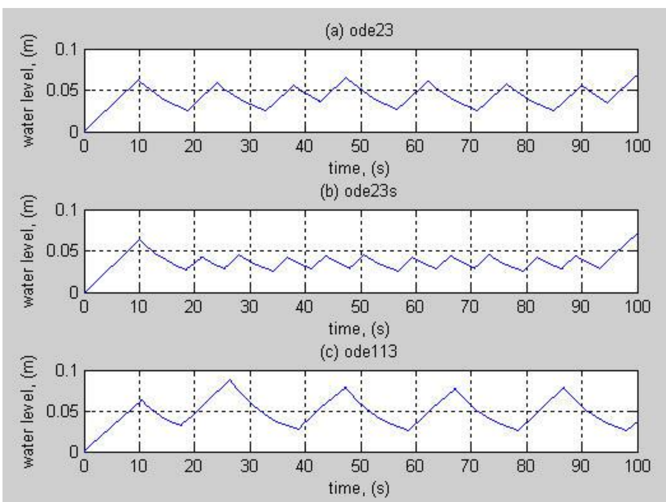
CHAPTER 23

23.1 Here is a script to implement the computations and create the plot:

```
global siphon
siphon = 0;
tspan = [0 100]; y0 = 0;
[tp1,yp1]=ode23(@Plinyode,tspan,y0);
subplot(3,1,1),plot(tp1,yp1)
xlabel('time, (s)')
ylabel('water level, (m)')
title('(a) ode23'),grid
[tp1,yp1]=ode23s(@Plinyode,tspan,y0);
subplot(3,1,2),plot(tp1,yp1)
xlabel('time, (s)')
ylabel('water level, (m)')
title('(b) ode23s'),grid
[tp1,yp1]=ode113(@Plinyode,tspan,y0);
subplot(3,1,3),plot(tp1,yp1)
xlabel('time, (s)')
ylabel('water level, (m)')
title('(c) ode113'),grid
```



23.2 The equations can first be written to account for the fact that recovered individuals can become susceptible:

$$\frac{dS}{dt} = -aSI + \rho R$$

$$\frac{dI}{dt} = aSI - rI$$

$$\frac{dR}{dt} = rI - \rho R$$

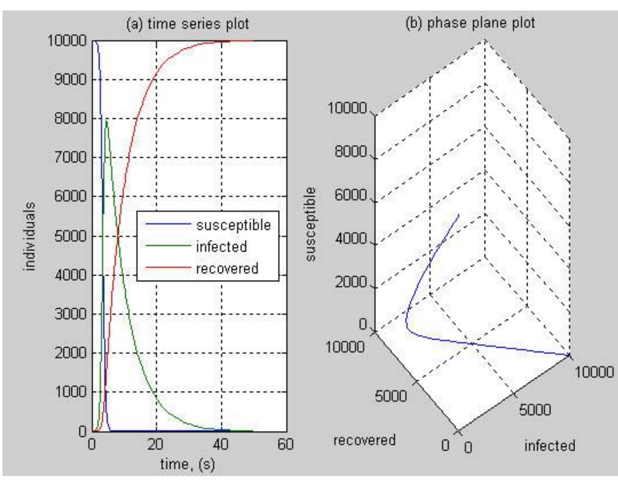
Then, we can first develop an M-file to hold these ODEs:

```
function dy=epidemic(t,y,a,r,rho)
dy=[-a*y(1)*y(2)+rho*y(3);a*y(1)*y(2)-r*y(2);r*y(2)-rho*y(3)];
```

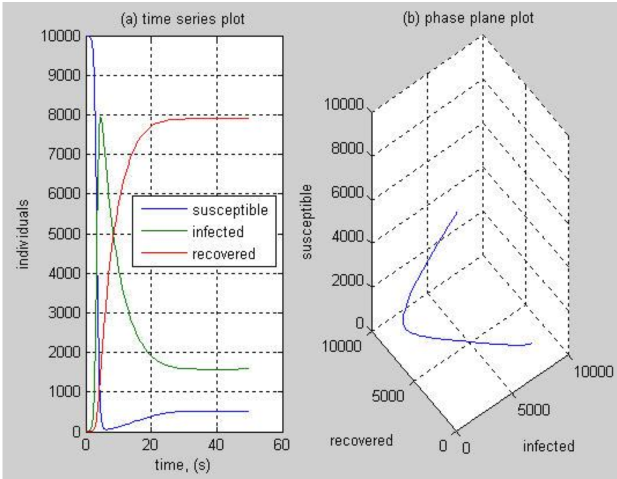
Here is a script to implement the computations and create the plots:

```
tspan = [0 50]; y0 = [10000 1 0];
[t,y]=ode23s(@epidemic,tspan,y0,[],0.002/7,0.15,0);
subplot(1,2,1),plot(t,y)
xlabel('time, (s)')
ylabel('individuals')
title('(a) time series plot'),grid
legend('susceptible','infected','recovered')
subplot(1,2,2),plot3(y(:,3),y(:,2),y(:,1))
xlabel('infected'),ylabel('recovered'),zlabel('susceptible')
title('(b) phase plane plot'),grid
pause
[t,y]=ode23s(@epidemic,tspan,y0,[],0.002/7,0.15,0.03);
subplot(1,2,1),plot(t,y)
xlabel('time, (s)')
ylabel('individuals')
title('(a) time series plot'),grid
legend('susceptible','infected','recovered')
subplot(1,2,2),plot3(y(:,3),y(:,2),y(:,1))
xlabel('infected'),ylabel('recovered'),zlabel('susceptible')
title('(b) phase plane plot'),grid
```

Here are the results for the first case where there is no re-susceptibility of the recovered individuals. Notice how after about 50 days the epidemic has burnt out.



In contrast, when the recovered become susceptible, there is a constant significant level of infected individuals:



23.3 First step:

Predictor:

$$y_1^0 = 5.222138 + [-0.5(4.143883) + e^{-2}]1 = 3.285532$$

Corrector:

$$y_1^1 = 4.143883 + \frac{-0.5(4.143883) + e^{-2} - 0.5(3.285532) + e^{-2.5}}{2} 0.5 = 3.269562$$

The corrector can be iterated to yield

j	y_{i+1}^j	$\lvert\varepsilon_a\rvert$,%
1	3.269562	
2	3.271558	0.061

Second step:

Predictor:

$$y_2^0 = 4.143883 + [-0.5(3.271558) + e^{-2.5}]1 = 2.590189$$

Corrector:

$$y_2^1 = 3.271558 + \frac{-0.5(3.271558) + e^{-2.5} - 0.5(2.590189) + e^{-3}}{2} 0.5 = 2.571807$$

The corrector can be iterated to yield

i	y_{i+1}^{i}	$\lvert\varepsilon_a\rvert$,%
1	2.571807	
2	2.574105	0.0893

23.4 Before solving, for comparative purposes, we can develop the analytical solution as

$$y = e^{\frac{t^3}{3} - t}$$

Thus, the true values being simulated in this problem are

t	y
0	1
0.25	0.782868
0.5	0.632337

The first step is taken with the fourth-order RK:

$$k_1 = f(0,1) = 1(0)^2 - 1 = -1$$
$$y(0.125) = 1 - 1(0.125) = 0.875$$

$$k_2 = f(0.125, 0.875) = -0.861328$$
$$y(0.125) = 1 - 0.861328(0.125) = 0.89233$$

$$k_3 = f(0.125, 0.89233) = -0.87839$$
$$y(0.25) = 1 - 0.87839(0.25) = 0.78040$$

$$k_4 = f(0.25, 0.78040) = -0.73163$$

$$\phi = \frac{-1 + 2(-0.861328 - 0.87839) - 0.73163}{6} = -0.86851$$

$$y(0.25) = 1 - 0.86851(0.25) = 0.7828723$$

This result compares favorably with the analytical solution. The second step can then be implemented with the non-self-starting Heun method:

Predictor:

$$y(0.5) = 1 + (0.7828723(0.25)^2 - 0.7828723)0.5 = 0.633028629$$

Corrector: (First iteration):

$$y(0.5) = 0.7828723 + \frac{-0.7339 + (0.633028629(0.5)^2 - 0.633028629)}{2} 0.25 = 0.63178298$$

Corrector: (Second iteration):

$$y(0.5) = 0.7828723 + \frac{-0.7339 + (0.63178298(0.5)^2 - 0.63178298)}{2} 0.25 = 0.63189976$$

The iterative process can be continued with the final result converging on 0.63188975.

23.5 **(a)** $h < 2/100{,}000 = 2\times 10^{-5}$.

(b) The implicit Euler can be written for this problem as

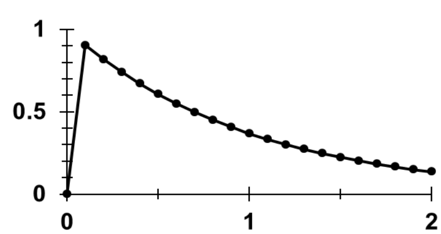
$$y_{i+1} = y_i + (-100{,}000 y_{i+1} + 99{,}999 e^{-t_{i+1}})h$$

which can be solved for

$$y_{i+1} = \frac{y_i + 99{,}999 e^{-t_{i+1}} h}{1 + 100{,}000 h}$$

The results of applying this formula for the first few steps are shown below. A plot of the entire solution is also displayed

t	*y*
0	0
0.1	0.904738
0.2	0.818731
0.3	0.740819
0.4	0.67032
0.5	0.606531



23.6 The implicit Euler can be written for this problem as

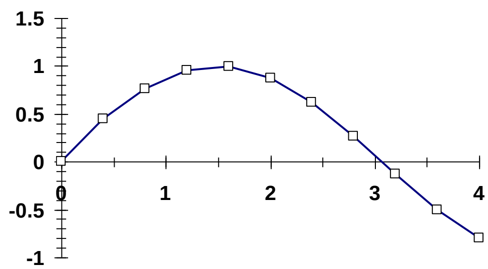
$$y_{i+1} = y_i + \left[30(\sin t_{i+1} - y_{i+1}) + 3\cos t_{i+1}\right]h$$

which can be solved for

$$y_{i+1} = \frac{y_i + 30\sin t_{i+1}h + 3\cos t_{i+1}h}{1+30h}$$

The results of applying this formula are tabulated and graphed below.

t	*y*	*t*	*y*	*t*	*y*	*t*	*y*
0	0	1.2	0.952306	2.4	0.622925	3.6	−0.50089
0.4	0.444484	1.6	0.993242	2.8	0.270163	4	−0.79745
0.8	0.760677	2	0.877341	3.2	−0.12525		



23.7 (a) The explicit Euler can be written for this problem as

$$x_{1,i+1} = x_{1,i} + (999x_{1,i} + 1999x_{2,i})h$$
$$x_{2,i+1} = x_{2,i} + (-1000x_{1,i} - 2000x_{2,i})h$$

Because the step-size is much too large for the stability requirements, the solution is unstable,

t	x_1	x_2	dx_1/dt	dx_2/dt
0	1	1	2998	−3000
0.05	150.9	−149	−147102	147100
0.1	−7204.2	7206	7207803	−7207805
0.15	353186	−353184	−3.5E+08	3.53E+08
0.2	−1.7E+07	17305943	1.73E+10	−1.7E+10

(b) The implicit Euler can be written for this problem as

$$x_{1,i+1} = x_{1,i} + (999x_{1,i+1} + 1999x_{2,i+1})h$$

$$x_{2,i+1} = x_{2,i} + (-1000x_{1,i+1} - 2000x_{2,i+1})h$$

or collecting terms

$$(1-999h)x_{1,i+1} - 1999hx_{2,i+1} = x_{1,i}$$
$$1000hx_{1,i+1} + (1+2000h)x_{2,i+1} = x_{2,i}$$

or substituting $h = 0.05$ and expressing in matrix format

$$\begin{bmatrix} -48.95 & -99.95 \\ 50 & 101 \end{bmatrix} \begin{Bmatrix} x_{1,i+1} \\ x_{2,i+1} \end{Bmatrix} = \begin{Bmatrix} x_{1,i} \\ x_{2,i} \end{Bmatrix}$$

Thus, to solve for the first time step, we substitute the initial conditions for the right-hand side and solve the 2×2 system of equations. The best way to do this is with LU decomposition since we will have to solve the system repeatedly. For the present case, because it's easier to display, we will use the matrix inverse to obtain the solution. Thus, if the matrix is inverted, the solution for the first step amounts to the matrix multiplication,

$$\begin{Bmatrix} x_{1,i+1} \\ x_{2,i+1} \end{Bmatrix} = \begin{bmatrix} 1.886088 & 1.86648 \\ -0.93371 & -0.9141 \end{bmatrix} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix} = \begin{Bmatrix} 3.752568 \\ -1.84781 \end{Bmatrix}$$

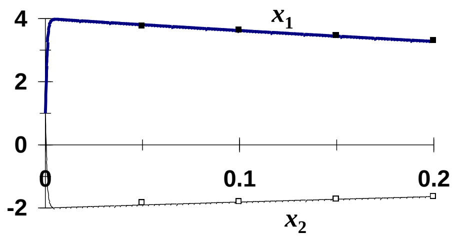
For the second step (from $x = 0.05$ to 0.1),

$$\begin{Bmatrix} x_{1,i+1} \\ x_{2,i+1} \end{Bmatrix} = \begin{bmatrix} 1.886088 & 1.86648 \\ -0.93371 & -0.9141 \end{bmatrix} \begin{Bmatrix} 3.752568 \\ -1.84781 \end{Bmatrix} = \begin{Bmatrix} 3.62878 \\ -1.81472 \end{Bmatrix}$$

The remaining steps can be implemented in a similar fashion to give

t	x_1	x_2
0	1	1
0.05	3.752568	−1.84781
0.1	3.62878	−1.81472
0.15	3.457057	−1.72938
0.2	3.292457	−1.64705

The results are plotted below, along with a solution with the explicit Euler using a step of 0.0005.



23.8 **(a)** The exact solution is

$$y = Ae^{5t} + t^2 + 0.4t + 0.08$$

If the initial condition at $t = 0$ is 0.08, $A = 0$,

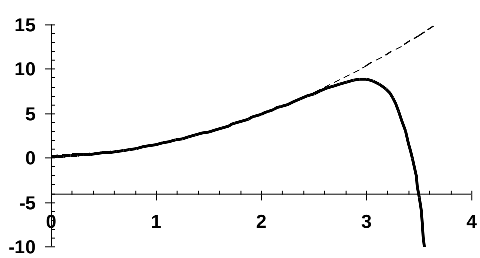
$$y = t^2 + 0.4t + 0.08$$

Note that even though the choice of the initial condition removes the positive exponential terms, it still lurks in the background. Very tiny round off errors in the numerical solutions bring it to the fore. Hence all of the following solutions eventually diverge from the analytical solution.

(b) 4th order RK. Here are the first few steps:

t	y
0	0.08
0.03125	0.093476
0.0625	0.108906
0.09375	0.126289
0.125	0.145625
0.15625	0.166914
0.1875	0.190156
0.21875	0.215351
0.25	0.242499

The plot shows the numerical solution (bold line) along with the exact solution (fine line).



(c)

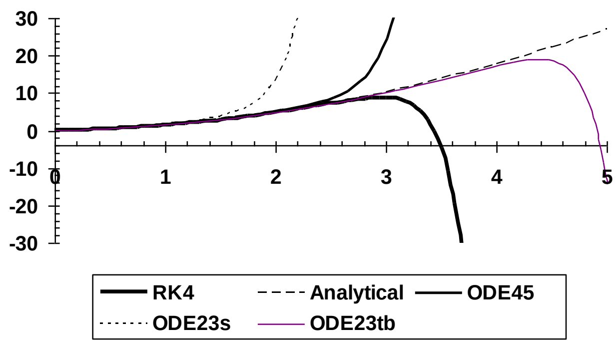
```
function yp = dy(t,y)
yp = 5*(y-t^2);

>> tspan = [0,5];
>> y0 = 0.08;
>> [t,y] = ode45(@dy,tspan,y0);
```

(d)

```
>> [t,y] = ode23s(@dy,tspan,y0);
```

(e)

```
>> [t,y] = ode23tb(@dy,tspan,y0);
```



23.9 **(a)** As in Example 20.5, the `humps` function can be integrated with the `quad` function as in

```
>> format long
>> quad(@humps,0,1)

ans =
  29.85832612842764
```

(b) Using `ode45` is based on recognizing that the evaluation of the definite integral

$$I = \int_a^b f(x)\, dx$$

is equivalent to solving the differential equation

$$\frac{dy}{dx} = f(x)$$

for $y(b)$ given the initial condition $y(a) = 0$. Thus, we must solve the following initial-value problem:

$$\frac{dy}{dx} = \frac{1}{(x-0.3)^2 + 0.01} + \frac{1}{(x-0.9)^2 + 0.04} - 6$$

where $y(0) = 0$. To do this with `ode45`, we must first set up an M-file to evaluate the right-hand side of the differential equation,

```
function dy = humpsODE(x,y)
dy = 1./((x-0.3).^2 + 0.01) + 1./((x-0.9).^2+0.04) - 6;
```

Then, the integral can be evaluated as

```
>> [x,y] = ode45(@humpsODE,[0 0.5 1],0);
>> disp([x,y])
                   0                   0
   0.50000000000000  21.78356481821654
   1.00000000000000  29.85525185285369
```

Thus, the integral estimate is within 0.01% of the estimate obtained with the `quad` function. Note that a better estimate can be obtained by using the `odeset` function to set a smaller relative tolerance as in

```
>> options = odeset('RelTol',1e-8);
>> [x,y] = ode45(@humpsODE,[0 0.5 1],0,options);
>> disp([x,y])
                   0                   0
   0.50000000000000  21.78683736423308
   1.00000000000000  29.85832514287622
```

23.10 The nonlinear model can be expressed as the following set of ODEs,

$$\frac{d\theta}{dt} = v$$
$$\frac{dv}{dt} = -\frac{g}{l}\sin\theta$$

where v = the angular velocity. A function can be developed to compute the right-hand-side of this pair of ODEs for the case where $g = 9.81$ and $l = 0.6$ m,

```
function dy = dpnon(t, y)
dy = [y(2);-9.81/0.6*sin(y(1))];
```

The linear model can be expressed as the following set of ODEs,

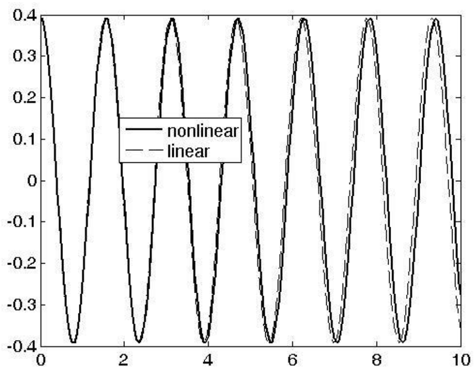
$$\frac{d\theta}{dt} = v \qquad \frac{dv}{dt} = -\frac{g}{l}\theta$$

A function can be developed as,

```
function dy = dplin(t, y)
dy = [y(2);-9.81/0.6*y(1)];
```

Then, the solution and plot can be obtained for the case where $\theta(0) = \pi/8$. Note that we only depict the displacement (θ or `y(1)`) in the plot

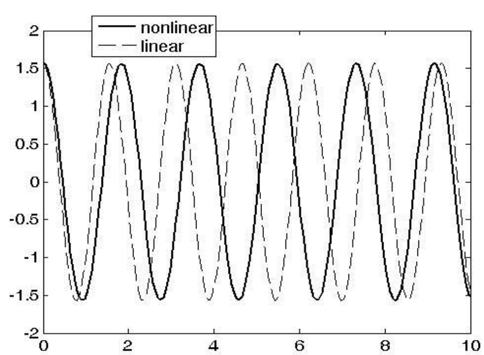
```
>> [tn yn] = ode45(@dpnon,[0 10],[pi/8 0]);
>> [tl yl] = ode45(@dplin,[0 10],[pi/8 0]);
>> plot(tn,yn(:,1),tl,yl(:,1),'--')
>> legend('nonlinear','linear')
```



You should notice two aspects of this plot. First, because the displacement is small, the linear solution provides a decent approximation of the more physically realistic nonlinear case. Second, the two solutions diverge as the computation progresses.

For the larger initial displacement ($\theta(0) = \pi/2$), the solution and plot can be obtained as,

```
>> [tn yn] = ode45(@dpnon,[0 10],[pi/2 0]);
>> [tl yl] = ode45(@dplin,[0 10],[pi/2 0]);
>> plot(tn,yn(:,1),tl,yl(:,1),'--')
>> legend('nonlinear','linear')
```



Because the linear approximation is only valid at small displacements, there are now clear and significant discrepancies between the nonlinear and linear cases that are exacerbated as the solution progresses.

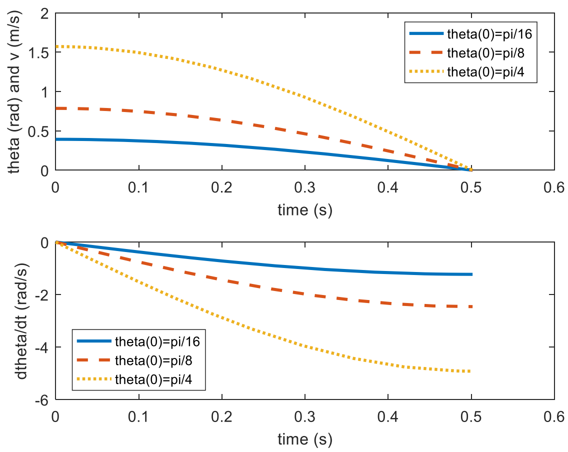
23.11

```
clear,clc,clf
format compact
opts=odeset('events',@linpendevent);
[ta,ya,tea,yea]=ode45(@linpend,[0 inf],[pi/8 0],opts,1);
tea,yea
[tb,yb,teb,yeb]=ode45(@linpend,[0 inf],[pi/4 0],opts,1);
teb,yeb
[tc,yc,tec,yec]=ode45(@linpend,[0 inf],[pi/2 0],opts,1);
tec,yec
Tpa=4*tea
Tpb=4*teb
Tpc=4*tec
subplot(2,1,1)
plot(ta,ya(:,1),'-',tb,yb(:,1),'--
',tc,yc(:,1),':','LineWidth',2)
legend('theta(0)=pi/16','theta(0)=pi/8','theta(0)=pi/4','Locatio
n','Best')
xlabel('time (s)');ylabel('theta (rad) and v (m/s)')
subplot(2,1,2)
plot(ta,ya(:,2),'-',tb,yb(:,2),'--
',tc,yc(:,2),':','LineWidth',2)
legend('theta(0)=pi/16','theta(0)=pi/8','theta(0)=pi/4','Locatio
n','Best')
xlabel('time (s)');ylabel('dtheta/dt (rad/s)')

function [detect,stopint,direction]=linpendevent(t,y,varargin)
% Locate the time when height passes through zero
% and stop integration.
detect=y(1);   % Detect height = 0
stopint=1;     % Stop the integration
direction=0;   % Direction does not matter

function dydt=linpend(t,y,l)
% y(1) = theta and y(2) = dtheta/dt
grav=9.81;
dydt=[y(2);-grav/l*y(1)];
```

__**Results:**__

```
tea =
    0.5015
yea =
   -0.0000   -1.2301
teb =
    0.5015
yeb =
   -0.0000   -2.4602
tec =
    0.5015
```

```
yec =
   -0.0000   -4.9198
Tpa =
    2.0059
Tpb =
    2.0060
Tpc =
    2.0059
```



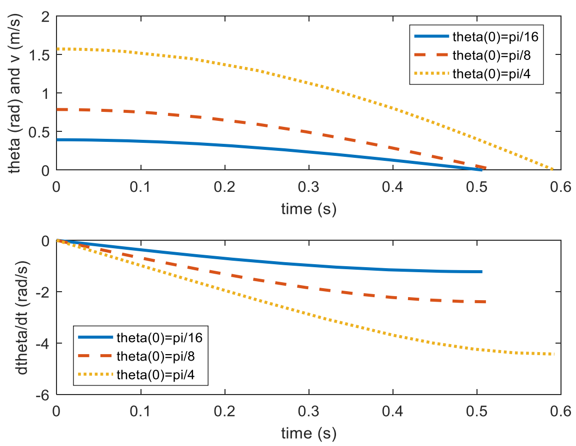
23.12

```
clear,clc,clf
format compact
opts=odeset('events',@nlinpendevent);
[ta,ya,tea,yea]=ode45(@nlinpend,[0 inf],[pi/8 0],opts,1);
tea,yea
[tb,yb,teb,yeb]=ode45(@nlinpend,[0 inf],[pi/4 0],opts,1);
teb,yeb
[tc,yc,tec,yec]=ode45(@nlinpend,[0 inf],[pi/2 0],opts,1);
tec,yec
Tpa=4*tea
Tpb=4*teb
Tpc=4*tec
subplot(2,1,1)
plot(ta,ya(:,1),'-',tb,yb(:,1),'--
',tc,yc(:,1),':','LineWidth',2)
legend('theta(0)=pi/16','theta(0)=pi/8','theta(0)=pi/4','Locatio
n','Best')
xlabel('time (s)');ylabel('theta (rad) and v (m/s)')
subplot(2,1,2)
plot(ta,ya(:,2),'-',tb,yb(:,2),'--
',tc,yc(:,2),':','LineWidth',2)
legend('theta(0)=pi/16','theta(0)=pi/8','theta(0)=pi/4','Locatio
n','Best')
xlabel('time (s)');ylabel('dtheta/dt (rad/s)')

function [detect,stopint,direction]=nlinpendevent(t,y,varargin)
% Locate the time when height passes through zero
```

```
% and stop integration.
detect=y(1);    % Detect height = 0
stopint=1;      % Stop the integration
direction=0;    % Direction does not matter

function dydt=nlinpend(t,y,l)
% y(1) = theta and y(2) = dtheta/dt
grav=9.81;
dydt=[y(2);-grav/l*sin(y(1))];
```

Results:

```
tea =
    0.5063
yea =
   -0.0000   -1.2222
teb =
    0.5215
yeb =
   -0.0000   -2.3981
tec =
    0.5919
yec =
   -0.0000   -4.4300
Tpa =
    2.0254
Tpb =
    2.0861
Tpc =
    2.3678
```



23.13 In MATLAB, the first step is to set up a function to hold the differential equations:

```
function dc = dcdtstiff(t, c)
dc = [-0.013*c(1)-1000*c(1)*c(3);-2500*c(2)*c(3);-0.013*c(1)-
1000*c(1)*c(3)-2500*c(2)*c(3)];
```

Then, an ODE solver like the function `ode45` can be implemented as in

```
>> tspan=[0,50];
>> y0=[1,1,0];
>> [t,y]=ode45(@dcdtstiff,tspan,y0);
```

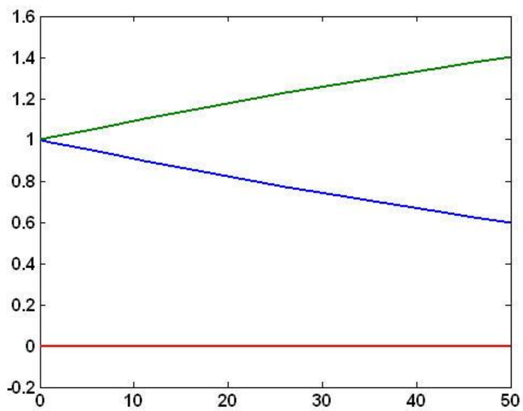
If this is done, the solution will take a relatively long time to compute the results. In contrast, because it is expressly designed to handle stiff systems, a function like `ode23s` yields results almost instantaneously.

```
>> [t,y]=ode23s(@dcdtstiff,tspan,y0);
```

In either case, a plot of the results can be developed as

```
>> plot(t,y)
```

Both plots will be identical as shown here



23.14 (a) Analytic solution:

$$y = \frac{1}{999}(1000\, e^{-x} - e^{-1000x})$$

(b) The second-order differential equation can be expressed as the following pair of first-order ODEs,

$$\frac{dy}{dx} = w$$
$$\frac{dw}{dx} = -1000y - 1001w$$

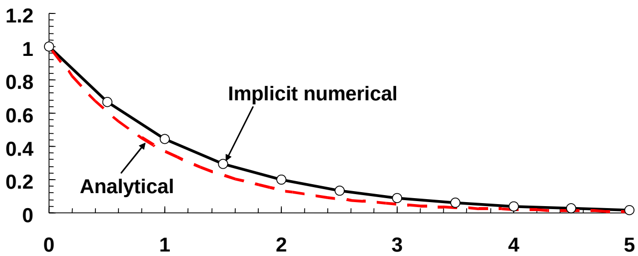
where $w = y'$. Using the same approach as described in Sec. 23.3, the following simultaneous equations need to be solved to advance each time step,

$$y_{i+1} - hw_{i+1} = y_i$$
$$1000hy_{i+1} + (1+1001h)w_{i+1} = w_i$$

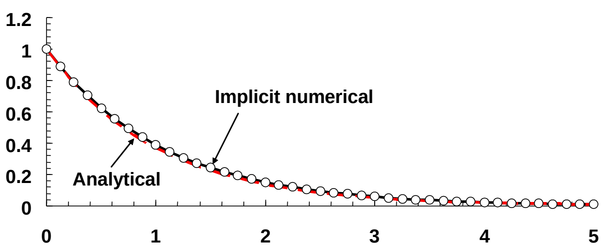
If these are implemented with a step size of 0.5, the following values are simulated

x	*y*	*w*
0	1	0
0.5	0.667332	–0.66534
1	0.444889	–0.44489
1.5	0.296593	–0.29659
2	0.197729	–0.19773
2.5	0.131819	–0.13182
3	0.087879	–0.08788
3.5	0.058586	–0.05859
4	0.039057	–0.03906
4.5	0.026038	–0.02604
5	0.017359	–0.01736

The results for *y* along with the analytical solution are displayed below:



Note that because we are using an implicit method the results are stable. However, also notice that the results are somewhat inaccurate. This is due to the large step size. If we use a smaller step size, the results will converge on the analytical solution. For example, if we use $h = 0.125$, the results are:

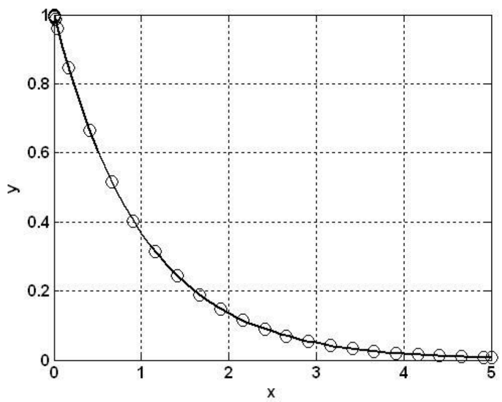


Finally, we can also solve this problem using one of the MATLAB routines expressly designed for stiff systems. To do this, we first develop a function to hold the pair of ODEs,

```
function dy = dydx(x, y)
dy = [y(2);-1000*y(1)-1001*y(2)];
```

Then the following session generates a plot of both the analytical and numerical solutions. As can be seen, the results are indistinguishable.

```
x=[0:.1:5];
y=1/999*(1000*exp(-x)-exp(-1000*x));
xspan=[0 5];
x0=[1 0];
[xx,yy]=ode23s(@dydx,xspan,x0);
plot(x,y,xx,yy(:,1),'o')
grid
xlabel('x')
ylabel('y')
```

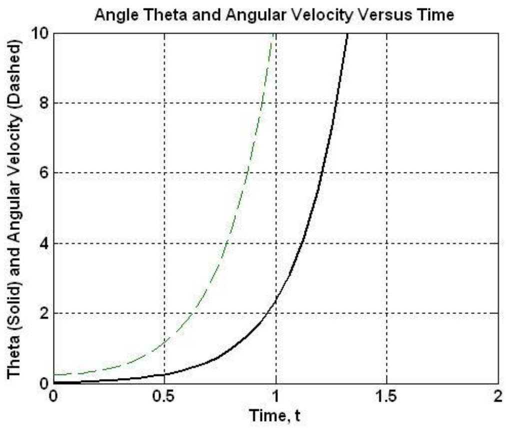


23.15 The second-order equation can be composed into a pair of first-order equations as

$$\frac{d\theta}{dt} = x \qquad \frac{dx}{dt} = \frac{g}{l}\theta$$

We can use MATLAB to solve this system of equations.

```
tspan=[0,5]';
x0=[0,0.25]';
[t,x]=ode45('dxdt',tspan,x0);
plot(t,x(:,1),t,x(:,2),'--')
grid
title('Angle Theta and Angular Velocity Versus Time')
xlabel('Time, t')
ylabel('Theta (Solid) and Angular Velocity (Dashed)')
axis([0 2 0 10])
zoom

function dx=dxdt(t,x)
dx=[x(2);(9.81/0.5)*x(1)];
```



23.16 Analytic solution: Take Laplace transform

$$sX - x(0) = -700X - \frac{1000}{s+1}$$

$$sX + 700X = x(0) - \frac{1000}{s+1}$$

$$X = \frac{x(0)}{s+700} - \frac{1000}{(s+1)(s+700)}$$

Substituting the initial condition and expanding the last term with partial fractions gives,

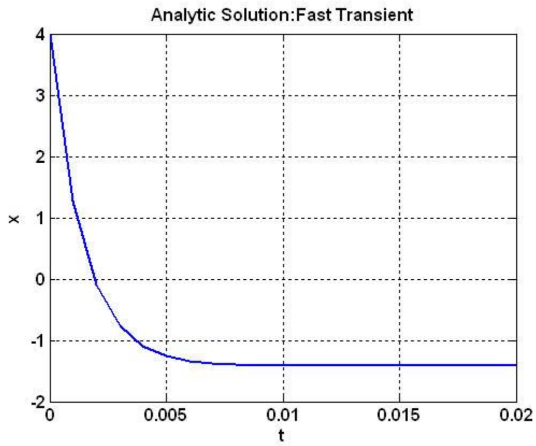
$$X = \frac{4}{s+700} - \frac{1.430615}{s+1} + \frac{1.430615}{s+700}$$

Taking inverse transforms yields

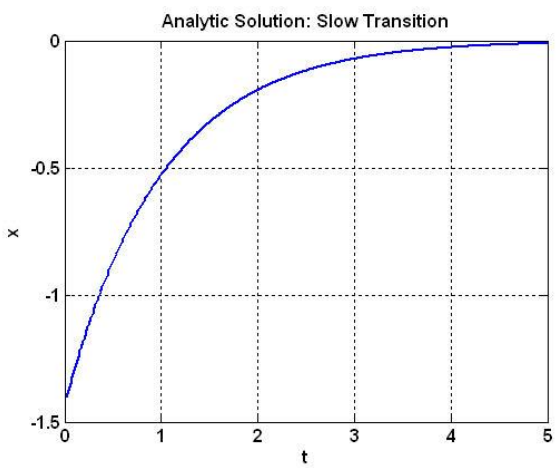
$$x = 5.430615e^{-700t} - 1.430615e^{-t}$$

MATLAB can be used to plot both the fast transient and the slow phases of the analytical solution.

```
t=[0:.001:.02];
x=5.430615*exp(-700*t)-1.430615*exp(-t);
plot(t,x)
grid, xlabel('t'), ylabel('x')
title('Analytic Solution:Fast Transient')
```




```
t=[0.02:.01:5];
x=5.430615*exp(-700*t)-1.430615*exp(-t);
plot(t,x)
grid, xlabel('t'),ylabel('x')
title('Analytic Solution: Slow Transient')
```



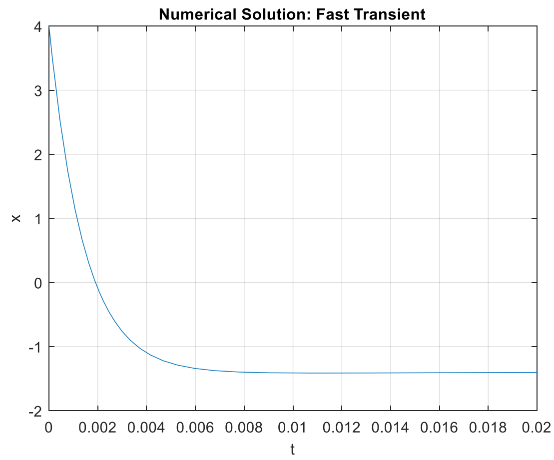
Numerical solution: First set up the function holding the differential equation:

```
function dx=dxdt(t,x)
dx=-700*x-1000*exp(-t);
```

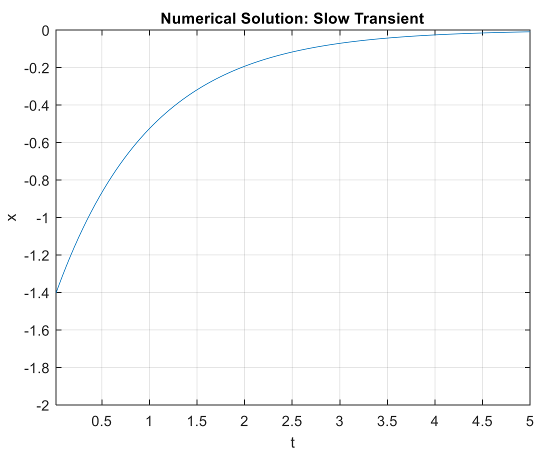
The following MATLAB code then generates the solutions and plots:

```
tspan=[0 5];
x0=[4];
[t,x]=ode23s(@dxdt,tspan,x0);
plot(t,x)
grid
xlabel('t')
ylabel('x')
```

```
title('Numerical Solution: Fast Transient')
axis([0 .02 -2 4])
```




```
tspan=[0 5];
x0=[4];
[t,x]=ode23s(@dxdt,tspan,x0);
plot(t,x)
grid
xlabel('t')
ylabel('x')
title('Numerical Solution: Slow Transient')
axis([0.02 5 -2 0])
```



23.17 (a) Analytic solution:

$$y = 1e^{-10t}$$

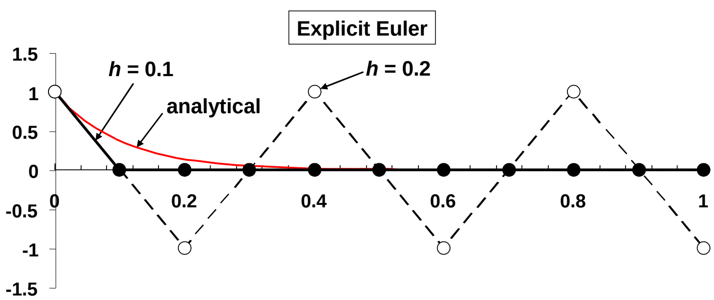
(b) Explicit Euler

$$y_{i+1} = y_i + (-10y_i)h$$

Here are the results for $h = 0.2$. Notice that the solution oscillates:

t	y	dy/dt
0	1	–10
0.2	–1	10
0.4	1	–10
0.6	–1	10
0.8	1	–10
1	–1	10

For $h = 0.1$, the solution plunges abruptly to 0 at $t = 0.1$ and then stays at zero thereafter. Both results are displayed along with the analytical solution below:



(c) Implicit Euler

$$y_{i+1} = \frac{y_i}{1+10h}$$

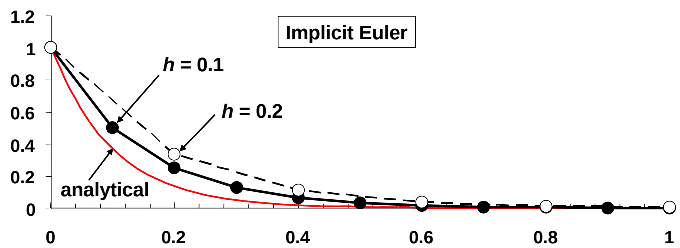
Here are the results for $h = 0.2$. Notice that although the solution is not very accurate, it is stable and declines monotonically in a similar fashion to the analytical solution.

t	y
0	1
0.2	0.333333333
0.4	0.111111111
0.6	0.037037037
0.8	0.012345679
1	0.004115226

For $h = 0.1$, the solution is also stable and tracks closer to the analytical solution.

t	*y*
0	1
0.1	0.5
0.2	0.25
0.3	0.125
0.4	0.0625
0.5	0.03125
0.6	0.015625
0.7	0.0078125
0.8	0.00390625
0.9	0.001953125
1	0.000976563

Both results are displayed along with the analytical solution below:

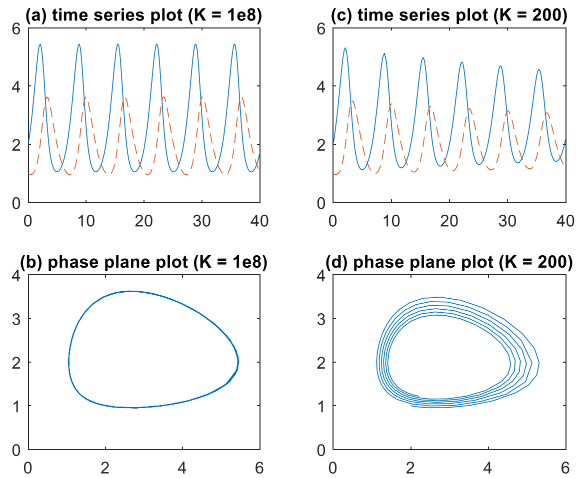


23.18 We can develop an M-file to compute the ODEs,

```
function yp = predpreylog(t,y,a,b,c,d,K)
yp = [a*(1-y(1)/K)*y(1)-b*y(1)*y(2);-c*y(2)+d*y(1)*y(2)];
```

The following script employs the `ode45` function to generate the solution.

```
a=1.2;b=0.6;c=0.8;d=0.3;
[t y] = ode45(@predpreylog,[0 40],[2 1],[],a,b,c,d,1e8);
subplot(2,2,1);plot(t,y(:,1),t,y(:,2),'--')
title('(a) time series plot (K = 1e8)')
subplot(2,2,3);plot(y(:,1),y(:,2))
title('(b) phase plane plot (K = 1e8)')
[t y] = ode45(@predpreylog, [0 40],[2 1],[],a,b,c,d,200);
subplot(2,2,2);plot(t,y(:,1),t,y(:,2),'--')
title('(c) time series plot (K = 200)')
subplot(2,2,4);plot(y(:,1),y(:,2))
title('(d) phase plane plot (K = 200)')
```



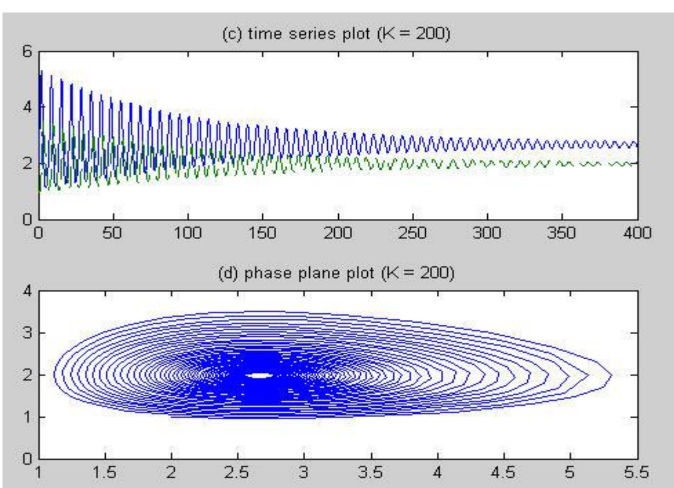
Two things are indicated by these plots:

1. The period of the oscillations seems to be unaffected by the introduction of a carrying capacity effect.
2. The amplitudes of the oscillations decrease with time when a meaningful carrying capacity is imposed.

The second result might suggest a further question of whether or not the oscillations would eventually stabilize. This can be investigated by extending the integration interval as in the following script:

```
a=1.2;b=0.6;c=0.8;d=0.3;
[t y] = ode45(@predpreylog,[0 400],[2 1],[],a,b,c,d,200);
subplot(2,1,1);plot(t,y(:,1),t,y(:,2),'--')
title('(a) time series plot (K = 200)')
subplot(2,1,2);plot(y(:,1),y(:,2))
title('(b) phase plane plot (K = 200)')
```

The resulting plot indicates that the solution does not have stable oscillations but seems to be converging on stable constant populations.



23.19 The second-order equations can be expressed as the following system of first-order ODEs,

$$\frac{dx_1}{dt} = x_3 \qquad \frac{dx_2}{dt} = x_4$$

$$\frac{dx_3}{dt} = -\frac{k_1}{m_1}(x_1 - L_1) + \frac{k_2}{m_1}(x_2 - x_1 - w_1 - L_2)$$

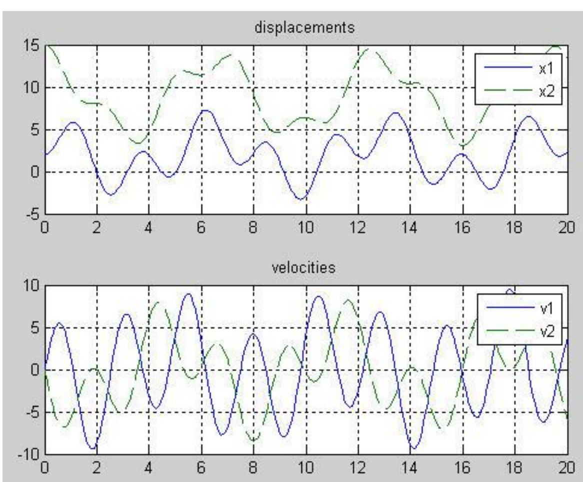
$$\frac{dx_4}{dt^2} = -\frac{k_2}{m_2}(x_2 - x_1 - w_1 - L_2)$$

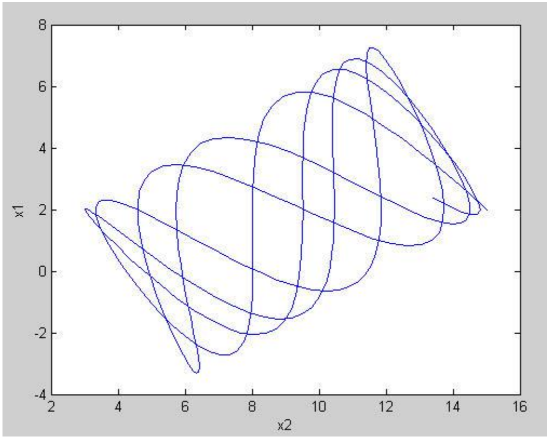
We can develop an M-file to compute them,

```
function dx = dxdtProb2319(t,x,k1,k2,m1,m2,w1,w2,L1,L2)
dx = [x(3);x(4);-k1/m1*(x(1)-L1)+k2/m1*(x(2)-x(1)-w1-L2);-
k2/m2*(x(2)-x(1)-w1-L2)];
```

The following script employs the `ode45` function to generate the solution.

```
k1=5;k2=5;m1=2;m2=2;w1=5;w2=5;L1=2;L2=2;
[t,x]=ode45(@dxdtProb2319,[0 20],[2 15 0
0],[],k1,k2,m1,m2,w1,w2,L1,L2);
subplot(2,1,1);plot(t,x(:,1),t,x(:,2),'--')
grid;title('displacements');legend('x1','x2')
subplot(2,1,2);plot(t,x(:,3),t,x(:,4),'--')
grid;title('velocities');legend('v1','v2')
pause
subplot(1,1,1),plot(x(:,2),x(:,1))
xlabel('x2');ylabel('x1')
```



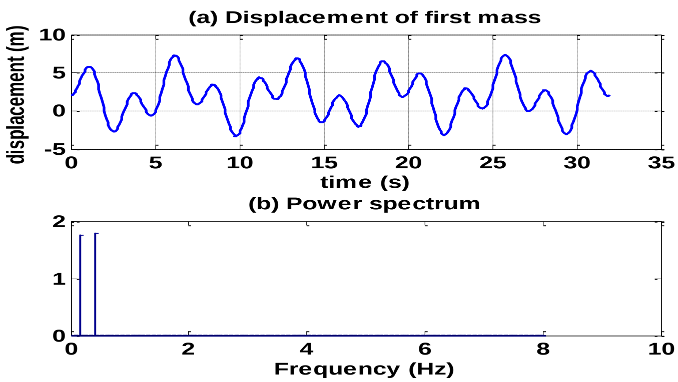


23.20 The following function can be used to hold the ODEs:

```
function dx = dxdtProb2320(t,x,k1,k2,m1,m2,w1,w2,L1,L2)
dx = [x(3);x(4);-k1/m1*(x(1)-L1)+k2/m1*(x(2)-x(1)-w1-L2); ...
      -k2/m2*(x(2)-x(1)-w1-L2)];
```

The following script employs the `ode45` function to generate the solution and then generates the power spectrum for the first mass's displacement:

```
clear,clf,clc
k1=5;k2=5;m1=2;m2=2;w1=5;w2=5;L1=2;L2=2;
[t,x]=ode45(@dxdtProb2320,[0:1/16:32],[2 15 0
0],[],k1,k2,m1,m2,w1,w2,L1,L2);
subplot(2,1,1);plot(t,x(:,1))
grid;title('(a) Displacement of first mass')
n=length(t)-1; dt=(max(t)-min(t))/n;fs=1/dt;nyquist=fs/2;
Y=fft(x(:,1))/n;
Y(1)=[];
% compute and display the power spectrum
f = (1:n/2)/(n/2)*nyquist;
Pyy = abs(Y(1:n/2)).^2;
subplot(2,1,2);
bar(f,Pyy)
title('(b) Power spectrum')
xlabel('Frequency (Hz)');
```



The power spectrum indicates peaks at 0.1563 and 0.4063 Hz. This result can be validated by determining the systems eigenvalues:

```
A=[k1/m1+k2/m1 -k2/m1;-k2/m2 k2/m2];
lambda=eig(A);
freq=sqrt(lambda)/(2*pi)
freq =
    0.1555
    0.4072
```

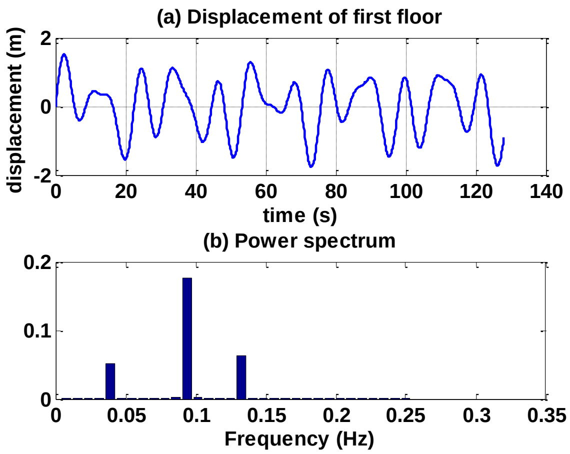
23.21 The following function can be used to hold the ODEs:

```
function dx = dxdtProb2321(t,x,m1,m2,m3,k1,k2,k3)
dx=[x(4);x(5);x(6);-k1/m1*x(1)+k2/m1*(x(2)-x(1)); ...
    k2/m2*(x(1)-x(2))+k3/m2*(x(3)-x(2)); ...
    k3/m3*(x(2)-x(3))];
```

The following script employs the `ode45` function to generate the solution and then generates the power spectrum for the floor’s displacement:

```
clear,clf,clc
m1=12000;m2=10000;m3=8000;
k1=3000;k2=2400;k3=1800;
tspan=[0:1/8:128];y0=[0 0 0 1 0 0];
[t,x]=ode45(@dxdtProb2321,tspan,y0,[],m1,m2,m3,k1,k2,k3);
subplot(2,1,1);plot(t,x(:,1))
xlabel('time (s)');ylabel('displacement (m)');
grid;title('(a) Displacement of first floor')
n=length(t)-1; dt=(max(t)-min(t))/n;fs=1/dt;nyquist=fs/2;
Y=fft(x(:,1))/n;
Y(1)=[];
% compute and display the power spectrum
f = (1:n/2)/(n/2)*nyquist;
Pyy = abs(Y(1:n/2)).^2;
subplot(2,1,2);
bar(f(1:n/32),Pyy(1:n/32))
title('(b) Power spectrum')
```

```
xlabel('Frequency (Hz)');
% Use eigenvalues to determine fundamental frequencies
```



The power spectrum indicates peaks at 0.0390625, 0.09375, and 0.1328125 Hz. This result can be validated by determining the systems eigenvalues:

```
A=[(k1+k2)/m1 -k2/m1 0; ...
   -k2/m2 (k2+k3)/m2 -k3/m2; ...
    0 -k3/m3 k3/m3];
lambda=eig(A);
freq=sqrt(lambda)/(2*pi)
freq =
    0.1330
    0.0927
    0.0380
```

23.22 Script:

```
clear,clc,clf
opts=odeset('events',@endevent2322);
y0=[-200 -20];
[t,y,te,ye]=ode45(@freefall,[0 inf],y0,opts,0.25,68.1);
te,ye
subplot(2,1,1)
plot(t,-y(:,1),'LineWidth',2)
title('height (m)')
ylabel('x (m)'),grid
subplot(2,1,2)
plot(t,-y(:,2),'LineWidth',2)
title('velocity (m/s)')
xlabel('time (s)');ylabel('v (m/s)'),grid
```

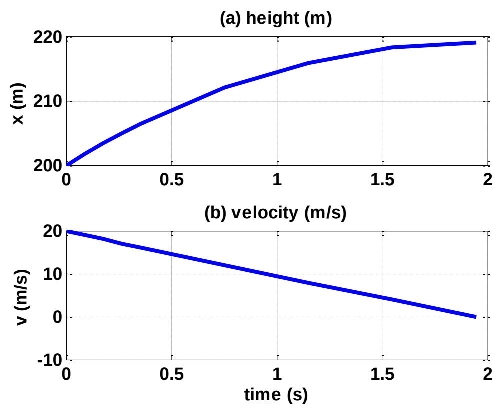
Supporting functions:

```
function dydt=freefall(t,y,cd,m)
```

```
% y(1) = x and y(2) = v
grav=9.81;
dydt=[y(2);grav-cd/m*y(2)*abs(y(2))];

function [detect,stopint,direction]=endevent2322(t,y,varargin)
% Locate the time when velocity is zero at maximum
% and stop integration.
detect=y(2);    % Detect height = 0
stopint=1;      % Stop the integration
direction=0;    % Direction does not matter
```

Results:

```
te =
    1.9456
ye =
 -218.9975    0.0000
```



23.23 Script:

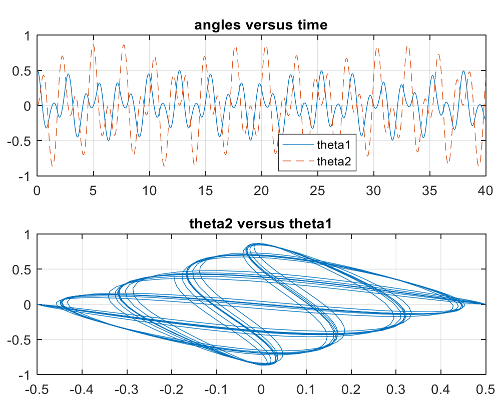
```
clear,clc,clf
L1=1;L2=1;m1=.5;m2=.25;tspan=[0 40];
% Case 1
y0=[.5 0 0 0];
DoubPendSolver(tspan,y0,m1,m2,L1,L2)
pause
% Case 2
y0=[1.5 0 0 0];
DoubPendSolver(tspan,y0,m1,m2,L1,L2)
```

Functions:

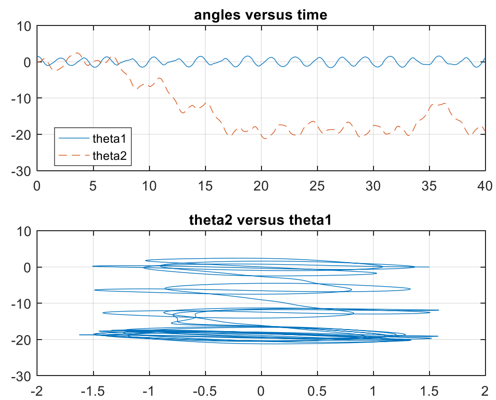
```
function DoubPendSolver(tspan,y0,m1,m2,L1,L2)
clf
[t,x]=ode45(@DoublePend,tspan,y0,[],m1,m2,L2,L2);
subplot(2,1,1);plot(t,x(:,1),t,x(:,2),'--')
grid;title('angles versus time');
legend('theta1','theta2','location','best')
subplot(2,1,2);plot(x(:,1),x(:,2))
grid;title('theta2 versus theta1');
pause
```

```
clf
x1=L1*sin(x(:,1));y1=-L1*cos(x(:,1));
x2=x1+L2*sin(x(:,2));y2=y1-L2*cos(x(:,2));
for i=1:length(x1)
  xx=[0 x1(i) x2(i)];yy=[0 y1(i) y2(i)];
  plot(xx,yy,'-
ok','LineWidth',2,'MarkerSize',10,'MarkerEdgeColor','k',...
                          'MarkerFaceColor','r')
  xlim([-L1-L2 L1+L2]);ylim([-L1-L2 L1+L2])
  pause(0.01)
end

function [dydx] = DoublePend(t,y,m1,m2,L1,L2)
% Nonlinear double pendulum ODEs
%   Function to compute the right-hand sides of ODEs describing
motion
%   of a nonlinear double pendulum
% Inputs:
%   y(1) & y(2) = angle of top and bottom pendulum, respectively
%   y(3) & y(4) = angular velocity of top and bottom pendulum,
respectively
%   m1 & m2 = mass of top and bottom pendulum, respectively
%   L1 & L2 = length of top and bottom pendulum, respectively
% Outputs:
%   dydt = right-hand sides of ODEs

g=9.81;  %gravitational acceleration
dydx = [y(3); y(4);...
(-g*(2*m1+m2)*sin(y(1))-m2*g*sin(y(1)-2*y(2))-...
  2*sin(y(1)-y(2))*m2*(y(4)^2*L2+y(3)^2*L1*cos(y(1)-y(2))))...
  /(L1*(2*m1+m2-m2*cos(2*y(1)-2*y(2))));
(2*sin(y(1)-y(2))*(y(3)^2*L1*(m1+m2)+g*(m1+m2)*cos(y(1))+...
  y(4)^2*L2*m2*cos(y(1)-y(2))))...
  /(L2*(2*m1+m2-m2*cos(2*y(1)-2*y(2))))];
end
```

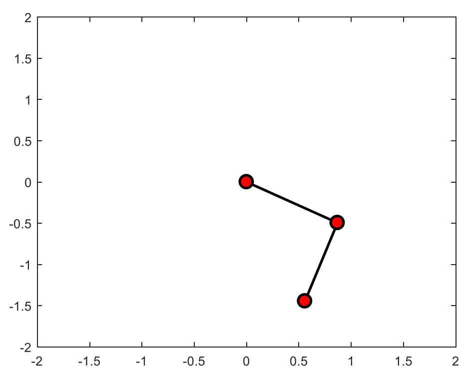
Case 1: Small displacement:



Case 2: Large displacement:



Animation: Both cases should look something like this with the big displacement case more chaotic.



23.24 (a)

$$m = m_P + m_G = m_P + \frac{\pi d_b^3}{6}\frac{P}{RT_a}$$

$$m\frac{dv}{dt} = F_B - F_G - F_P - F_D = \frac{\pi d_b^3}{6}\rho_a g - \frac{\pi d_b^3}{6}\frac{P}{RT_a}g - m_P g - \frac{1}{2}\rho_a\frac{\pi d_b^2}{4}C_d v^2$$

$$\frac{dv}{dt} = \frac{\pi d_b^3}{6m}\rho_a g - \frac{\pi d_b^3}{6m}\frac{P}{RT_a}g - \frac{m_P}{m}g - \frac{1}{8}\rho_a\frac{\pi d_b^2}{m}C_d v^2$$

$$\frac{dv}{dt} = \left[\frac{\pi d_b^3}{6\left(m_P + \frac{\pi d_b^3}{6}\frac{P}{RT_a}\right)}\left[\rho_a - \frac{P}{RT_a}\right] - \frac{m_P}{m_P + \frac{\pi d_b^3}{6}\frac{P}{RT_a}}\right]g - \frac{1}{8}\rho_a\frac{\pi d_b^2}{m_P + \frac{\pi d_b^3}{6}\frac{P}{RT_a}}C_d v^2$$

(b) Using Eq. (1) at steady state

$$0 = F_B - F_G - F_P - F_D = V_b\rho_a g - V_b\rho_g g - m_P g - \frac{1}{2}\rho_a A_b C_d v^2$$

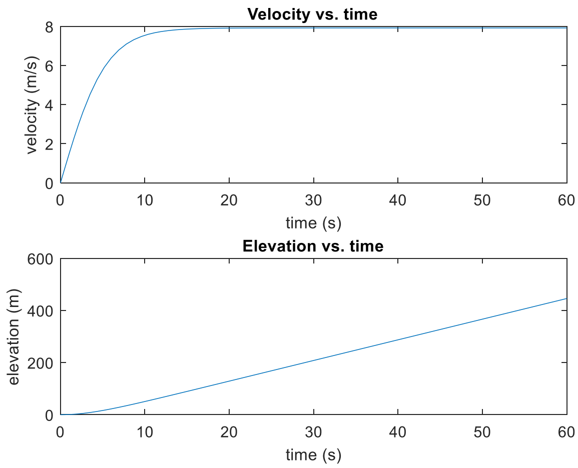
$$v = \sqrt{\frac{V_b\rho_a g - V_b\rho_g g - m_P g}{\frac{1}{2}\rho_a A_b C_d}} = \sqrt{\frac{2711(1.2)9.81 - 2711(0.9459)9.81 - 265(9.81)}{\frac{1}{2}(1.2)235(0.47)}} = \sqrt{\frac{4158.297}{66.28743}} = 7.92031\frac{\text{m}}{\text{s}}$$

(c)

```
% Hot Air Balloon Script
clear,clc,clf
format compact
global g
% set parameters
g=9.81;
r=8.65; % balloon radius
CD=0.47; % dimensionless drag coefficient
mP=265; % mass of payload
P=101300; % atmospheric pressure
Rgas=287; % Universal gas constant for dry air
TC=100; % air temperature
rhoa=1.2; % air density
ti=0; % initial time (s)
tf=60; % final time (s)
vi=0; % initial velocity
zi=0; % initial elevation

% precomputations
d = 2 * r; Ta = TC + 273.15; Ab = pi / 4 * d ^ 2;
Vb = pi / 6 * d ^ 3; rhog = P / Rgas / Ta; mG = Vb * rhog;
FB = Vb * rhoa * g; FG = mG * g; cdp = rhoa * Ab * CD / 2;

% compute times, velocities and elevations
tspan=[ti tf]; yinit=[vi zi];
[t,y]=ode45(@dydt2324c,tspan,yinit,[],FB,FG,mG,cdp,mP);

% Display results
fprintf('      t(s)      v(m/s)      z(m/s)\n');
xx=[t y];
fprintf('%10.3f %10.3f %10.3f\n',xx');

% Plot results
subplot(2,1,1)
plot(t,y(:,1))
title('Velocity vs. time')
xlabel('time (s)'),ylabel('velocity (m/s)')
subplot(2,1,2)
plot(t,y(:,2))
title('Elevation vs. time')
xlabel('time (s)'),ylabel('elevation (m)')

function dy=dydt2324c(t,y,FB,FG,mG,cdp,mP)
global g
dy = [(FB - FG) / (mP + mG) - mP * g / (mP + mG) - ...
```

```
                    cdp / (mP + mG) * y(1) ^ 2; y(1)];
```

```
   t(s)       v(m/s)      z(m/s)
   0.000       0.000       0.000
   0.000       0.000       0.000
   0.000       0.000       0.000
   0.000       0.000       0.000
   0.000       0.000       0.000
   0.000       0.000       0.000
   0.000       0.001       0.000
   0.001       0.001       0.000
   0.001       0.001       0.000
   0.002       0.002       0.000
   0.003       0.004       0.000
   0.003       0.005       0.000
   0.004       0.006       0.000
    .  .        .  .        .  .
    .  .        .  .        .  .
    .  .        .  .        .  .
  53.943       7.920     397.677
  55.443       7.920     409.557
  56.943       7.920     421.438
  57.707       7.920     427.491
  58.472       7.920     433.544
  59.236       7.920     439.597
  60.000       7.920     445.650
```

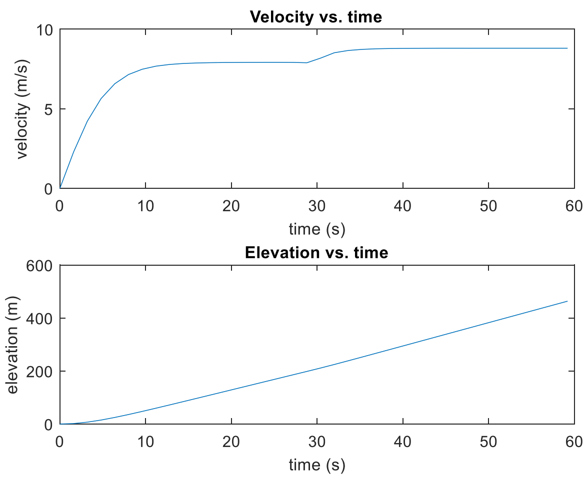


Notice that the terminal velocity (7.92) confirms the result computed analytically in (b). Such checks are useful in verifying that you numerical results are valid.

23.25

```
% Hot Air Balloon Script
clear,clc,clf, format compact
global g
g=9.81;
% set parameters
```

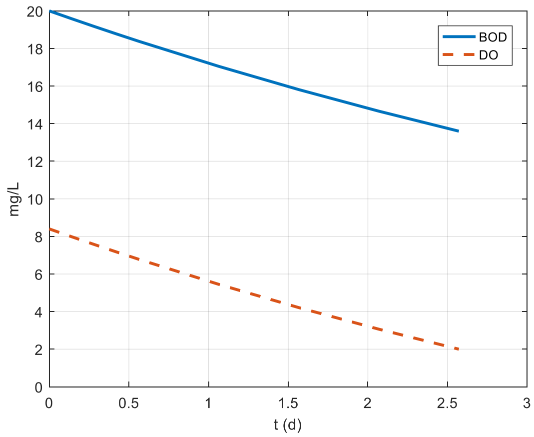
```
r=8.65; % balloon radius
CD=0.47; % dimensionless drag coefficient
mP=265; % mass of payload
P=101300; % atmospheric pressure
Rgas=287; % Universal gas constant for dry air
TC=100; % air temperature
rhoa=1.2; % air density
zd=200; % elevation at which mass is jettisoned
md=100; % mass jettisoned
ti=0; % initial time (s)
tf=60; % final time (s)
vi=0; % initial velocity
zi=0; % initial elevation
dt=1.6; % integration time step
% precomputations
d = 2 * r; Ta = TC + 273.15; Ab = pi / 4 * d ^ 2;
Vb = pi / 6 * d ^ 3; rhog = P / Rgas / Ta; mG = Vb * rhog;
FB = Vb * rhoa * g; FG = mG * g; cdp = rhoa * Ab * CD / 2;
% compute times, velocities and elevations
tspan=[ti:dt:tf]; yinit=[vi zi];
[t,y]=ode45(@dydt2325,tspan,yinit,[],FB,FG,mG,cdp,mP,md,zd);
% Plot results
subplot(2,1,1)
plot(t,y(:,1))
title('Velocity vs. time')
xlabel('time (s)'),ylabel('velocity (m/s)')
subplot(2,1,2)
plot(t,y(:,2))
title('Elevation vs. time')
xlabel('time (s)'),ylabel('elevation (m)')

function dy=dydt2325(t,y,FB,FG,mG,cdp,mP,md,zd)
global g
if y(2) < zd, mP1 = mP; else, mP1 = mP - md; end
dy = [(FB - FG) / (mP1 + mG) - mP1 * g / (mP1 + mG) - ...
                   cdp / (mP1 + mG) * y(1) ^ 2; y(1)];
```



23.26

```
clear,clc,clf
format compact

%set parameters
kdT = 0.15; c0=20; o0 = 8.4; oc = 2;
y0=[c0 o0];

%perform calculations
opts=odeset('events',@endevent);
[tout,cout,te,ce]=ode45(@derivs,[0 inf],y0,opts,kdT);

%Output results
te,ce
z = [tout,cout]';
fprintf('       t       BOD        DO\n');
fprintf('%10.3f %10.3f %10.3f\n',z);
plot(tout,cout(:,1),'-',tout,cout(:,2),'--','LineWidth',2), grid
legend('BOD','DO'),xlabel('t (d)'),ylabel('mg/L')

function [detect,stopint,direction]=endevent(t,c,varargin)
% Locate the time when height passes through zero
% and stop integration.
detect=c(2)-2; % Detect height = 0
stopint=1; % Stop the integration
direction=0; % Direction does not matter

function dc=derivs(t,c,kdT)
dc=[-kdT*c(1);-kdT*c(1)];
```

```
te =
    2.5710
ce =
   13.6000    2.0000
       t        BOD        DO
     0.000     20.000      8.400
     0.141     19.582      7.982
     0.281     19.174      7.574
     0.422     18.773      7.173
     0.563     18.381      6.781
     1.065     17.048      5.448
     1.567     15.811      4.211
     2.069     14.664      3.064
     2.571     13.600      2.000
```



23.27

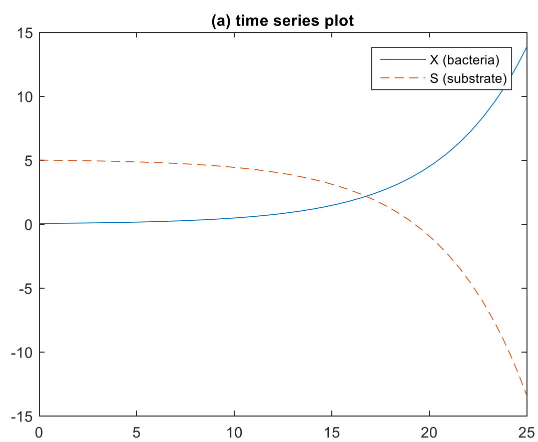
```
clear,clc,clf
Y=0.75;kmax=0.3;ks=1e-4;
tspan=[0 25];y0=[0.05 5];
disp('ode23');
tic;[t, y] = ode23(@bacteria,tspan,y0,[],Y,kmax,ks);toc;
plot(t,y(:,1),t,y(:,2),'--'),legend('X (bacteria)','S
(substrate)')
title('(a) time series plot')
pause
disp('ode23 with tol = 1e-6');
options=odeset('RelTol',1e-6);
tic;[t, y] = ode23(@bacteria,tspan,y0,options,Y,kmax,ks);toc;
plot(t,y(:,1),t,y(:,2),'--'),legend('X (bacteria)','S
(substrate)')
title('(a) time series plot')
pause
disp('ode45 with tol = 1e-6');
tic;[t, y] = ode45(@bacteria,tspan,y0,options,Y,kmax,ks);toc;
plot(t,y(:,1),t,y(:,2),'--'),legend('X (bacteria)','S
(substrate)')
title('(a) time series plot')
pause
disp('ode15s with tol = 1e-6');
tic;[t, y] = ode15s(@bacteria,tspan,y0,options,Y,kmax,ks);toc;
plot(t,y(:,1),t,y(:,2),'--'),legend('X (bacteria)','S
(substrate)')
title('(a) time series plot')
pause
disp('ode23s with tol = 1e-6');
tic;[t, y] = ode23s(@bacteria,tspan,y0,options,Y,kmax,ks);toc;
plot(t,y(:,1),t,y(:,2),'--'),legend('X (bacteria)','S
(substrate)')
title('(a) time series plot')
pause
disp('ode23t with tol = 1e-6');
tic;[t, y] = ode23t(@bacteria,tspan,y0,options,Y,kmax,ks);;toc;
```

```
plot(t,y(:,1),t,y(:,2),'--'),legend('X (bacteria)','S
(substrate)')
title('(a) time series plot')
pause
disp('ode23tb with tol = 1e-6');
tic;[t, y] = ode23tb(@bacteria,tspan,y0,options,Y,kmax,ks);toc;
plot(t,y(:,1),t,y(:,2),'--'),legend('X (bacteria)','S
(substrate)')
title('(a) time series plot')

function yp = bacteria(t,y,Y,kmax,ks)
yp = [Y*kmax*y(2)/(ks+y(2))*y(1);-kmax*y(2)/(ks+y(2))*y(1)];
end
```

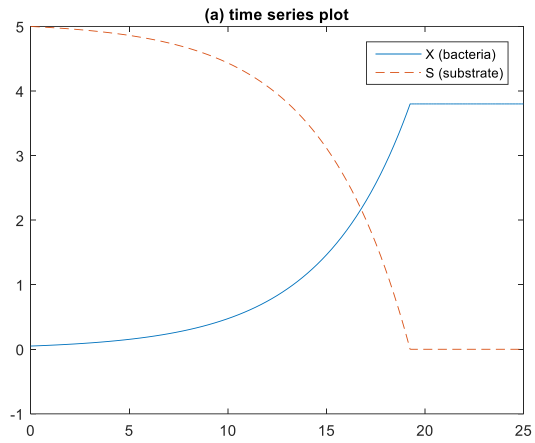
Results:

```
ode23
Elapsed time is 0.024489 seconds.
ode23 with tol = 1e-6
Elapsed time is 1.835925 seconds.
ode45 with tol = 1e-6
Elapsed time is 3.070684 seconds.
ode15s with tol = 1e-6
Elapsed time is 0.077614 seconds.
ode23s with tol = 1e-6
Elapsed time is 0.105561 seconds.
ode23t with tol = 1e-6
Elapsed time is 0.072509 seconds.
ode23tb with tol = 1e-6
Elapsed time is 0.069694 seconds.
```

The first option yields incorrect negative results, but does so quickly.



The remaining functions all yield positive results, but the non-stiff solvers (ode23 and ode45) take significantly longer to execute. Notice that the ode45 takes the longest. All the stiff solvers succeed quickly.



23.28 (a) Define the angular velocity as

$$w = \frac{d\theta}{dt}$$

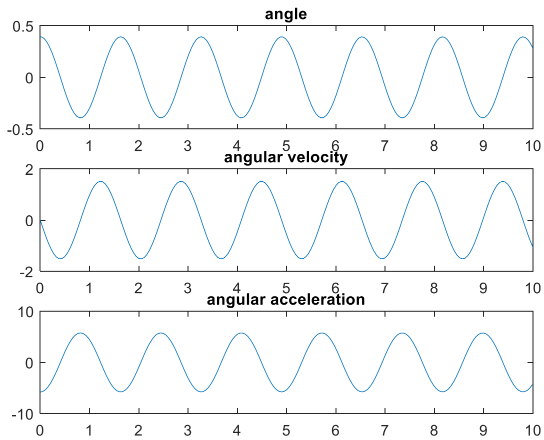
Therefore, the system of first-order differential equations to be solved are

$$\frac{d\theta}{dt} = w$$
$$\frac{dw}{dt} = -\frac{g}{l}\sin\theta$$

(b-d) Here is the script along with the function holding the differential equation:

```
clear,clc,clf
[t y] = ode45(@dpnon,[0 10],[pi/8 0]);
%generate angular accelerations
n=length(t);tm=(t(1:n-1)+t(2:n))./2;
accel=diff(y(:,2))./diff(t);
subplot(3,1,1)
plot(t,y(:,1))
title('angle')
subplot(3,1,2)
plot(t,y(:,2))
title('angular velocity')
subplot(3,1,3)
plot(tm,accel)
title('angular acceleration')

function dy = dpnon(t, y)
dy = [y(2);-9.81/0.65*sin(y(1))];
```

Here is the result:



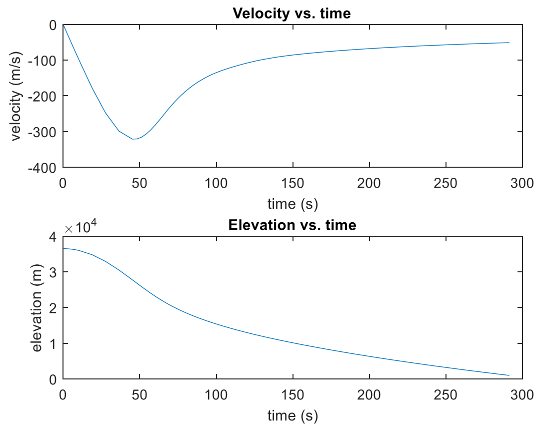
23.29 (a)

$$\frac{dv}{dt} = -g(z) - \frac{0.5\rho_{air}(z)AC_D}{m} v|v|$$
$$\frac{dz}{dt} = v$$

(b)

```
clear,clc,clf
format compact
zz=[-1 0 1 2 3 4 5 6 7 8 9 10 15 20 25 30 40 50 60 70 80]*1e3;
rho=[1.347 1.225 1.112 1.007 0.9093 0.8194 0.7364...
0.6601 0.5900 0.5258 0.4671 0.4135 0.1948 0.08891...
0.04008 0.01841 0.003996 0.001027 0.0003097 8.283e-5 1.846e-5];
A=0.55;Cd=1;m=80;
% numerically compute times, velocities and elevations
opts=odeset('events',@endevent2329);
[t,y]=ode45(@derivs,[0 inf],[0 36500],opts,zz,rho,A,Cd,m);
% Display results
fprintf('      t(s)      v(m/s)      z(m)\n');
xx=[t y];
fprintf('%10.3f %10.3f %10.3f\n',xx');
% Plot results
subplot(2,1,1)
plot(t,y(:,1))
title('Velocity vs. time')
xlabel('time (s)'),ylabel('velocity (m/s)')
subplot(2,1,2)
plot(t,y(:,2))
title('Elevation vs. time')
xlabel('time (s)'),ylabel('elevation (m)')

function [dy] = derivs(t,y,zz,rho,A,Cd,m)
% y(1) = velocity
```

```
% y(2) = elevation
rhoa = interp1(zz, rho, y(2),'pchip');
grav=9.806412-0.000003039734*y(2);
cd=0.5*rhoa*A*Cd;
dy = [-grav-cd/m*y(1)*abs(y(1));y(1)];

function [detect,stopint,direction]=endevent2329(t,y,varargin)
% Locate the time when elevation is 1 km and stop integration.
detect=y(2)-1000;    % Detect height = 0
stopint=1;        % Stop the integration
direction=0;      % Direction does not matter
```




```
     t(s)       v(m/s)      z(m)
    0.000        0.000  36500.000
    0.000       -0.000  36500.000
    0.000       -0.000  36500.000
    0.000       -0.000  36500.000
    0.000       -0.000  36500.000
    0.000       -0.000  36500.000
     ..          ..          ..
     ..          ..          ..
     ..          ..          ..
  280.119      -52.496   1579.438
  282.204      -52.180   1470.253
  284.289      -51.885   1361.709
  286.374      -51.628   1253.841
  288.459      -51.368   1146.532
  289.175      -51.266   1109.801
  289.890      -51.162   1073.130
  290.606      -51.060   1036.529
  291.321      -50.961   1000.000
```

23.30 (a) Force balances:

$$m\frac{dv_x}{dt} = -c_d v^2 \cos\theta$$

$$m\frac{dv_y}{dt} = mg - c_d v^2 \sin\theta$$

Because $\sin\theta = v_y/v$ and $\cos\theta = v_x/v$

$$\frac{dv_x}{dt} = -\frac{c_d}{m} v v_x \qquad \frac{dv_y}{dt} = g - \frac{c_d}{m} v v_y$$

The resultant of the velocity components is

$$v = \sqrt{v_x^2 + v_y^2}$$

Substituting this yields

$$\frac{dv_x}{dt} = -\frac{c_d}{m} v_x \sqrt{v_x^2 + v_y^2} \qquad \frac{dv_y}{dt} = g - \frac{c_d}{m} v_y \sqrt{v_x^2 + v_y^2}$$

The definition of velocity yields the other 2 differential equations

$$\frac{dx}{dt} = v_x \qquad \frac{dy}{dt} = v_y$$

(b) Substituting parameters:

$$\frac{dv_x}{dt} = -0.003125 v_x \sqrt{v_x^2 + v_y^2} \qquad \frac{dv_y}{dt} = 9.81 - 0.003125 v_y \sqrt{v_x^2 + v_y^2}$$

$$\frac{dx}{dt} = v_x \qquad \frac{dy}{dt} = v_y$$

Slopes at $t = 0$ (k_1s):

$$k_{1,1} = -0.003125(135)\sqrt{135 + 0} = -56.9531$$
$$k_{1,2} = 9.81 - 0.003125(0)\sqrt{135 + 0} = 9.81$$
$$k_{1,3} = 135$$
$$k_{1,4} = 0$$

Prediction at $t = 0.75h = 0.1875$:

$v_x(0.1875) = 135 + (-56.9531)0.1875 = 124.3213$
$v_y(0.1875) = 0 + (9.81)0.1875 = 1.839375$
$x(0.1875) = 0 + (135)0.1875 = 25.3125$
$y(0.1875) = 0 + (0)0.1875 = 0$

Slopes at t = 0.1875 (k_2s):

$$k_{2,1} = -0.003125(124.3213)\sqrt{124.3213^2 + 1.839375^2} = -48.3046$$
$$k_{2,2} = -0.003125(1.839375)\sqrt{124.3213^2 + 1.839375^2} = 9.095317$$
$$k_{2,3} = 124.3213$$
$$k_{2,4} = 1.839375$$

Average slopes:

$$\frac{dv_x}{dt} = \frac{1}{3}(-56.9531) + \frac{2}{3}(-48.3046) = -51.1874$$
$$\frac{dv_y}{dt} = \frac{1}{3}(9.81) + \frac{2}{3}(9.095317) = 9.333545$$
$$\frac{dx}{dt} = \frac{1}{3}(135) + \frac{2}{3}(124.3213) = 127.8809$$
$$\frac{dy}{dt} = \frac{1}{3}(0) + \frac{2}{3}(1.839375) = 1.22625$$

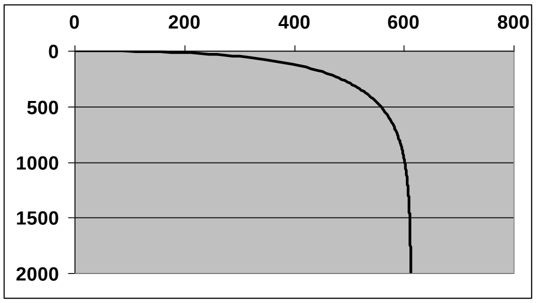
First step:

$v_x(0.25) = 135 + (-51.1874)0.25 = 122.2031$
$v_y(0.25) = 0 + (9.333545)0.25 = 2.333386$
$x(0.25) = 0 + (127.8809)0.25 = 31.97021$
$y(0.25) = 0 + (1.22625)0.25 = 0.306563$

Here are a few more steps to confirm that you are generating the correct results:

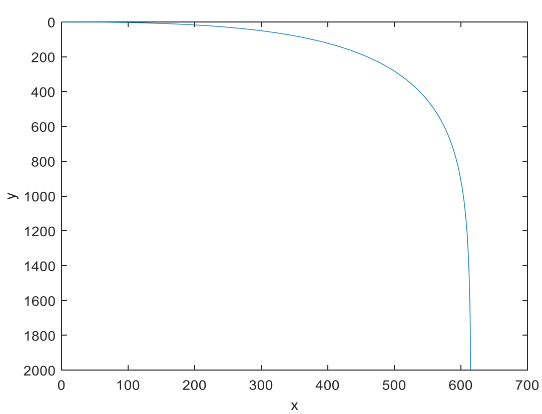
t	v_x	v_y	x	y
0	135.00	0.00	0.00	0.00
0.25	122.20	2.33	31.97	0.31
0.5	111.61	4.47	61.06	1.17
0.75	102.68	6.47	87.75	2.55
1	95.07	8.35	112.39	4.40

(c) A plot of *y* versus *x* can be generated as



(d)

```
clear,clc,clf,format compact
global g
g=9.81;cd=0.25;m=80;
y0=[135 0 0 0];tspan=[0 inf];
opts=odeset('events',@endevent2330);
[t y] = ode45(@dydt2330,tspan,y0,opts,m,cd);
plot(y(:,3),y(:,4))
set(gca,'YDir','reverse');
xlabel('x'),ylabel('y')

function dy = dydt2330(t,y,m,cd)
global g
% y(1)=vx
% y(2)=vy
% y(3)=x
% y(4)=y
dy =[-cd/m*y(1)*sqrt(y(1)^2+y(2)^2);...
     g-cd/m*y(2)*sqrt(y(1)^2+y(2)^2);...
     y(1);y(2)];

function [detect,stopint,direction]=endevent2330(t,y,varargin)
% Locate the time when elevation is 2000 and stop integration.
detect=y(4)-2000;   % Detect height = 0
stopint=1;     % Stop the integration
direction=0;   % Direction does not matter
```

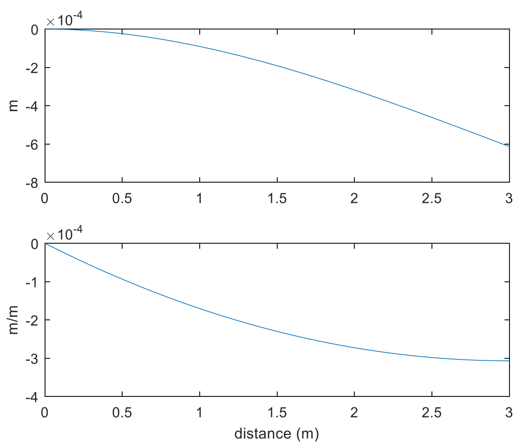


23.31 This is an initial-value problem because the values of the variables are given at the start of the integration interval; i.e., at $x = 0$, $y = z = 0$. In order to obtain a numerical solution, the second-order differential equation can be expressed as a pair of first-order ODEs,

$$\frac{dy}{dx} = z$$
$$\frac{dz}{dx} = -\frac{P}{EI}(L - x)$$

These equations can be simply integrated with any of the techniques from this part of the book. The following shows the results of using ode45 to obtain the solution.

```
clear,clc
E = 2e11; I = 0.00033; P = 4.5e3;L = 3;
xspan = [0:L/100:L]; y0 = [0 0];
[x,y]=ode45(@cantilever,xspan,y0,[],E,I,P,L);
subplot(2,1,1)
plot(x,y(:,1))
ylabel('m')
subplot(2,1,2)
plot(x,y(:,2))
xlabel('distance (m)')
ylabel('m/m')

function dy=cantilever(x,y,E,I,P,L)
dy=[y(2);-P/(E*I)*(L-x)];
```



23.32

Scripts and Functions:

```
clc, clear, close all
k = [0.15,0.15,0.15,0.15]';
% adjust parameters to minimize SSE
```

```
[ k SSEmin ] = fminsearch(@SSE,k)

function sum=SSE(k)
t=[1,2,3,4,5,6,8,9,10,12,15]';
c1exp=[85.3,66.6,60.6,56.1,49.1,45.3,41.9,37.8,33.7,34.4,35.1]';
c2exp=[16.9,18.7,24.1,20.9,18.9,19.9,20.6,13.9,19.1,14.5,15.4]';
c3exp=[4.7,7.9,20.1,22.8,32.5,37.7,42.4,47,50.5,52.3,51.3]';
c10=100;
c20=0;
c30=0;
tspan=[0;t];
y0=[c10,c20,c30]';
reactanon = @ (t,y) react(t,y,k);
[T Y]=ode45(reactanon,tspan,y0);
e1 = c1exp - Y(2:end,1);
e2 = c2exp - Y(2:end,2);
e3 = c3exp - Y(2:end,3);
sum=e1'*e1 + e2'*e2 + e3'*e3;

function dc=react(t,y,k)
c1=y(1);
c2=y(2);
c3=y(3);
dc(1)=-k(1)*c1+k(2)*c2+k(4)*c3;
dc(2)=k(1)*c1-k(2)*c2-k(3)*c2;
dc(3)=k(3)*c2-k(4)*c3;
dc=dc';
```

Results:

```
k =
    0.2030
    0.1103
    0.4160
    0.0973
SSEmin =
  131.6424
```