9443	-0.7743	1.2361	-0.5365	0.4721	0.7639	38.20%
4	0.4721	-0.7556	0.6525	-0.8525	0.7639	-0.8524	0.9443	-0.7743	0.2918	0.6525	27.64%
5	0.4721	-0.7556	0.5836	-0.8303	0.6525	-0.8525	0.7639	-0.8524	0.1803	0.6525	17.08%
6	0.5836	-0.8303	0.6525	-0.8525	0.6950	-0.8575	0.7639	-0.8524	0.1115	0.6950	9.91%
7	0.6525	-0.8525	0.6950	-0.8575	0.7214	-0.8574	0.7639	-0.8524	0.0689	0.6950	6.13%
8	0.6525	-0.8525	0.6788	-0.8564	0.6950	-0.8575	0.7214	-0.8574	0.0426	0.6950	3.79%
9	0.6788	-0.8564	0.6950	-0.8575	0.7051	-0.8578	0.7214	-0.8574	0.0263	0.7051	2.31%
10	0.6950	-0.8575	0.7051	-0.8578	0.7113	-0.8577	0.7214	-0.8574	0.0163	0.7051	1.43%
11	0.6950	-0.8575	0.7013	-0.8577	0.7051	-0.8578	0.7113	-0.8577	0.0100	0.7051	0.88%

After eleven iterations, the process falls below a stopping of 1% criterion and the result is converging on the true minimum at $x = 0.707107$ where the function has a value of $y = -0.85776$.

7.23

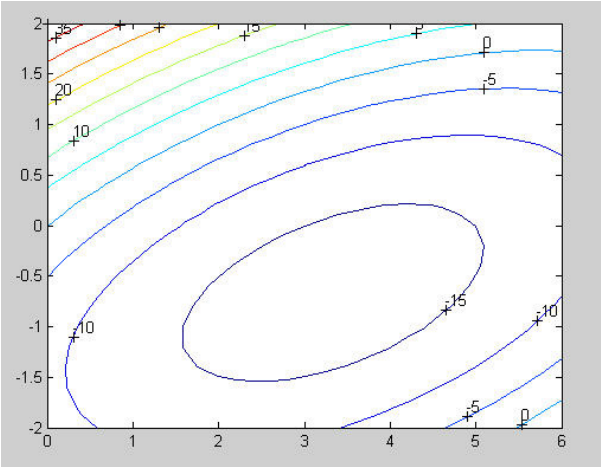
```
>> fxy=@(x) 2*x(2)^2-2.25*x(1)*x(2)-1.75*x(2)+1.5*x(1)^2;
>> [x,fval]= fminsearch(fxy,[0,0])
x =
    0.5676    0.7568
fval =
   -0.6622
```

7.24

```
>> fxy=@(x) -(4*x(1)+2*x(2)+x(1)^2-2*x(1)^4+2*x(1)*x(2)-3*x(2)^2);
>> [x,fval]= fminsearch(fxy,[0,0])
x =
    0.9676    0.6559
fval =
   -4.3440
```

7.25 (a)

```
>> [x,y] = meshgrid(0:.2:6,-2:.2:2);
>> z=-8*x+x.^2+12*y+4*y.^2-2*x.*y;
>> cs=contour(x,y,z);clabel(cs)
```



(b) and (c)

```
>> fxy=@(x) -8*x(1)+x(1)^2+12*x(2)+4*x(2)^2-2*x(1)*x(2);
>> [x,fval]= fminsearch(fxy,[0,0])
x =
    3.3333   -0.6666
fval =
  -17.3333
```

7.26 This problem can be solved in a number of different ways. For example, using the golden section search, the result is

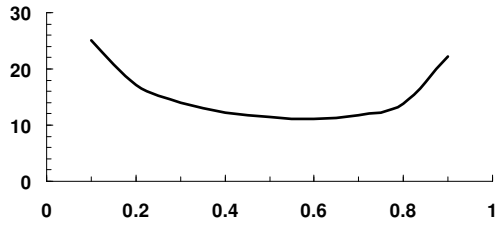
i	c_l	$g(c_l)$	c_2	$g(c_2)$	c_1	$g(c_1)$	c_u	$g(c_u)$	d	c_{opt}	ε_a
1	0.0000	0.0000	3.8197	0.2330	6.1803	0.1310	10.0000	0.0641	6.1803	3.8197	100.00%
2	0.0000	0.0000	2.3607	0.3350	3.8197	0.2330	6.1803	0.1310	3.8197	2.3607	100.00%
3	0.0000	0.0000	1.4590	0.3686	2.3607	0.3350	3.8197	0.2330	2.3607	1.4590	100.00%
4	0.0000	0.0000	0.9017	0.3174	1.4590	0.3686	2.3607	0.3350	1.4590	1.4590	61.80%
5	0.9017	0.3174	1.4590	0.3686	1.8034	0.3655	2.3607	0.3350	0.9017	1.4590	38.20%
6	0.9017	0.3174	1.2461	0.3593	1.4590	0.3686	1.8034	0.3655	0.5573	1.4590	23.61%
7	1.2461	0.3593	1.4590	0.3686	1.5905	0.3696	1.8034	0.3655	0.3444	1.5905	13.38%
8	1.4590	0.3686	1.5905	0.3696	1.6718	0.3688	1.8034	0.3655	0.2129	1.5905	8.27%
9	1.4590	0.3686	1.5403	0.3696	1.5905	0.3696	1.6718	0.3688	0.1316	1.5905	5.11%
10	1.5403	0.3696	1.5905	0.3696	1.6216	0.3694	1.6718	0.3688	0.0813	1.5905	3.16%
11	1.5403	0.3696	1.5713	0.3696	1.5905	0.3696	1.6216	0.3694	0.0502	1.5713	1.98%
12	1.5403	0.3696	1.5595	0.3696	1.5713	0.3696	1.5905	0.3696	0.0311	1.5713	1.22%
13	1.5595	0.3696	1.5713	0.3696	1.5787	0.3696	1.5905	0.3696	0.0192	1.5713	0.75%

Thus, after 13 iterations, the method is converging on the true value of $c = 1.5679$ which corresponds to a maximum specific growth rate of $g = 0.36963$.

7.27 This is a straightforward problem of varying x_A in order to minimize

$$f(x_A) = \left(\frac{1}{(1-x_A)^2}\right)^{0.6} + 6\left(\frac{1}{x_A}\right)^{0.6}$$

First, the function can be plotted versus x_A



The result indicates a minimum between 0.5 and 0.6. Using golden section search or MATLAB yields a minimum of 0.587683.

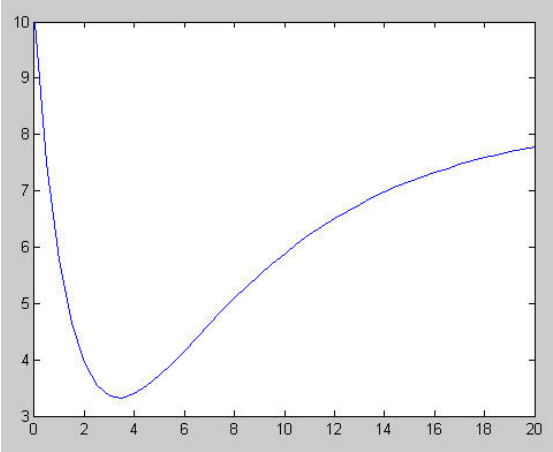
```
>> format long
>> fxa=@(xa) (1/(1-xa)^2)^0.6+6*(1/xa)^0.6;
>> [x fx]=fminbnd(fxa,0.2,1)
x =
   0.58767394491187
fx =
  11.14951022048255
```

7.28 As shown below, MATLAB gives: $x = 0.5$, $y = 0.8$ and $f_{\min} = -0.85$.

```
>> fxy=@(x) 5*x(1)^2-5*x(1)*x(2)+2.5*x(2)^2-x(1)-1.5*x(2);
>> [x,fval]=fminsearch(fxy,[0,0])
x =
    0.5000    0.8000
fval =
   -0.8500
```

7.29 A plot of the function indicates a minimum at about $t = 3.3$.

```
>> os=10;kd=0.1;ka=0.6;
>> ks=0.05;Lo=50;Sb=1;
>> ft=@(t) os-kd*Lo/(kd+ks-ka)*(exp(-ka*t)-exp(-(kd+ks)*t)) ...
                                                    -Sb/ka*(1-exp(-ka*t));
>> t=0:0.5:20;
>> o=ft(t);
>> plot(t,o)
```

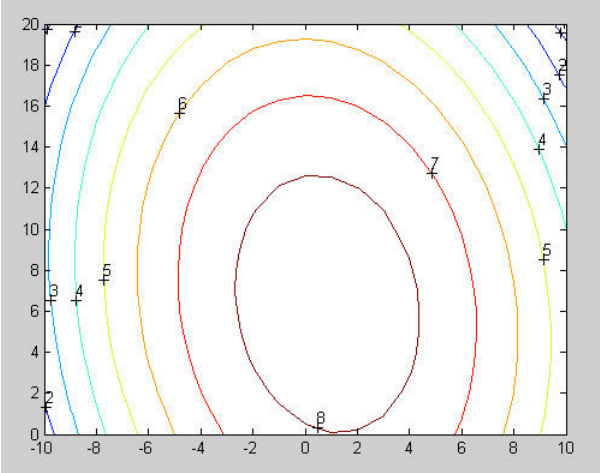


MATLAB can be used to determine that a minimum of $o = 3.3226$ occurs at a value of $t = 3.3912$.

```
>> [tmin omin]=fminbnd(ft,0,10)
tmin =
    3.3912
omin =
    3.3226
```

7.30 This problem can be solved graphically by using MATLAB to generate a contour plot of the function. As can be seen, a minimum occurs at approximately $x = 1$ and $y = 7$.

```
>> [x,y] = meshgrid(-10:10,0:20);
>> c=7.9+0.13*x+0.21*y-0.05*x.^2-0.016*y.^2-0.007*x.*y;
>> cs=contour(x,y,c);clabel(cs)
```



We can use MATLAB to determine a maximum of 8.6249 at $x = 0.853689$ and $y = 6.375787$.

```
>> c=@(x) -(7.9+0.13*x(1)+0.21*x(2)-0.05*x(1).^2...
                                  -0.016*x(2).^2-0.007*x(1).*x(2));
>> [x,fval]= fminsearch(c,[0,0])
x =
   0.85368944970842   6.37578748109051
fval =
  -8.62494446205611
```

7.31 The solution can be developed as a MATLAB script yielding $F = 1.709$ at $x = 0.6364$,

```
clear, clc, format long
e0=8.85e-12;q=2e-5;Q=q;a=0.9;
F=@(x) -(1/(4*pi*e0)*q*Q*x/(x^2+a^2)^(3/2));
[x,Fx]=fminsearch(F,0.5)

x =
   0.63642578125000
Fx =
  -1.70910974594861
```

7.32 This is a trick question. Because of the presence of $(1 - s)$ in the denominator, the function will experience a division by zero at the maximum. This can be rectified by merely canceling the $(1 - s)$ terms in the numerator and denominator to give

$$T = \frac{15s}{4s^2 - 3s + 4}$$

The following script uses `fminbnd` to determine that the maximum of $T = 3$ occurs at $s = 1$.

```
clear, clc, format long
T=@(s) -(15*s/(4*s^2-3*s+4));
[smin,Tmin]=fminbnd(T,0,4)
smin =
   0.99998407348199
Tmin =
  -2.99999999939122
```

7.33 (a) The `fminbnd` function can be used to determine the solution as in the following script:

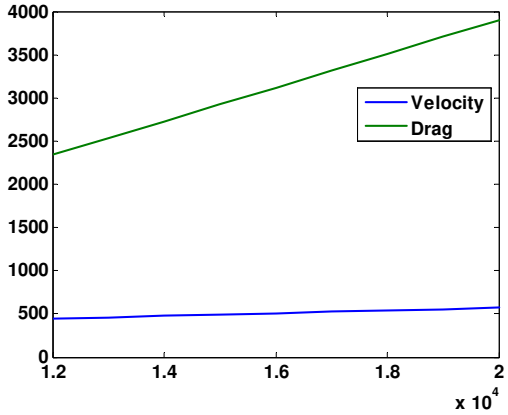
```
clear,clc,format short g
D=@(V) 0.01*(0.6)*V^2+0.95/0.6*(16000/V)^2;
[Vmin,Dmin]=fminbnd(D,0,1200)
Vmin =
       509.82
Dmin =
         3119
```

(b) The following script can be used to generate the sensitivity analysis:

```
clear,clc,format short g
W=[12000:1000:20000];
sigma=0.6;
for i=1:length(W)
  [Vmin(i),Dmin(i)]=fminbnd(@(V) 0.01*sigma*V^2+0.95/sigma*(W(i)/V)^2,0,1200);
end
z=[W;Vmin;Dmin];
fprintf('     Weight   Velocity   Drag\n')
fprintf('%10.0f %10.3f %10.3f\n',z)
clf
plot(W,Vmin,'--',W,Dmin)
legend('Velocity','Drag','location','best')
```

The results indicate that the minimum drag is fairly linear for the specified range.

```
     Weight   Velocity   Drag
     12000    441.515   2339.231
     13000    459.544   2534.167
     14000    476.891   2729.102
     15000    493.629   2924.038
     16000    509.818   3118.974
     17000    525.508   3313.910
     18000    540.744   3508.846
     19000    555.561   3703.782
     20000    569.994   3898.718
```



7.34 MATLAB can be used to determine the solution as

```
>> format long
>> fx=@(x) -(0.4/sqrt(1+x^2)-sqrt(1+x^2)*(1-0.4/(1+x^2))+x);
>> [x,f]=fminbnd(fx,0,2)
x =
   1.05185912272736
```

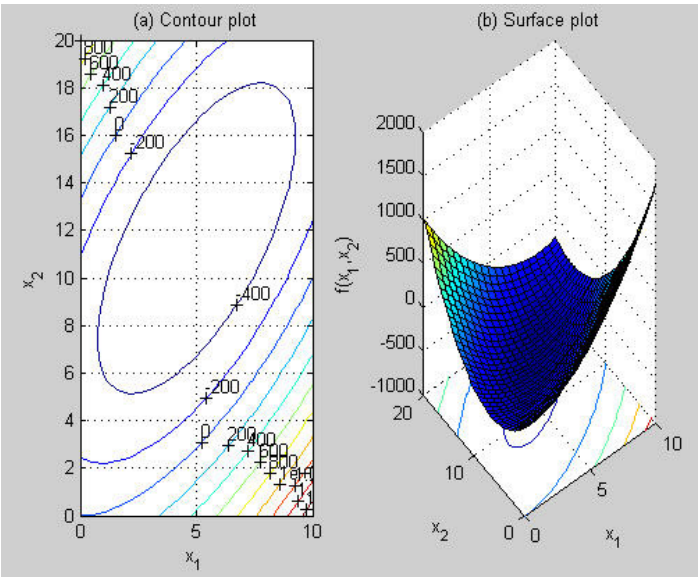
```
f =
  -0.15172444148082
```

7.35 The potential energy function can be written as

$$PE = 0.5k_a x_1^2 + 0.5k_b(x_1 - x_2)^2 - Fx_2$$

Contour and surface plots can be generated with the following script,

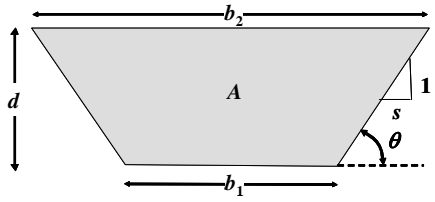
```
ka=20;kb=15;F=100;
x=linspace(0,10,20);y=linspace(0,20,40);
[x1,x2] = meshgrid(x,y);
Z=0.5*ka*x1.^2+0.5*kb*(x2-x1).^2-F*x2;
subplot(1,2,1);
cs=contour(x1,x2,Z);clabel(cs);
xlabel('x_1');ylabel('x_2');
title('(a) Contour plot');grid;
subplot(1,2,2);
cs=surfc(x1,x2,Z);
zmin=floor(min(Z));
zmax=ceil(max(Z));
xlabel('x_1');ylabel('x_2');zlabel('f(x_1,x_2)');
title('(b) Mesh plot');
```



The values of x_1 and x_2 that minimize the PE function can be determined as

```
>> PE=@(x) 0.5*ka*x(1).^2+0.5*kb*(x(2)-x(1)).^2-F*x(2);
>> [xmin,PEmin]=fminsearch(PE,[5,5])
xmin =
   4.99998098312971  11.66668727728067
PEmin =
   -5.833333333179395e+002
```

7.36 Using the diagram shown below, a number of formulas can be developed:



$$\theta = \tan^{-1}\frac{1}{s} \tag{1}$$

$$P = b_1 + 2d\sqrt{1+s^2} \tag{2}$$

$$A = (b_1 + sd)d \tag{3}$$

$$b_2 = b_1 + 2sd \tag{4}$$

We can solve Eq. 3 for b_1,

$$b_1 = \frac{A}{d} - sd \tag{5}$$

and substitute the result into Eq. 2 to give,

$$P = \frac{A}{d} + d\left(2\sqrt{1+s^2} - s\right) \tag{6}$$

Given a value for A, we can determine the values of d and s for this two-dimensional function,

```
>> P=@(x) 50/x(1)+x(1)*(2*sqrt(1+x(2).^2)-x(2));
>> [xmin,fmin]=fminsearch(P,[5,1])
xmin =
    5.3729    0.5773
fmin =
   18.6121
```

The optimal parameters are $d = 5.3729$ and $s = 0.5773$. Using Eq. 1 gives

$$\theta = \tan^{-1}\frac{1}{0.5773} = 1.0472 \text{ radians} \times \frac{180°}{\pi} = 60°$$

Thus, this specific application indicates that a 60^o angle yields the minimum wetted perimeter. The other dimensions can be computed as (Eqs. 5 and 4),

$$b_1 = \frac{50}{5.3729} - 0.5773(5.3729) = 6.2041$$

$$b_2 = 6.2041 + 2(0.5773)(5.3729) = 12.40807$$

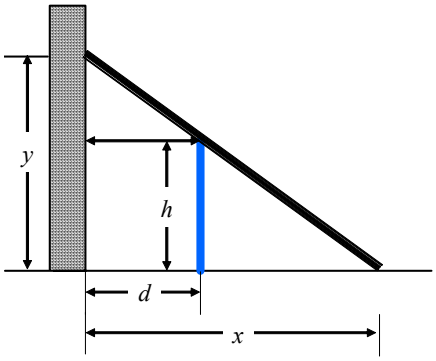
The verification of whether this result is universal can be attained inductively or deductively. The inductive approach involves trying several different desired areas in conjunction with our MATLAB solution. As long as the desired area is greater than 0, the result for the optimal design will be 60^o.

The deductive verification involves calculus. If both A and d are constants and s is a variable, the condition for the minimum perimeter is $dP/ds = 0$. Differentiating Eq. 6 with respect to s and setting the resulting equation to zero,

$$\frac{dP}{ds} = d\left(\frac{2s}{\sqrt{1+s^2}} - 1\right) = 0$$

The term in parentheses can be solved for $s = 1/\sqrt{3}$. Using Eq. 1, this corresponds to $\theta = 60^o$.

7.37 We have to first define some additional variables,



The problem amounts to minimizing the length of the ladder,

$$L = \sqrt{x^2 + y^2} \tag{1}$$

Using similar triangles

$$\frac{y}{x} = \frac{h}{x-d}$$

which can be solved for

$$y = \frac{hx}{x-d}$$

This value can be substituted into Eq. 1 along with the values of $h = d = 4$ to give

$$L = \sqrt{x^2 + \left(\frac{4x}{x-4}\right)^2}$$

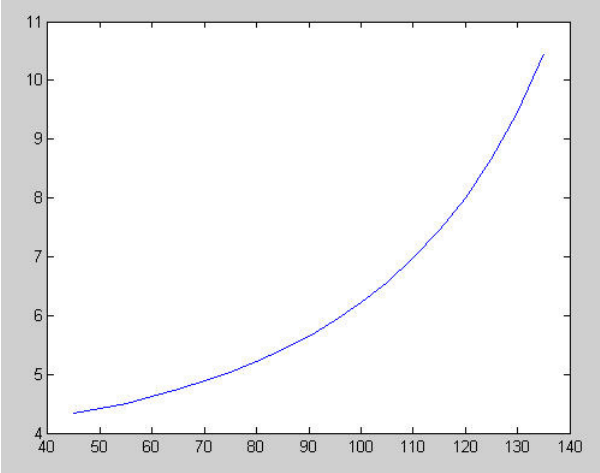
MATLAB can then be used to determine the value of x that minimizes the length,

```
>> fx=@(x) sqrt(x^2+(4*x/(x-4))^2);
>> [xmin,fmin]=fminbnd(fx,1,10)
xmin =
    8.0000
fmin =
   11.3137
```

Therefore, the shortest ladder is L = 11.3137 m, x = 8, and y = 4(8)/(8 – 4) = 8.

7.38 The following script generates the plot of L versus a range of α's:

```
alphad=[45:5:135];
alpha=alphad*pi/180;
for i=1:length(alpha)
  [t,Lmin(i)]=fminsearch(@(x) 2/sin(x)+2/sin(pi-alpha(i)-x),0.3);
end
plot(alphad,Lmin)
```



CHAPTER 8

8.1

```
>> 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
```

8.2 **(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)

(1) $[E]+[B] = \begin{bmatrix} 5 & 8 & 13 \\ 8 & 3 & 9 \\ 6 & 0 & 10 \end{bmatrix}$

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

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

(4) $7[B] = \begin{bmatrix} 28 & 21 & 49 \\ 7 & 14 & 42 \\ 14 & 0 & 28 \end{bmatrix}$

(5) $[C]^T = \lfloor 2 \quad 6 \quad 1 \rfloor$

(6) $[E][B] = \begin{bmatrix} 21 & 13 & 61 \\ 35 & 23 & 67 \\ 28 & 12 & 52 \end{bmatrix}$

(7) $[B][E] = \begin{bmatrix} 53 & 23 & 75 \\ 39 & 7 & 48 \\ 18 & 10 & 36 \end{bmatrix}$

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

(9) $[G][C] = 56$

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

(11) $E^T[E] = \begin{bmatrix} 66 & 12 & 51 \\ 12 & 26 & 33 \\ 51 & 33 & 81 \end{bmatrix}$

(12) $C^T[C] = 41$

8.3 The terms can be collected to give

$$\begin{bmatrix} 0 & -6 & 5 \\ 0 & 2 & 7 \\ -4 & 3 & -7 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 50 \\ -30 \\ 50 \end{Bmatrix}$$

Here is the MATLAB session:

```
>> A = [0 -6 5;0 2 7;-4 3 -7];
>> b = [50;-30;50];
>> x = A\b

x =
  -17.0192
   -9.6154
   -1.5385

>> AT = A'
     0     0    -4
    -6     2     3
     5     7    -7

>> AI = inv(A)
AI =
   -0.1683   -0.1298   -0.2500
   -0.1346    0.0962         0
    0.0385    0.1154         0
```

8.4 (a) Here are all the possible multiplications:

```
>> A=[6 -1;12 7;-5 3];
>> B=[4 0;0.6 8];
>> C=[1 -2;-6 1];
>> A*B
ans =
   23.4000   -8.0000
   52.2000   56.0000
  -18.2000   24.0000
>> A*C
ans =
    12   -13
   -30   -17
   -23    13
>> B*C
ans =
    4.0000   -8.0000
  -47.4000    6.8000
>> C*B
ans =
    2.8000  -16.0000
  -23.4000    8.0000
```

(b) [*B*][*A*] and [*C*][*A*] are impossible because the inner dimensions do not match:

(2×2) * (3×2)

(c) According to **(a)**, [*B*][*C*] ≠ [*C*][*B*]

8.5

```
>> A=[3+2*i 4;-i 1]
>> b=[2+i;3]
>> z=A\b

z =
  -0.5333 + 1.4000i
   1.6000 - 0.5333i
```

8.6

```
function X=mmult(Y,Z)
% mmult: matrix multiplication
%   X=mmult(Y,Z)
%     multiplies two matrices
% input:
%   Y = first matrix
%   Z = second matrix
% output:
%   X = product

if nargin<2,error('at least 2 input arguments required'),end
[m,n]=size(Y);[n2,p]=size(Z);
if n~=n2,error('Inner matrix dimensions must agree.'),end
for i=1:m
  for j=1:p
    s=0.;
    for k=1:n
      s=s+Y(i,k)*Z(k,j);
    end
    X(i,j)=s;
  end
end
```

Test of function for cases from Prob. 8.4:

```
>> A=[6 -1;12 7;-5 3];
>> B=[4 0;0.6 8];
>> C=[1 -2;-6 1];
>> mmult(A,B)
ans =
   23.4000   -8.0000
   52.2000   56.0000
  -18.2000   24.0000

>> mmult(A,C)
ans =
    12   -13
   -30   -17
   -23    13

>> mmult(B,C)
ans =
    4.0000   -8.0000
  -47.4000    6.8000

>> mmult(C,B)
ans =
    2.8000  -16.0000
  -23.4000    8.0000

>> mmult(B,A)
```

```
??? Error using ==> mmult
Inner matrix dimensions must agree.

>> mmult(C,A)
??? Error using ==> mmult
Inner matrix dimensions must agree.
```

8.7

```
function AT=matran(A)
% matran: matrix transpose
%   AT=mtran(A)
%      generates the transpose of a matrix
% input:
%   A = original matrix
% output:
%   AT = transpose

[m,n]=size(A);
for i = 1:m
  for j = 1:n
    AT(j,i) = A(i,j);
  end
end
```

Test of function for cases from Prob. 8.4:

```
>> matran(A)
ans =
     6    12    -5
    -1     7     3

>> matran(B)
    4.0000    0.6000
         0    8.0000

>> matran(C)
ans =
     1    -6
    -2     1
```

8.8

```
function B = permut(A,r1,r2)
% permut: Switch rows of matrix A with a permutation matrix
% B = permut(A,r1,r2)
% input:
% A = original matrix
% r1, r2 = rows to be switched
% output:
% B = matrix with rows switched

[m,n] = size(A);
if m ~= n, error('matrix not square'), end
if r1 == r2 | r1>m | r2>m
  error('row numbers are equal or exceed matrix dimensions')
end
P = zeros(n);
P(r1,r2)=1;P(r2,r1)=1;
for i = 1:m
  if i~=r1 & i~=r2
    P(i,i)=1;
  end
```

```
end
B=P*A;
```

Test script:

```
clc
A=[1 2 3 4;5 6 7 8;9 10 11 12;13 14 15 16]
B = permut(A,3,1)
B = permut(A,3,5)

A =
     1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16
B =
     9    10    11    12
     5     6     7     8
     1     2     3     4
    13    14    15    16

??? Error using ==> permut
row numbers are equal or exceed matrix dimensions

Error in ==> permutScript at 4
B = permut(A,3,5)
```

8.9 The mass balances can be written as

$$\begin{aligned}
(Q_{15}+Q_{12})c_1 - Q_{31}c_3 &= Q_{01}c_{01}\\
-Q_{12}c_1 + (Q_{23}+Q_{24}+Q_{25})c_2 &= 0\\
-Q_{23}c_2 + (Q_{31}+Q_{34})c_3 &= Q_{03}c_{03}\\
-Q_{24}c_2 - Q_{34}c_3 + Q_{44}c_4 - Q_{54}c_5 &= 0\\
-Q_{15}c_1 - Q_{25}c_2 + (Q_{54}+Q_{55})c_5 &= 0
\end{aligned}$$

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

$$\begin{bmatrix} 9 & 0 & -3 & 0 & 0\\ -4 & 4 & 0 & 0 & 0\\ 0 & -2 & 9 & 0 & 0\\ 0 & -1 & -6 & 9 & -2\\ -5 & -1 & 0 & 0 & 6 \end{bmatrix} \begin{Bmatrix} c_1\\ c_2\\ c_3\\ c_4\\ c_5 \end{Bmatrix} = \begin{Bmatrix} 120\\ 0\\ 350\\ 0\\ 0 \end{Bmatrix}$$

The following MATLAB script can then be used to solve for the concentrations

```
clc
Q = [9 0 -3 0 0;
-4 4 0 0 0;
0 -2 9 0 0;
0 -1 -6 9 -2;
-5 -1 0 0 6];
Qc = [120;0;350;0;0];
c = Q\Qc

c =
```

```
   28.4000
   28.4000
   45.2000
   39.6000
   28.4000
```

8.10 The problem can be written in matrix form as

$$\begin{bmatrix} 0.866025 & 0 & -0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0.866025 & 0 & 0 & 0 \\ -0.866025 & -1 & 0 & -1 & 0 & 0 \\ -0.5 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & -0.866025 & 0 & 0 & -1 \end{bmatrix} \begin{Bmatrix} F_1 \\ F_2 \\ F_3 \\ H_2 \\ V_2 \\ V_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ -2000 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix}$$

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

```
clc; format short g
A = [0.866025 0 -0.5 0 0 0;
0.5 0 0.866025 0 0 0;
-0.866025 -1 0 -1 0 0;
-0.5 0 0 0 -1 0;
0 1 0.5 0 0 0;
0 0 -0.866025 0 0 -1];
b = [0 -2000 0 0 0 0]';
F = A\b

F =
        -1000
       866.03
      -1732.1
            0
          500
         1500
```

Therefore,

$F_1 = -1000$ $F_2 = 866.025$ $F_3 = -1732.1$ $H_2 = 0$ $V_2 = 500$ $V_3 = 1500$

8.11

```
clc; format short g
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

km =
    50   -40     0
   -40    80   -40
     0   -40    50
kmx =
          -0.9
 -2.2204e-016
           0.1
```

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

8.12 Vertical force balances can be written to give the following system of equations,

$$
\begin{aligned}
&m_1 g + k_2(x_2 - x_1) - k_1 x_1 = 0\\
&m_2 g + k_3(x_3 - x_2) - k_2(x_2 - x_1) = 0\\
&m_3 g + k_4(x_4 - x_3) - k_3(x_3 - x_2) = 0\\
&m_4 g + k_5(x_5 - x_4) - k_4(x_4 - x_3) = 0\\
&m_5 g - k_5(x_5 - x_4) = 0
\end{aligned}
$$

Collecting terms,

$$
\begin{bmatrix}
k_1 + k_2 & -k_2 & & & \\
-k_2 & k_2 + k_3 & -k_3 & & \\
 & -k_3 & k_3 + k_4 & -k_4 & \\
 & & -k_4 & k_4 + k_5 & -k_5\\
 & & & -k_5 & k_5
\end{bmatrix}
\begin{Bmatrix} x_1\\ x_2\\ x_3\\ x_4\\ x_5 \end{Bmatrix} =
\begin{Bmatrix} m_1 g\\ m_2 g\\ m_3 g\\ m_4 g\\ m_5 g \end{Bmatrix}
$$

After substituting the parameters, the equations can be expressed as (g = 9.81),

$$
\begin{bmatrix}
120 & -40 & & & \\
-40 & 110 & -70 & & \\
 & -70 & 170 & -100 & \\
 & & -100 & 120 & -20\\
 & & & -20 & 20
\end{bmatrix}
\begin{Bmatrix} x_1\\ x_2\\ x_3\\ x_4\\ x_5 \end{Bmatrix} =
\begin{Bmatrix} 637.65\\ 735.75\\ 588.60\\ 735.75\\ 882.90 \end{Bmatrix}
$$

The solution can then be obtained with the following MATLAB script:

```
clc; format short g
g=9.81;
m1=65;m2=75;m3=60;m4=75;m5=90;
k1=80;k2=40;k3=70;k4=100;k5=20;
A=[k1+k2 -k2 0 0 0
-k2 k2+k3 -k3 0 0
0 -k3 k3+k4 -k4 0
0 0 -k4 k4+k5 -k5
0 0 0 -k5 k5]
b=[m1*g m2*g m3*g m4*g m5*g]'
x=A\b

A =
   120   -40     0     0     0
   -40   110   -70     0     0
     0   -70   170  -100     0
     0     0  -100   120   -20
     0     0     0   -20    20
b =
       637.65
       735.75
        588.6
```

```
       735.75
        882.9
x =
       44.758
       118.33
       149.87
       166.05
        210.2
```

8.13 The position of the three masses can be modeled by the following steady-state force balances

$$0 = k(x_2 - x_1) + m_1 g - kx_1$$
$$0 = k(x_3 - x_2) + m_2 g - k(x_2 - x_1)$$
$$0 = m_3 g - k(x_3 - x_2)$$

Terms can be combined to yield

$$\begin{aligned} 2kx_1 - kx_2 &= m_1 g \\ -kx_1 + 2kx_2 - kx_3 &= m_2 g \\ -kx_2 + kx_3 &= m_3 g \end{aligned}$$

Substituting the parameter values

$$\begin{bmatrix} 30 & -15 & 0 \\ -15 & 30 & -15 \\ 0 & -15 & 15 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 19.62 \\ 24.525 \\ 29.43 \end{Bmatrix}$$

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

```
clc; format short g
g=9.81;k=15;
K=[2*k -k 0;-k 2*k -k;0 -k k]
m=[2;2.5;3];
mg=m*g
x=K\mg

K =
    30   -15     0
   -15    30   -15
     0   -15    15
mg =
        19.62
       24.525
        29.43
x =
        4.905
        8.502
       10.464
```

8.14 Just as in Sec. 8.3, the simultaneous equations can be expressed in matrix form as

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & R_{52} & -R_{32} & 0 & -R_{54} & -R_{43} \\ R_{12} & -R_{52} & 0 & -R_{65} & 0 & 0 \end{bmatrix} \begin{Bmatrix} i_{12} \\ i_{52} \\ i_{32} \\ i_{65} \\ i_{54} \\ i_{43} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ V_1 - V_6 \end{Bmatrix}$$

or substituting the resistances

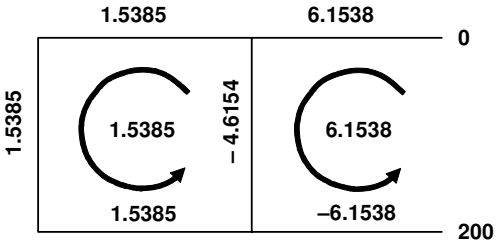
$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 10 & -10 & 0 & -15 & -5 \\ 5 & -10 & 0 & -20 & 0 & 0 \end{bmatrix} \begin{Bmatrix} i_{12} \\ i_{52} \\ i_{32} \\ i_{65} \\ i_{54} \\ i_{43} \end{Bmatrix} = \begin{Bmatrix} 0 \\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

This system can be solved with MATLAB,

```
clc; format short g
R12=5;R52=10;R32=10;R65=20;R54=15;R43=5;
V1=200;V6=0;
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 R52 -R32 0 -R54 -R43;
R12 -R52 0 -R65 0 0]
B=[0 0 0 0 0 V1-V6]'
I=A\B

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

$i_{21} = 6.1538$ $i_{52} = -4.6154$ $i_{32} = -1.5385$ $i_{65} = -6.1538$ $i_{54} = -1.5385$ $i_{43} = -1.5385$

Here are the resulting currents superimposed on the circuit:



8.15 The current equations can be written as

$$-i_{21} - i_{23} + i_{52} = 0$$
$$i_{23} - i_{35} + i_{43} = 0$$
$$-i_{43} + i_{54} = 0$$
$$i_{35} - i_{52} + i_{65} - i_{54} = 0$$

Voltage equations:

$$i_{21} = \frac{V_2 - 20}{35} \qquad i_{54} = \frac{V_5 - V_4}{15}$$
$$i_{23} = \frac{V_2 - V_3}{30} \qquad i_{35} = \frac{V_3 - V_5}{7}$$
$$i_{43} = \frac{V_4 - V_3}{8} \qquad i_{52} = \frac{V_5 - V_2}{10}$$
$$i_{65} = \frac{140 - V_5}{5}$$

$$\begin{bmatrix} -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 35 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 30 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 15 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 10 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} i_{21} \\ i_{23} \\ i_{52} \\ i_{35} \\ i_{43} \\ i_{54} \\ i_{65} \\ V_2 \\ V_3 \\ V_4 \\ V_5 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -20 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 140 \end{Bmatrix}$$

A MATLAB script can be developed to generate the solution,

```
clc; format short g
R12=35;R52=10;R32=30;R34=8;R45=15;R35=7;R25=10;R65=5;
V1=20;V6=140;
A=[-1   -1  1   0   0   0   0   0  0  0  0;
    0   1   0   -1  1   0   0   0  0  0  0;
```

```
    0   0   0   0   -1  1   0   0  0  0  0;
    0   0   -1  1   0   -1  1   0  0  0  0;
    R12 0   0   0   0   0   0   -1 0  0  0;
    0   R32 0   0   0   0   0   -1 1  0  0;
    0   0   0   0   R34 0   0   0  1  -1 0;
    0   0   0   0   0   R45 0   0  0  1  -1;
    0   0   0   R35 0   0   0   0  -1 0  1;
    0   0   R25 0   0   0   0   1  0  0  -1;
    0   0   0   0   0   0   R65 0  0  0  1]
B=[0 0 0 0 -V1 0 0 0 0 0 V6]'
I=A\B

I =
       2.5107
     -0.55342
       1.9573
     -0.42429
      0.12913
      0.12913
       2.5107
       107.87
       124.48
       125.51
       127.45
```

Thus, the solution is

$i_{21} = 2.5107$ $i_{23} = -0.55342$ $i_{52} = 1.9573$ $i_{35} = -0.42429$ $i_{43} = 0.12913$
$i_{54} = 0.12913$ $i_{65} = 2.5107$ $V_2 = 107.87$ $V_3 = 124.48$ $V_4 = 125.51$
$V_5 = 127.45$

CHAPTER 9

9.1 The flop counts for the tridiagonal algorithm in Fig. 9.6 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	$5n-4$	$3n-3$	$\boldsymbol{8n-7}$

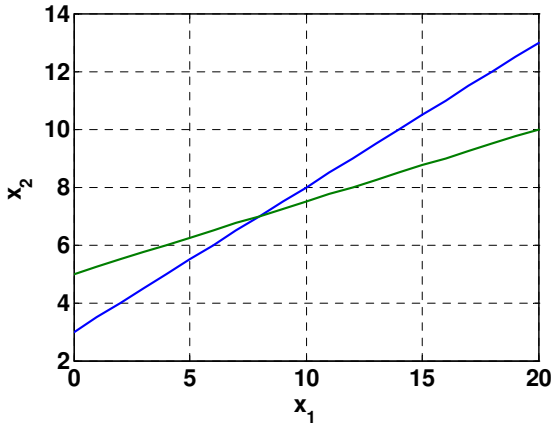
Thus, as n increases, the effort is much, much less than for a full matrix solved with Gauss elimination which is proportional to n^3.

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

$$x_2 = \frac{18+3x_1}{6} = 3+0.5x_1$$
$$x_2 = \frac{40+2x_1}{8} = 5+0.25x_1$$

which can be plotted as

```
a11=3;a12=-6;b1=-18;
a21=-2;a22=8;b2=40;
x1=[0:20];x21=(b1-a11*x1)/a12;x22=(b2-a21*x1)/a22;
plot(x1,x21,x1,x22,'--'),grid
xlabel('x_1'),ylabel('x_2')
```

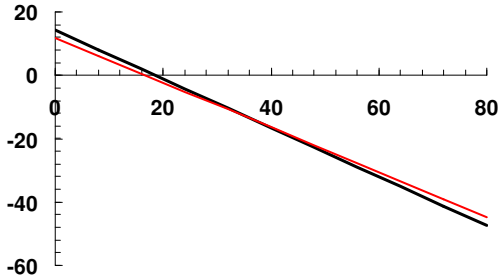


Thus, the solution is $x_1 = 8$, $x_2 = 7$. The solution can be checked by substituting it back into the equations to give

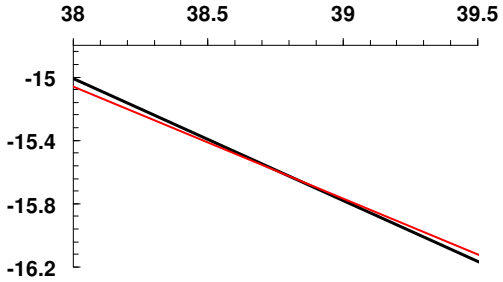
$$3(8)-6(7) = 24-42 = -18$$
$$-2(8)+8(7) = -16+56 = 40$$

9.3 (a) The equations can be rearranged into a format for plotting x_2 versus x_1:

$$x_2 = 14.25 - 0.77x_1$$
$$x_2 = 11.76471 - \frac{1.2}{1.7}x_1$$



If you zoom in, it appears that there is a root at about (38.7, –15.6).



The results can be checked by substituting them back into the original equations:

$$0.77(38.7) - 15.6 = 14.2 \cong 14.25$$
$$1.2(38.7) + 1.7(-15.6) = 19.92 \cong 20$$

(b) The plot suggests that the system may be ill-conditioned because the slopes are so similar.

(c) The determinant can be computed as

$$D = 0.77(1.7) - 1(1.2) = 0.11$$

which is relatively small. Note that if the system is normalized first by dividing each equation by the largest coefficient,

$$0.77x_1 + x_2 = 14.25$$
$$0.705882x_1 + x_2 = 11.76471$$

the determinant is even smaller

$$D = 0.77(1) - 1(0.705882) = 0.064$$

9.4 (a) The determinant can be computed as:

$$A_1 = \begin{vmatrix} 1 & 1 \\ 1 & 0 \end{vmatrix} = 1(0) - 1(1) = -1$$
$$A_2 = \begin{vmatrix} 2 & 1 \\ 3 & 0 \end{vmatrix} = 2(0) - 1(3) = -3$$
$$A_3 = \begin{vmatrix} 2 & 1 \\ 3 & 1 \end{vmatrix} = 2(1) - 1(3) = -1$$

$$D = 0(-1) - 2(-3) + 5(-1) = 1$$

(b) Cramer's rule

$$x_1 = \frac{\begin{vmatrix} 1 & 2 & 5 \\ 1 & 1 & 1 \\ 2 & 1 & 0 \end{vmatrix}}{D} = \frac{-2}{1} = -2$$

$$x_2 = \frac{\begin{vmatrix} 0 & 1 & 5 \\ 2 & 1 & 1 \\ 3 & 2 & 0 \end{vmatrix}}{D} = \frac{8}{1} = 8$$

$$x_3 = \frac{\begin{vmatrix} 0 & 2 & 1 \\ 2 & 1 & 1 \\ 3 & 1 & 2 \end{vmatrix}}{D} = \frac{-3}{1} = -3$$

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

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

Multiply pivot row 1 by 2/3 and subtract the result from the second row to eliminate the a_{21} term. Note that because $a_{31} = 0$, it does not have to be eliminated

$$\begin{aligned} 3x_1 + x_2 &= 2 \\ 0.33333x_2 + x_3 &= -0.33333 \\ 2x_2 + 5x_3 &= 1 \end{aligned}$$

Pivoting is necessary so switch the second and third row,

$$\begin{aligned} 3x_1 + x_2 &= 2 \\ 2x_2 + 5x_3 &= 1 \\ 0.33333x_2 + x_3 &= -0.33333 \end{aligned}$$

Multiply pivot row 2 by 0.33333/2 and subtract the result from the third row to eliminate the a_{32} term.

$$\begin{aligned} 3x_1 + x_2 &= 2 \\ 2x_2 + 5x_3 &= 1 \\ +0.16667x_3 &= -0.5 \end{aligned}$$

The solution can then be obtained by back substitution

$$x_3 = \frac{-0.5}{0.16667} = -3$$

$$x_2 = \frac{1 - 5(-3)}{2} = 8$$

$$x_1 = \frac{2 - 0(-3) - 1(8)}{3} = -2$$

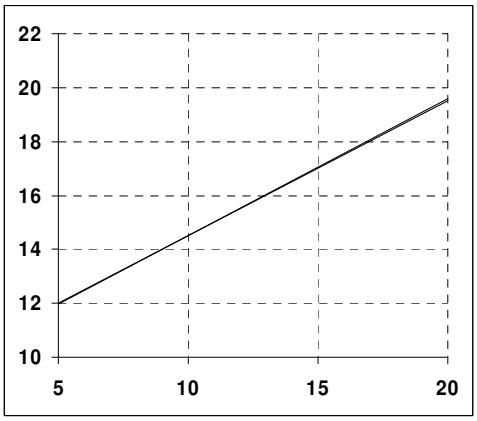
The results can be checked by substituting them back into the original equations:

$$2(8) + 5(-3) = 1$$
$$2(-2) + 8 - 3 = 1$$
$$3(-2) + 8 = 2$$

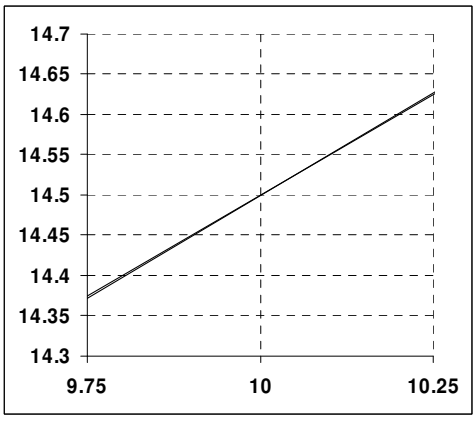
9.5 (a) The equations can be expressed in a format that is compatible with graphing x_2 versus x_1:

$$x_2 = 0.5x_1 + 9.5$$
$$x_2 = 0.51x_1 + 9.4$$

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,

$$\begin{aligned} 0.5x_1 - x_2 &= -9.5 \\ 0.04x_2 &= 0.58 \end{aligned}$$

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,

$$\begin{aligned} 0.52x_1 - x_2 &= -9.5 \\ -0.03846x_2 &= -0.16538 \end{aligned}$$

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.

9.6 (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.

$$\begin{aligned} 10x_1 + 2x_2 - x_3 &= 27 \\ -4.4x_2 + 1.7x_3 &= -53.4 \\ 0.8x_2 + 6.1x_3 &= -24.2 \end{aligned}$$

Multiply the second equation by 0.8/(–4.4) and subtract the result from the third equation to eliminate the x_2 term from the third equation,

$$\begin{aligned} 10x_1 + 2x_2 \qquad & - x_3 = 27 \\ -4.4x_2 \qquad & + 1.7x_3 = -53.4 \\ 6.409091x_3 &= -33.9091 \end{aligned}$$

Back substitution can then be used to determine the unknowns

$$x_3 = \frac{-33.9091}{6.409091} = -5.29078$$

$$x_2 = \frac{(-53.4 - 1.7(-5.29078))}{-4.4} = 10.0922$$

$$x_1 = \frac{(27 - 5.29078 - 2(10.0922))}{10} = 0.152482$$

(b) Check:

$$10(0.152482) + 2(10.0922) - (-5.29078) = 27$$

$$-3(0.152482) - 5(10.0922) + 2(-5.29078) = -61.5$$

$$0.152482 + 10.0922 + 5(-5.29078) = -21.5$$

9.7 (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.

$$\begin{aligned} -8x_1 \qquad & + x_2 \qquad - 2x_3 = -20 \\ & -1.375x_2 + 7.75x_3 = -26.5 \\ & -5.75x_2 \quad -1.5x_3 = -43 \end{aligned}$$

Pivoting is necessary so switch the second and third row,

$$\begin{aligned} -8x_1 \qquad & + x_2 \qquad - 2x_3 = -20 \\ & -5.75x_2 \quad -1.5x_3 = -43 \\ & -1.375x_2 + 7.75x_3 = -26.5 \end{aligned}$$

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

$$\begin{aligned} -8x_1 \qquad & + x_2 \qquad\qquad - 2x_3 = -20 \\ & -5.75x_2 \qquad -1.5x_3 = -43 \\ & \qquad\qquad 8.108696x_3 = -16.21739 \end{aligned}$$

At this point, the determinant can be computed as

$$D = -8 \times -5.75 \times 8.108696 \times (-1)^2 = 373$$

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:

$$2(4) - 6(8) - (-2) = -38$$
$$-3(4) - (8) + 7(-2) = -34$$
$$-8(4) + (8) - 2(-2) = -20$$

9.8 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:

$$0.8(173.75) - 0.4(245) = 41$$
$$-0.4(173.75) + 0.8(245) - 0.4(253.75) = 25$$
$$-0.4(245) + 0.8(253.75) = 105$$

9.9 Mass balances can be written for each of the reactors as

$$200 - Q_{13}c_1 - Q_{12}c_1 + Q_{21}c_2 = 0$$
$$Q_{12}c_1 - Q_{21}c_2 - Q_{23}c_2 = 0$$
$$500 + Q_{13}c_1 + Q_{23}c_2 - Q_{33}c_3 = 0$$

Values for the flows can be substituted and the system of equations can be written in matrix form as

$$\begin{bmatrix} 130 & -30 & 0 \\ -90 & 90 & 0 \\ -40 & -60 & 120 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \end{Bmatrix} = \begin{Bmatrix} 200 \\ 0 \\ 500 \end{Bmatrix}$$

The solution can then be developed using MATLAB,

```
>> A=[130 -30 0;-90 90 0;-40 -60 120];
>> B=[200;0;500];
>> C=A\B

C =
    2.0000
    2.0000
    5.8333
```

9.10 Let x_i = the volume taken from pit i. Therefore, the following system of equations must hold

$$0.52x_1 + 0.20x_2 + 0.25x_3 = 4800$$
$$0.30x_1 + 0.50x_2 + 0.20x_3 = 5800$$
$$0.18x_1 + 0.30x_2 + 0.55x_3 = 5700$$

MATLAB can be used to solve this system of equations for

```
>> A=[0.55 0.2 0.25;0.3 0.5 0.2;0.18 0.3 0.55];
>> b=[4800;5800;5700];
>> x=A\b

x =
  1.0e+003 *
    3.7263
    7.2991
    5.1628
```

Therefore, we take x_1 = 3726.3, x_2 = 7299.1, and x_3 = 5162.8 m^3 from pits 1, 2 and 3 respectively.

9.11 Let c_i = component i. Therefore, the following system of equations must hold

$$15c_1 + 17c_2 + 19c_3 = 2120$$
$$0.25c_1 + 0.33c_2 + 0.42c_3 = 43.4$$
$$1.0c_1 + 1.2c_2 + 1.6c_3 = 164$$

The solution can be developed with MATLAB:

```
A=[15 17 19;0.25 0.33 0.42;1 1.2 1.6];
b=[2120;43.4;164];
```

```
c=A\b

c =
   20.0000
   40.0000
   60.0000
```

Therefore, $c_1 = 20$, $c_2 = 40$, and $c_3 = 60$.

9.12 Centered differences (recall Chap. 4) can be substituted for the derivatives to give

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

collecting terms yields

$$-(D + 0.5U\Delta x)c_{i-1} + (2D + k\Delta x^2)c_i - (D - 0.5U\Delta x)c_{i+1} = 0$$

Assuming $\Delta x = 1$ and substituting the parameters gives

$$-2.5c_{i-1} + 4.2c_i - 1.5c_{i+1} = 0$$

For the first interior node ($i = 1$),

$$4.2c_1 - 1.5c_2 = 200$$

For the last interior node ($i = 9$)

$$-2.5c_8 + 4.2c_9 = 15$$

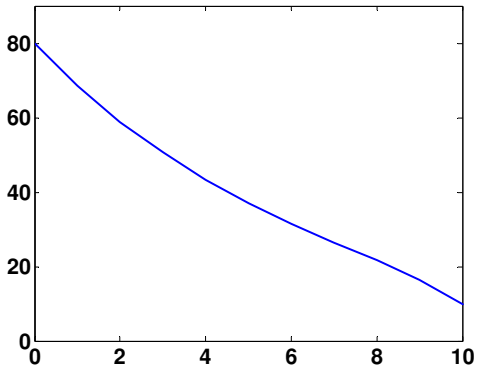
These and the equations for the other interior nodes can be assembled in matrix form as

$$\begin{bmatrix} 4.2 & -1.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2.5 & 4.2 & -1.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2.5 & 4.2 & -1.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2.5 & 4.2 & -1.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2.5 & 4.2 & -1.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2.5 & 4.2 & -1.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2.5 & 4.2 & -1.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2.5 & 4.2 & -1.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2.5 & 4.2 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \\ c_9 \end{Bmatrix} = \begin{Bmatrix} 200 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 15 \end{Bmatrix}$$

The following script generates and plots the solution:

```
A=[4.2 -1.5 0 0 0 0 0 0 0
     -2.5 4.2 -1.5 0 0 0 0 0 0
     0 -2.5 4.2 -1.5 0 0 0 0 0
     0 0 -2.5 4.2 -1.5 0 0 0 0
     0 0 0 -2.5 4.2 -1.5 0 0 0
     0 0 0 0 -2.5 4.2 -1.5 0 0
     0 0 0 0 0 -2.5 4.2 -1.5 0
     0 0 0 0 0 0 -2.5 4.2 -1.5
```

```
     0 0 0 0 0 0 0 -2.5 4.2];
b=[200 0 0 0 0 0 0 0 15]';
c=A\b;
c=[80 c' 10];
x=0:1:10;
plot(x,c)
```



9.13 For the first stage, the mass balance can be written as

$$F_1 y_{\text{in}} + F_2 x_2 = F_2 x_1 + F_1 x_1$$

Substituting $x = Ky$ and rearranging gives

$$-\left(1+\frac{F_2}{F_1}K\right)y_1 + \frac{F_2}{F_1}Ky_2 = -y_{\text{in}}$$

Using a similar approach, the equation for the last stage is

$$y_4 - \left(1+\frac{F_2}{F_1}K\right)y_5 = -\frac{F_2}{F_1}x_{\text{in}}$$

For interior stages,

$$y_{\text{i-1}} - \left(1+\frac{F_2}{F_1}K\right)y_{\text{i}} + \frac{F_2}{F_1}Ky_{i+1} = 0$$

These equations can be used to develop the following system,

$$\begin{bmatrix} 11 & -10 & 0 & 0 & 0 \\ -1 & 11 & -10 & 0 & 0 \\ 0 & -1 & 11 & -10 & 0 \\ 0 & 0 & -1 & 11 & -10 \\ 0 & 0 & 0 & -1 & 11 \end{bmatrix} \begin{Bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{Bmatrix} = \begin{Bmatrix} 0.1 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix}$$

The solution can be developed in a number of ways. For example, using MATLAB,

```
>> format short g
>> A=[11 -10 0 0 0;-1 11 -10 0 0;0 -1 11 -10 0;0 0 -1 11 -10;0 0 0 -1 11];
>> B=[0.1;0;0;0;0];
```

```
>> Y=A\B
Y =
    0.0099999
    0.0009999
    9.99e-005
     9.9e-006
       9e-007
```

Note that the corresponding values of X can be computed as

```
>> X=5*Y
X =
         0.05
    0.0049995
    0.0004995
    4.95e-005
     4.5e-006
```

Therefore, $y_{out} = 0.0000009$ and $x_{out} = 0.05$.

9.14 Assuming a unit flow for Q_1, the simultaneous equations can be written in matrix form as

$$\begin{bmatrix} -2 & 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & -2 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & -2 & 3 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & -1 \end{bmatrix} \begin{Bmatrix} Q_2 \\ Q_3 \\ Q_4 \\ Q_5 \\ Q_6 \\ Q_7 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{Bmatrix}$$

These equations can then be solved with MATLAB,

```
>> A=[-2 1 2 0 0 0;
0 0 -2 1 2 0;
0 0 0 0 -2 3;
1 1 0 0 0 0;
0 1 -1 -1 0 0;
0 0 0 1 -1 -1];
>> B=[0 0 0 1 0 0 ]';
>> Q=A\B

Q =
    0.5059
    0.4941
    0.2588
    0.2353
    0.1412
    0.0941
```

9.15 The solution can be generated with MATLAB,

```
>> A=[1 0 0 0 0 0 0 0 1 0;
      0 0 1 0 0 0 0 1 0 0;
      0 1 0 3/5 0 0 0 0 0 0;
      -1 0 0 -4/5 0 0 0 0 0 0;
      0 -1 0 0 0 0 3/5 0 0 0;
      0 0 0 0 -1 0 -4/5 0 0 0;
      0 0 -1 -3/5 0 1 0 0 0 0;
      0 0 0 4/5 1 0 0 0 0 0;
```

```
      0 0 0 0 0 -1 -3/5 0 0 0;
      0 0 0 0 0 0 4/5 0 0 1];
>> B=[0 0 -74 0 0 24 0 0 0 0]';
>> x=A\B

x =
   37.3333
  -46.0000
   74.0000
  -46.6667
   37.3333
   46.0000
  -76.6667
  -74.0000
  -37.3333
   61.3333
```

Therefore, in kN

$AB = 37.3333$ $BC = -46$ $AD = 74$ $BD = -46.6667$ $CD = 37.3333$
$DE = 46$ $CE = -76.6667$ $A_x = -74$ $A_y = -37.33333$ $E_y = 61.3333$

9.16

```
function x=pentasol(A,b)
% pentasol: pentadiagonal system solver banded system
%   x=pentasol(A,b):
%      Solve a pentadiagonal system Ax=b
% input:
%   A = pentadiagonal matrix
%   b = right hand side vector
% output:
%   x = solution vector

% Error checks
[m,n]=size(A);
if m~=n,error('Matrix must be square');end
if length(b)~=m,error('Matrix and vector must have the same number of
rows');end
x=zeros(n,1);

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

% Decomposition
delta(1)=f(1);
epsilon(1)=g(1)/delta(1);
gamma(1)=h(1)/delta(1);
alpha(2)=e(2);
delta(2)=f(2)-alpha(2)*epsilon(1);
epsilon(2)=(g(2)-alpha(2)*gamma(1))/delta(2);
```

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

Test of function:

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

x =
    0.7993
    0.5721
    0.2503
    0.5491
    0.5599
```

9.17 Here is the M-file function based on Fig. 9.5 to implement Gauss elimination with partial pivoting

```
function [x, D] = GaussPivotNew(A, b, tol)
% GaussPivotNew: Gauss elimination pivoting
%   [x, D] = GaussPivotNew(A,b,tol): Gauss elimination with pivoting.
% input:
%   A = coefficient matrix
%   b = right hand side vector
%   tol = tolerance for detecting "near zero"
% output:
%   x = solution vector
%   D = determinant

[m,n]=size(A);
if m~=n, error('Matrix A must be square'); end
nb=n+1;
Aug=[A b];
npiv=0;
% forward elimination
```

```
for k = 1:n-1
  % partial pivoting
  [big,i]=max(abs(Aug(k:n,k)));
  ipr=i+k-1;
  if ipr~=k
    npiv=npiv+1;
    Aug([k,ipr],:)=Aug([ipr,k],:);
  end
  absakk=abs(Aug(k,k));
  if abs(Aug(k,k))<=tol
    D=0;
    error('Singular or near singular system')
  end
  for i = k+1:n
    factor=Aug(i,k)/Aug(k,k);
    Aug(i,k:nb)=Aug(i,k:nb)-factor*Aug(k,k:nb);
  end
end
for i = 1:n
  if abs(Aug(i,i))<=tol
    D=0;
    error('Singular or near singular system')
  end
end
% back substitution
x=zeros(n,1);
x(n)=Aug(n,nb)/Aug(n,n);
for i = n-1:-1:1
  x(i)=(Aug(i,nb)-Aug(i,i+1:n)*x(i+1:n))/Aug(i,i);
end
D=(-1)^npiv;
for i=1:n
  D=D*Aug(i,i);
end
```

Here is a script to solve Prob. 9.5 for the two cases of tol:

```
clear; clc; format short g
A=[0.5 -1;1.02 -2];
b=[-9.5;-18.8];
disp('Solution and determinant calculated with GaussPivotNew:')
[x, D] = GaussPivotNew(A,b,1e-5)
disp('Determinant calculated with det:')
D=det(A)
```

The resulting output is

```
Solution and determinant calculated with GaussPivotNew:
x =
          10
        14.5
D =
        0.02

Determinant calculated with det:
D =
        0.02
```

CHAPTER 10

10.1 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.

10.2 Equation (10.6) is

$$[L]\{[U]\{x\}-\{d\}\}=[A]\{x\}-\{b\} \tag{10.6}$$

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. (10.7) follows

$$[L][U]=[A] \tag{10.7}$$

Equating the constant terms yields Eq. (10.8)

$$[L]\{d\}=\{b\} \tag{10.8}$$

10.3 (a) The coefficient a_{21} is eliminated by multiplying row 1 by f_{21} = 2/7 and subtracting the result from row 2. a_{31} is eliminated by multiplying row 1 by f_{31} = 1/7 and subtracting the result from row 3. The factors f_{21} and f_{31} can be stored in a_{21} and a_{31}.

$$\begin{bmatrix} 7 & 2 & -3 \\ 0.285714 & 4.428571 & -2.14286 \\ 0.142857 & -1.28571 & -5.57143 \end{bmatrix}$$

a_{32} is eliminated by multiplying row 2 by f_{32} = –1.28571/4.428571 = –0.29032 and subtracting the result from row 3. The factor f_{32} can be stored in a_{32}.

$$\begin{bmatrix} 7 & 2 & -3 \\ 0.285714 & 4.428571 & -2.14286 \\ 0.142857 & -0.29032 & -6.19355 \end{bmatrix}$$

Therefore, the *LU* decomposition is

$$[L]=\begin{bmatrix}1 & 0 & 0\\ 0.285714 & 1 & 0\\ 0.142857 & -0.29032 & 1\end{bmatrix} \qquad [U]=\begin{bmatrix}7 & 2 & -3\\ 0 & 4.428571 & -2.14286\\ 0 & 0 & -6.19355\end{bmatrix}$$

These two matrices can be multiplied to yield the original system. For example, using MATLAB to perform the multiplication gives

```
>> L=[1 0 0;0.285714 1 0;0.142857 -0.29032 1];
>> U=[7 2 -3;0 4.428571 -2.14286;0 0 -6.19355];
>> L*U
ans =
    7.0000    2.0000   -3.0000
    2.0000    5.0000   -3.0000
    1.0000   -1.0000   -6.0000
```

10.4 (a) Forward substitution: $[L]\{D\} = \{B\}$

$$\begin{bmatrix}1 & 0 & 0\\ 0.285714 & 1 & 0\\ 0.142857 & -0.29032 & 1\end{bmatrix}\begin{Bmatrix}d_1\\ d_2\\ d_3\end{Bmatrix}=\begin{Bmatrix}-12\\ -20\\ -26\end{Bmatrix}$$

Solving yields $d_1 = -12$, $d_2 = -16.5714$, and $d_3 = -29.0968$.

Back substitution:

$$\begin{bmatrix}7 & 2 & -3\\ 0 & 4.428571 & -2.14286\\ 0 & 0 & -6.19355\end{bmatrix}\begin{Bmatrix}x_1\\ x_2\\ x_3\end{Bmatrix}=\begin{Bmatrix}-12\\ -16.5714\\ -29.0968\end{Bmatrix}$$

$$x_3=\frac{-29.0968}{-6.19355}=4.697917$$

$$x_2=\frac{-16.5714-(-2.1486)(4.697917)}{4.428571}=-1.46875$$

$$x_1=\frac{-12-(-3)4.697917-2(1.46875)}{7}=0.71875$$

(b) Forward substitution: $[L]\{D\} = \{B\}$

$$\begin{bmatrix}1 & 0 & 0\\ 0.285714 & 1 & 0\\ 0.142857 & -0.29032 & 1\end{bmatrix}\begin{Bmatrix}d_1\\ d_2\\ d_3\end{Bmatrix}=\begin{Bmatrix}12\\ 18\\ -6\end{Bmatrix}$$

Solving yields $d_1 = 12$, $d_2 = 14.57143$, and $d_3 = -3.48387$.

Back substitution:

$$\begin{bmatrix}7 & 2 & -3\\ 0 & 4.428571 & -2.14286\\ 0 & 0 & -6.19355\end{bmatrix}\begin{Bmatrix}x_1\\ x_2\\ x_3\end{Bmatrix}=\begin{Bmatrix}12\\ 14.57143\\ -3.48387\end{Bmatrix}$$

$$x_3=\frac{-3.48387}{-6.19355}=0.5625$$

$$x_2 = \frac{14.57143 - (-2.14286)(0.5625)}{4.428571} = 3.5625$$

$$x_1 = \frac{12 - (-3)(0.5625) - 2(3.5625)}{7} = 0.9375$$

10.5 The system can be written in matrix form as

$$[A] = \begin{bmatrix} 2 & -6 & -1 \\ -3 & -1 & 6 \\ -8 & 1 & -2 \end{bmatrix} \qquad \{b\} = \begin{Bmatrix} -38 \\ -34 \\ -20 \end{Bmatrix} \qquad [P] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Partial pivot:

$$[A] = \begin{bmatrix} -8 & 1 & -2 \\ -3 & -1 & 7 \\ 2 & -6 & -1 \end{bmatrix} \qquad [P] = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Compute factors:

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

Forward eliminate and store factors in zeros:

$$[LU] = \begin{bmatrix} -8 & 1 & -2 \\ 0.375 & -1.375 & 7.75 \\ -0.25 & -5.75 & -1.5 \end{bmatrix}$$

Pivot again

$$[LU] = \begin{bmatrix} -8 & 1 & -2 \\ -0.25 & -5.75 & -1.5 \\ 0.375 & -1.375 & 7.75 \end{bmatrix} \qquad [P] = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Compute factors:

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

Forward eliminate and store factor in zero:

$$[LU] = \begin{bmatrix} -8 & 1 & -2 \\ -0.25 & -5.75 & -1.5 \\ 0.375 & 0.23913 & 7.108696 \end{bmatrix}$$

Therefore, the *LU* decomposition is

$$[L]\{U\} = \begin{bmatrix} 1 & 0 & 0 \\ -0.25 & 1 & 0 \\ 0.375 & 0.23913 & 1 \end{bmatrix} \begin{bmatrix} -8 & 1 & -2 \\ 0 & -5.75 & -1.5 \\ 0 & 0 & 7.108696 \end{bmatrix}$$

Forward substitution. First multiply right-hand side vector $\{b\}$ by $[P]$ to give

$$[P]\{b\} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{Bmatrix} -38 \\ -34 \\ -20 \end{Bmatrix} = \begin{Bmatrix} -20 \\ -38 \\ -34 \end{Bmatrix}$$

Therefore,

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

$$d_1 = -20$$
$$d_2 = -38 - (-0.25)(-20) = -43$$
$$d_3 = -34 - 0.375(-20) - 0.23913(-43) = -16.21739$$

Back substitution:

$$\begin{bmatrix} -8 & 1 & -2 \\ 0 & -5.75 & -1.5 \\ 0 & 0 & 7.108696 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} -20 \\ -43 \\ -16.21739 \end{Bmatrix}$$

$$x_3 = \frac{-16.21739}{7.108696} = -2.28135$$
$$x_2 = \frac{-43 - (-1.5)(-2.28135)}{-5.75} = 8.073394$$
$$x_1 = \frac{-20 - 1(8.073394) - (-2)(-2.28135)}{-8} = 4.079511$$

10.6 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. 10.3

```
>> 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
```

10.7 The result of Example 10.4 can be substituted into Eq. (10.14) 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

$$a_{11} = 2.44949^2 = 6.000001$$
$$a_{12} = 6.123724 \times 2.44949 = 15$$
$$a_{13} = 22.45366 \times 2.44949 = 55.00002$$
$$a_{21} = 2.44949 \times 6.123724 = 15$$
$$a_{22} = 6.123724^2 + 4.1833^2 = 54.99999$$
$$a_{22} = 22.45366 \times 6.123724^2 + 20.9165 \times 4.1833 = 225$$
$$a_{31} = 2.44949 \times 22.45366 = 55.00002$$

$$a_{32} = 6.123724 \times 22.45366 + 4.1833 \times 20.9165 = 225$$
$$a_{33} = 22.45366^2 + 20.9165^2 + 6.110101^2 = 979.0002$$

10.8 (a) For the first row (i = 1), Eq. (10.15) is employed to compute

$$u_{11} = \sqrt{a_{11}} = \sqrt{8} = 2.828427$$

Then, Eq. (10.16) can be used to determine

$$u_{12} = \frac{a_{12}}{u_{11}} = \frac{20}{2.828427} = 7.071068$$
$$u_{13} = \frac{a_{13}}{u_{11}} = \frac{16}{2.828427} = 5.656854$$

For the second row (i = 2),

$$u_{22} = \sqrt{a_{22} - u_{12}^2} = \sqrt{80 - (7.071068)^2} = 5.477226$$
$$u_{23} = \frac{a_{23} - u_{12}u_{13}}{u_{22}} = \frac{50 - 7.071068(5.656854)}{5.477226} = 1.825742$$

For the third row (i = 3),

$$u_{33} = \sqrt{a_{33} - u_{13}^2 - u_{23}^2} = \sqrt{60 - (5.656854)^2 - (1.825742)^2} = 4.966555$$

Thus, the Cholesky decomposition yields

$$[U] = \begin{bmatrix} 2.828427 & 7.071068 & 5.656854 \\ & 5.477226 & 1.825742 \\ & & 4.966555 \end{bmatrix}$$

The validity of this decomposition can be verified by substituting it and its transpose into Eq. (10.14) to see if their product yields the original matrix [A].

```
>> U = [2.828427 7.071068 5.656854; 0 5.477226 1.825742; 0 0 4.966555];
>> U'*U
ans =
    8.0000   20.0000   16.0000
   20.0000   80.0000   50.0000
   16.0000   50.0000   60.0000
```

(b)

```
>> A = [8 20 16;20 80 50;16 50 60];
>> U = chol(A)
U =
    2.8284    7.0711    5.6569
         0    5.4772    1.8257
         0         0    4.9666
```

(c) The solution can be obtained by hand or by MATLAB. Using MATLAB:

```
>> b = [100;250;100];
>> d=U'\b
d =
   35.3553
    0.0000
  -20.1347
>> x=U\d
x =
   17.2297
    1.3514
   -4.0541
```

10.9 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. 10.8

```
>> A = [8 20 16;20 80 50;16 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
```

10.10 The system can be written in matrix form as

$$[A]=\begin{bmatrix}3 & -2 & 1\\ 2 & 6 & -4\\ -8 & -2 & 5\end{bmatrix} \qquad \{b\}=\begin{Bmatrix}-10\\ 44\\ -26\end{Bmatrix} \qquad [P]=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

Partial pivot:

$$[A]=\begin{bmatrix}-8 & 1 & -2\\ 2 & -6 & -1\\ 3 & -2 & 1\end{bmatrix} \qquad [P]=\begin{bmatrix}0 & 0 & 1\\ 0 & 1 & 0\\ 1 & 0 & 0\end{bmatrix}$$

Compute factors:

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

Forward eliminate and store factors in zeros:

$$[LU]=\begin{bmatrix}-8 & -2 & 5\\ -0.25 & 5.5 & -2.75\\ -0.375 & -2.75 & 2.875\end{bmatrix}$$

Pivot again

$$[LU]=\begin{bmatrix}-8 & -2 & 5\\ -0.375 & -2.75 & 2.875\\ -0.25 & 5.5 & -2.75\end{bmatrix} \qquad [P]=\begin{bmatrix}0 & 0 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}$$

Compute factors:

$f_{32} = 5.5/(-2.75) = -2$

Forward eliminate and store factor in zero:

$$[LU]=\begin{bmatrix}-8 & -2 & 5\\ -0.375 & -2.75 & 2.875\\ -0.25 & -2 & 3\end{bmatrix}$$

Therefore, the *LU* decomposition is

$$[L]\{U\}=\begin{bmatrix}1 & 0 & 0\\ -0.375 & 1 & 0\\ -0.25 & -2 & 1\end{bmatrix}\begin{bmatrix}-8 & -2 & 5\\ 0 & -2.75 & 2.875\\ 0 & 0 & 3\end{bmatrix}$$

Forward substitution. First multiply right-hand side vector $\{b\}$ by $[P]$ to give

$$[P]\{b\}=\begin{bmatrix}0 & 0 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}\begin{Bmatrix}-10\\ 44\\ -26\end{Bmatrix}=\begin{Bmatrix}-26\\ -10\\ 44\end{Bmatrix}$$

Therefore,

$$\begin{bmatrix} 1 & 0 & 0 \\ -0.375 & 1 & 0 \\ -0.25 & -2 & 1 \end{bmatrix} \{d\} = \begin{Bmatrix} -26 \\ -10 \\ 44 \end{Bmatrix}$$

$$d_1 = -26$$
$$d_2 = -10 - (-0.375)(-26) = -19.75$$
$$d_3 = 44 - (-0.25)(-26) - (-2)(-19.75) = -2$$

Back substitution:

$$\begin{bmatrix} -8 & -2 & 5 \\ 0 & -2.75 & 2.875 \\ 0 & 0 & 3 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} -26 \\ -19.75 \\ -2 \end{Bmatrix}$$

$$x_3 = \frac{-2}{3} = -0.66667$$
$$x_2 = \frac{-19.75 - 2.875(-0.66667)}{-2.75} = 6.484848$$
$$x_1 = \frac{-26 - 5(0.666667) + 2(6.484848)}{-8} = 1.212121$$

10.11 (a) Multiply first row by $f_{21} = 3/8 = 0.375$ and subtract the result from the second row to give

$$\begin{bmatrix} 8 & 5 & 1 \\ 0 & 5.125 & 3.625 \\ 2 & 3 & 9 \end{bmatrix}$$

Multiply first row by $f_{31} = 2/8 = 0.25$ and subtract the result from the third row to give

$$\begin{bmatrix} 8 & 5 & 1 \\ 0 & 5.125 & 3.625 \\ 0 & 1.75 & 8.75 \end{bmatrix}$$

Multiply second row by $f_{32} = 1.75/5.125 = 0.341463$ and subtract the result from the third row to give

$$[U] = \begin{bmatrix} 8 & 5 & 1 \\ 0 & 5.125 & 3.625 \\ 0 & 0 & 7.512195 \end{bmatrix}$$

As indicated, this is the U matrix. The L matrix is simply constructed from the f's as

$$[L]=\begin{bmatrix} 1 & 0 & 0 \\ 0.375 & 1 & 0 \\ 0.25 & 0.341463 & 1 \end{bmatrix}$$

Merely multiply $[L][U]$ to yield the original matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0.375 & 1 & 0 \\ 0.25 & 0.341463 & 1 \end{bmatrix}\begin{bmatrix} 8 & 5 & 1 \\ 0 & 5.125 & 3.625 \\ 0 & 0 & 7.512195 \end{bmatrix}=\begin{bmatrix} 8 & 5 & 1 \\ 3 & 7 & 4 \\ 2 & 3 & 9 \end{bmatrix}$$

(b) The determinant is equal to the product of the diagonal elements of $[U]$:

$$D=8\times 5.125\times 7.51295=308$$

(c) Solution with MATLAB:

```
>> A=[8 5 1;3 7 42;2 3 9];
>> [L,U]=lu(A)
L =
    1.0000         0         0
    0.3750    1.0000         0
    0.2500    0.3415    1.0000
U =
    8.0000    5.0000    1.0000
         0    5.1250    3.6250
         0         0    7.5122
>> L*U
ans =
     8     5     1
     3     7     4
     2     3     9
>> det(A)
ans =
   308
```

10.12 (a) The determinant is equal to the product of the diagonal elements of $[U]$:

$$D=3\times 7.3333\times 3.6364=80$$

(b) Forward substitution:

```
>> L=[1 0 0;0.6667 1 0;-0.3333 -0.3636 1];
>> U=[3 -2 1;0 -7.3333 -4.6667;0 0 3.6364];
>> b=[-10 50 -26]';
>> d=L\b
d =
  -10.0000
   56.6670
   -8.7289
```

Back substitution:

```
>> x=U\d
x =
   -6.6664
   -6.1998
```

```
   -2.4004
```

10.13 Using MATLAB:

```
>> A=[2 -1 0;-1 2 -1;0 -1 2];
>> U=chol(A)

U =
    1.4142   -0.7071         0
         0    1.2247   -0.8165
         0         0    1.1547
```

The result can be validated by

```
>> U'*U

ans =
    2.0000   -1.0000         0
   -1.0000    2.0000   -1.0000
         0   -1.0000    2.0000
```

10.14 Using MATLAB:

```
>> A=[9 0 0;0 25 0;0 0 16];
>> U=chol(A)

U =
     3     0     0
     0     5     0
     0     0     4
```

Thus, the factorization of this diagonal matrix consists of another diagonal matrix where the elements are the square root of the original. This is consistent with Eqs. (10.15) and (10.16), which for a diagonal matrix reduce to

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

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

CHAPTER 11

11.1 The matrix to be evaluated is

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

First, compute the *LU* decomposition. 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 $\{d\}^T = [1\ 0.3\ -0.055556]$. 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 $\{d\}^T = [0\ 1\ 0.148148]$, 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 $\{b\}^T = [0\ 0\ 1]$ to solve for $\{d\}^T = \lfloor 0 \quad 0 \quad 1 \rfloor$, 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
```

11.2 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 $\{d\}^T$ = [1 0.25 -0.434783]. 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 $\{d\}^T$ = [0 1 -0.23913], 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 $\{b\}^T = [0\ 0\ 1]$ to solve for $\{d\}^T = [0\ 0\ 1]$, and the results are used with $[U]$ to determine $\{x\}$ by back substitution to generate the third column of the matrix inverse,

$$[A]^{-1} = \begin{bmatrix} -0.115282 & -0.013405 & -0.034853 \\ -0.029491 & -0.16622 & -0.032172 \\ -0.053619 & -0.029491 & 0.123324 \end{bmatrix}$$

11.3 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 = [4000 1500 2400]';
>> format short
>> c = AI*b

c =
  339.1710
  251.1917
  333.9896
```

(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

$$\Delta c_3 = 0.0259067(500) + 0.009326(250) = 12.9534 + 2.3316 = 15.285 \frac{\text{g}}{\text{m}^3}$$

11.4 The mass balances can be written and the result written in matrix form as

$$\begin{bmatrix} 9 & 0 & -3 & 0 & 0 \\ -4 & 4 & 0 & 0 & 0 \\ 0 & -2 & 9 & 0 & 0 \\ 0 & -1 & -6 & 9 & -2 \\ -5 & -1 & 0 & 0 & 6 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{Bmatrix} = \begin{Bmatrix} Q_{01}c_{01} \\ 0 \\ Q_{03}c_{03} \\ 0 \\ 0 \end{Bmatrix}$$

MATLAB can then be used to determine the matrix inverse

```
>> Q = [9 0 -3 0 0;-4 4 0 0 0;0 -2 9 0 0;0 -1 -6 9 -2;-5 -1 0 0 6];
>> inv(Q)
ans =

    0.1200    0.0200    0.0400         0         0
    0.1200    0.2700    0.0400         0         0
    0.0267    0.0600    0.1200         0         0
    0.0578    0.0837    0.0933    0.1111    0.0370
    0.1200    0.0617    0.0400         0    0.1667
```

The concentration in reactor 5 can be computed using the elements of the matrix inverse as in,

$$c_5 = a_{51}^{-1}Q_{01}c_{01} + a_{53}^{-1}Q_{03}c_{03} = 0.1200(6)10 + 0.0400(7)20 = 7.2 + 2.8 = 10$$

11.5 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,

$$F_1 = a_{12}^{-1}F_{1,v} + a_{15}^{-1}F_{3,h} = 0.5(-2000) + 0(-500) = -1000 + 0 = -1000$$

$$F_2 = a_{22}^{-1}F_{1,v} + a_{25}^{-1}F_{3,h} = -0.433(-2000) + 1(-500) = 866 - 500 = 366$$
$$F_3 = a_{32}^{-1}F_{1,v} + a_{35}^{-1}F_{3,h} = 0.866(-2000) + 0(-500) = -1732 + 0 = -1732$$

11.6 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
```

11.7 Prob. 11.2:

```
>> A = [-8 1 -2;2 -6 -1;-3 -1 7];
>> norm(A,'fro')

ans =
    13

>> norm(A,inf)

ans =
    11
```

Prob. 11.3:

```
>> A = [15 -3 -1;-3 18 -6;-4 -1 12]
>> norm(A,'fro')

ans =
   27.6586

>> norm(A,inf)

ans =
    27
```

11.8 (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
```

11.9 (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
```

11.10 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.

11.11 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.

11.12 (a) The solution can be developed using your own software or a package. For example, using MATLAB,

```
>> A=[13.422 0 0 0;
-13.422 12.252 0 0;
0 -12.252 12.377 0;
0 0 -12.377 11.797];
>> W=[750.5 300 102 30]';
>> AI=inv(A)

AI =
    0.0745         0         0         0
    0.0816    0.0816         0         0
    0.0808    0.0808    0.0808         0
    0.0848    0.0848    0.0848    0.0848

>> C=AI*W
C =
   55.9157
   85.7411
   93.1163
  100.2373
```

(b) The element of the matrix that relates the concentration of Havasu (lake 4) to the loading of Powell (lake 1) is $a_{41}^{-1} = 0.084767$. This value can be used to compute how much the loading to Lake Powell must be reduced in order for the chloride concentration of Lake Havasu to be 75 as

$$\Delta W_1 = \frac{\Delta c_4}{a_{41}^{-1}} = \frac{100.2373 - 75}{0.084767} = 297.725$$

(c) First, normalize the matrix to give

$$[A] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -0.91283 & 0 & 0 \\ 0 & -0.9899 & 1 & 0 \\ 0 & 0 & 1 & -0.95314 \end{bmatrix}$$

The column-sum norm for this matrix is 2. The inverse of the matrix can be computed as

$$[A]^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1.095495 & -1.09549 & 0 & 0 \\ 1.084431 & -1.08443 & 1 & 0 \\ 1.137747 & -1.13775 & 1.049165 & -1.04917 \end{bmatrix}$$

The column-sum norm for the inverse can be computed as 4.317672. The condition number is, therefore, 2(4.317672) = 8.635345. This means that less than 1 digit is suspect [$\log_{10}(8.635345) = 0.93628$]. Interestingly, if the original matrix is unscaled, the same condition number results.

11.13 (a) When MATLAB is used to determine the inverse, the following error message suggests that the matrix is ill-conditioned:

```
>> A=[1 2 3;4 5 6;7 8 9];
>> inv(A)
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.541976e-018.

ans =
  1.0e+016 *
   -0.4504    0.9007   -0.4504
    0.9007   -1.8014    0.9007
   -0.4504    0.9007   -0.4504

The high condition number reinforces this conclusion:

>> cond(A)
ans =
  3.8131e+016
```

(b) However, when one of the coefficients is changed slightly, the system becomes well-conditioned:

```
>> A=[1 2 3;4 5 6;7 8 9.1];
>> inv(A)
ans =
    8.3333  -19.3333   10.0000
  -18.6667   39.6667  -20.0000
   10.0000  -20.0000   10.0000

>> cond(A)
ans =
  994.8787
```

11.14 The five simultaneous equations can be set up as

$$\begin{aligned}
1.6\times10^{9}p_1 + 8\times10^{6}p_2 + 4\times10^{4}p_3 + 200p_4 + p_5 &= 0.746\\
3.90625\times10^{9}p_1 + 1.5625\times10^{7}p_2 + 6.25\times10^{4}p_3 + 250p_4 + p_5 &= 0.675\\
8.1\times10^{9}p_1 + 2.7\times10^{7}p_2 + 9\times10^{4}p_3 + 300p_4 + p_5 &= 0.616\\
2.56\times10^{10}p_1 + 6.4\times10^{7}p_2 + 16\times10^{4}p_3 + 400p_4 + p_5 &= 0.525\\
6.25\times10^{10}p_1 + 1.25\times10^{8}p_2 + 25\times10^{4}p_3 + 500p_4 + p_5 &= 0.457
\end{aligned}$$

MATLAB can then be used to solve for the coefficients,

```
>> format short g
>> A=[200^4 200^3 200^2 200 1
250^4 250^3 250^2 250 1
300^4 300^3 300^2 300 1
400^4 400^3 400^2 400 1
500^4 500^3 500^2 500 1]

A =
     1.6e+009       8e+006        40000          200            1
  3.9063e+009  1.5625e+007        62500          250            1
    8.1e+009     2.7e+007        90000          300            1
   2.56e+010     6.4e+007     1.6e+005          400            1
   6.25e+010    1.25e+008     2.5e+005          500            1

>> b=[0.746;0.675;0.616;0.525;0.457];
>> format long g
```

```
>> p=A\b
p =
     1.33333333333201e-012
    -4.53333333333155e-009
     5.29666666666581e-006
      -0.00317366666666649
           1.20299999999999

>> cond(A)
ans =
           11711898982423.4
```

Thus, because the condition number is so high, the system seems to be ill-conditioned. This implies that this might not be a very reliable method for fitting polynomials. Because this is generally true for higher-order polynomials, other approaches are commonly employed as will be described subsequently in Chap. 15.

11.15 (a) The balances for reactors 2 and 3 can be written as

$$Q_{2,in}c_{2,in} - Q_{2,1}c_2 + Q_{1,2}c_1 + Q_{3,2}c_3 - kV_2c_2 = 0$$
$$-Q_{3,2}c_3 - Q_{3,\text{out}}c_3 + Q_{1,3}c_1 - kV_3c_3 = 0$$

(b) The parameters can be substituted into the mass balances

$$100(10) - 5c_1 - 117c_1 + 22c_2 - 0.1(100)c_1 = 0$$
$$10(200) - 22c_2 + 5c_1 + 7c_3 - 0.1(50)c_2 = 0$$
$$-7c_3 - 110c_3 + 117c_1 - 0.1(150)c_3 = 0$$

Collecting terms

$$132c_1 - 22c_2 = 1000$$
$$-5c_1 + 27c_2 - 7c_3 = 2000$$
$$-117c_1 + 132c_3 = 0$$

or in matrix form

$$\begin{bmatrix} 132 & -22 & 0 \\ -5 & 27 & -7 \\ -117 & 0 & 132 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \end{Bmatrix} = \begin{Bmatrix} 1000 \\ 2000 \\ 0 \end{Bmatrix}$$

(c) Using MATLAB

```
>> A=[132 -22 0;-5 27 -7;-117 0 132];
>> [L,U]=lu(A)
L =
    1.0000         0         0
   -0.0379    1.0000         0
   -0.8864   -0.7452    1.0000
U =
  132.0000  -22.0000         0
         0   26.1667   -7.0000
         0         0  126.7834
```

(d)

```
>> b=[1;0;0]; d1=L\b
d1 =
   1.00000000000000
   0.03787878787879
   0.91459177764910
>> c1=U\d1
c1 =
   0.00813865862849
   0.00337740631637
   0.00721381105707
>> b=[0;1;0]; d2=L\b
d2 =
                  0
   1.00000000000000
   0.74522292993631
>> c2=U\d2
c2 =
   0.00663149962321
   0.03978899773926
   0.00587792012057
>> b=[0;0;1]; d3=L\b
d3 =
     0
     0
     1
>> c3=U\d3
c3 =
   0.00035167043456
   0.00211002260739
   0.00788746546094
>> Ainv=[c1 c2 c3]
Ainv =
   0.00813865862849   0.00663149962321   0.00035167043456
   0.00337740631637   0.03978899773926   0.00211002260739
   0.00721381105707   0.00587792012057   0.00788746546094
```

(e) (*i*)

```
>> b=[1000;2000;0];
>> c=Ainv*b
c =
  21.40165787490580
  82.95540179488936
  18.96965129821196
```

(*ii*) $\Delta c_1 = a_{12}^{-1} \times \Delta W_2 = (0.0066315) \times (-2000) = -13.263$

(*iii*) $c_3 = a_{31}^{-1} \times W_1 + a_{32}^{-1} \times W_2 = (0.0072138) \times (2000) + (0.0058779) \times (1000) = 20.3055$

11.16 (a) Here is a script to compute the matrix inverse:

```
K = [150 -100 0;-100 150 -50;0 -50 50]
KI = inv(K)

K =
   150  -100     0
  -100   150   -50
     0   -50    50
KI =
    0.0200    0.0200    0.0200
    0.0200    0.0300    0.0300
    0.0200    0.0300    0.0500
```

(b) $\Delta x_1 = a_{13}^{-1} \times \Delta m_1 g = 0.02 \times (100 \times 9.81) = 19.62$ m

(c) The position of the third jumper as a function of an additional force applied to the third jumper can be formulated in terms of the matrix inverse as

$$x_3 = x_{3,\text{original}} + a_{33}^{-1} \Delta F_3$$

which can be solved for

$$\Delta F_3 = \frac{x_3 - x_{3,\text{original}}}{a_{33}^{-1}}$$

Recall from Example 8.2 that the original position of the third jumper was 131.6130 m. Thus, the additional force is

$$\Delta F_3 = \frac{140 - 131.6130}{0.05} = 167.74 \text{ N}$$

11.17 The current can be computed as

$$i_{52} = a_{25}^{-1} V_6 + a_{26}^{-1} V_1$$

The matrix inverse can be computed as

```
>> 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];
>> AI=inv(A)

AI =
      0.84615      0.61538      0.76923      0.73077    0.0076923     0.030769
      0.11538     -0.46154    -0.076923     -0.17308     0.019231    -0.023077
     0.038462     -0.15385     -0.69231     -0.55769    -0.026923   -0.0076923
      0.15385      0.38462      0.23077      0.26923   -0.0076923    -0.030769
     0.038462     -0.15385      0.30769      0.44231    -0.026923   -0.0076923
     0.038462     -0.15385      0.30769     -0.55769    -0.026923   -0.0076923
```

Therefore,

$$i_{52} = 0.019231(200) - 0.023077(100) = 1.5385$$

11.18 (a) First, flow balances can be used to compute $Q_{13} = 100$, $Q_{23} = 100$, $Q_{34} = 150$, and $Q_{4,\text{out}} = 150$. Then, steady-state mass balances can be written for the rooms as

$$W_1 - Q_{12}c_1 - Q_{13}c_1 + E_{13}(c_3 - c_1)$$
$$W_2 + Q_{12}c_1 - Q_{23}c_2 + E_{23}(c_3 - c_2)$$
$$Q_{13}c_1 + Q_{23}c_2 - Q_{34}c_3 - Q_{3,\text{out}}c_3 + E_{13}(c_1 - c_3) + E_{23}(c_2 - c_3) + E_{34}(c_4 - c_3)$$
$$W_4 + Q_{34}c_3 - Q_{4,\text{out}}c_4 + E_{34}(c_3 - c_4)$$

Substituting parameters

$$150 - 50c_1 - 100c_1 + 50(c_3 - c_1)$$
$$2000 + 50c_1 - 100c_2 + 50(c_3 - c_2)$$
$$100c_1 + 100c_2 - 150c_3 - 50c_3 + 50(c_1 - c_3) + 50(c_2 - c_3) + 90(c_4 - c_3)$$
$$5000 + 150c_3 - 150c_4 + 90(c_3 - c_4)$$

Collecting terms and expressing in matrix form

$$\begin{bmatrix} 200 & 0 & -50 & 0 \\ -50 & 150 & -50 & 0 \\ -150 & -150 & 390 & -90 \\ 0 & 0 & -240 & 240 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{Bmatrix} = \begin{Bmatrix} 150 \\ 2000 \\ 0 \\ 5000 \end{Bmatrix}$$

(b) The matrix inverse can be computed and used to solve for the concentrations as

```
clear,clc
format short g
A=[200 0 -50 0;-50 150 -50 0;-150 -150 390 -90; 0 0 -240 240];
b=[150 2000 0 5000]';
AI=inv(A)
c=AI*b

AI =
      0.00625       0.00125       0.00125    0.00046875
      0.00375       0.00875     0.0020833    0.00078125
        0.005         0.005         0.005      0.001875
        0.005         0.005         0.005     0.0060417

c =
        5.7813
        21.969
        20.125
        40.958
```

(c) The concentration of the second room as a function of the change in load to the fourth room can be formulated in terms of the matrix inverse as

$$c_2 = c_{2,\text{original}} + a_{24}^{-1}\Delta W_4$$

which can be solved for

$$\Delta W_4 = \frac{c_2 - c_{2,\text{original}}}{a_{24}^{-1}}$$

Substituting values gives

$$\Delta W_4 = \frac{20 - 21.969}{0.00078125} = -2520$$