CHAPTER 28

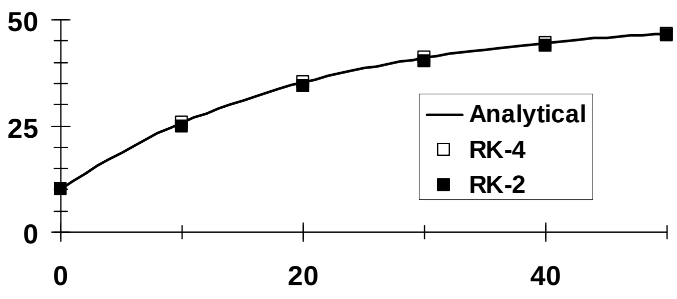
28.1 The solution with the 2nd-order RK (Heun without corrector) can be laid out as

t	c	k_1	c	k_2	ϕ
0	10	2	30	1	1.5
10	25	1.25	37.5	0.625	0.9375
20	34.375	0.78125	42.1875	0.390625	0.585938
30	40.23438	0.488281	45.11719	0.244141	0.366211
40	43.89648	0.305176	46.94824	0.152588	0.228882
50	46.1853	0.190735	48.09265	0.095367	0.143051

For the 4th-order RK, the solution is

t	c	k_1	c_{mid}	k_2	c_{mid}	k_3	c_{end}	k_4	ϕ
0	10	2	20	1.5	17.5	1.625	26.25	1.1875	1.572917
10	25.72917	1.213542	31.79688	0.910156	30.27995	0.986003	35.58919	0.72054	0.9544
20	35.27317	0.736342	38.95487	0.552256	38.03445	0.598278	41.25594	0.437203	0.579102
30	41.06419	0.446791	43.29814	0.335093	42.73965	0.363017	44.69436	0.265282	0.351382
40	44.57801	0.2711	45.93351	0.203325	45.59463	0.220268	46.78069	0.160965	0.213208
50	46.71009	0.164495	47.53257	0.123371	47.32695	0.133652	48.04662	0.097669	0.129369

A plot of both solutions along with the analytical result is displayed below:

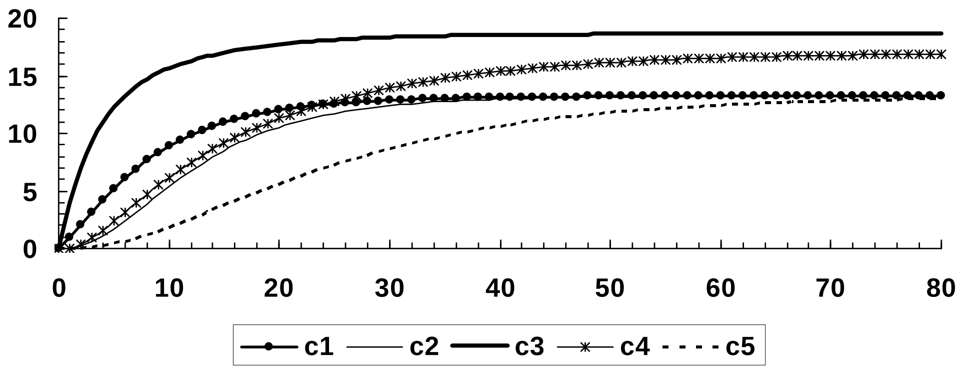


28.2 The mass-balance equations can be written as

$$\frac{dc_1}{dt} = -0.16c_1 + 0.06c_3 + 1$$
$$\frac{dc_2}{dt} = 0.2c_1 - 0.2c_2$$
$$\frac{dc_3}{dt} = 0.05c_2 - 0.25c_3 + 4$$
$$\frac{dc_4}{dt} = 0.0875c_3 - 0.125c_4 + 0.0375c_5$$
$$\frac{dc_5}{dt} = 0.04c_1 + 0.02c_2 - 0.06c_5$$

Selected solution results (Euler’s method) are displayed below, along with a plot of the results.

t	c_1	c_2	c_3	c_4	c_5	dc_1/dt	dc_2/dt	dc_3/dt	dc_4/dt	dc_5/dt
0	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	4.0000	0.0000	0.0000
1	1.0000	0.0000	4.0000	0.0000	0.0000	1.0800	0.2000	3.0000	0.3500	0.0400
2	2.0800	0.2000	7.0000	0.3500	0.0400	1.0872	0.3760	2.2600	0.5703	0.0848
3	3.1672	0.5760	9.2600	0.9203	0.1248	1.0488	0.5182	1.7138	0.6999	0.1307
4	4.2160	1.0942	10.9738	1.6201	0.2555	0.9839	0.6244	1.3113	0.7673	0.1752
5	5.1999	1.7186	12.2851	2.3874	0.4307	0.9051	0.6963	1.0147	0.7927	0.2165
•										
•										
•										
76	13.2416	13.2395	18.6473	16.8515	12.9430	0.0002	0.0004	0.0001	0.0106	0.0179
77	13.2418	13.2399	18.6475	16.8621	12.9609	0.0002	0.0004	0.0001	0.0099	0.0168
78	13.2419	13.2403	18.6476	16.8720	12.9777	0.0001	0.0003	0.0001	0.0093	0.0158
79	13.2421	13.2406	18.6477	16.8813	12.9935	0.0001	0.0003	0.0001	0.0088	0.0149
80	13.2422	13.2409	18.6478	16.8901	13.0084	0.0001	0.0003	0.0001	0.0082	0.0140



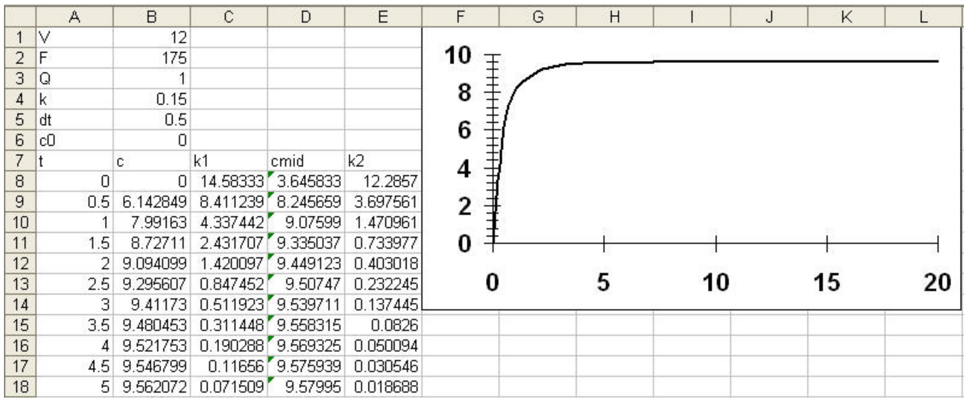
Finally, MATLAB can be used to determine the eigenvalues and eigenvectors:

```
>> a=[.16 -.06 0 0 0;-.2 .2 0 0 0;0 -.05 .25 0 0;0 0 -.0875 .125 -.0375;-.04 -
.02 0 0 .06]
a =
    0.1600   -0.0600         0         0         0
   -0.2000    0.2000         0         0         0
         0   -0.0500    0.2500         0         0
         0         0   -0.0875    0.1250   -0.0375
   -0.0400   -0.0200         0         0    0.0600

>> [v,d]=eig(a)
v =
         0         0         0   -0.1039    0.2604
         0         0         0   -0.1582   -0.5701
         0    0.8192         0   -0.0436    0.6892
    1.0000   -0.5735    0.4997    0.4956   -0.3635
         0         0    0.8662    0.8466    0.0043
d =
    0.1250         0         0         0         0
         0    0.2500         0         0         0
         0         0    0.0600         0         0
         0         0         0    0.0686         0
         0         0         0         0    0.2914
```

28.3 Substituting the parameters into the differential equation gives

$$\frac{dc}{dt} = 14.583333 - 0.0833333c - 0.15c^2$$

The mid-point method can be applied with the result:



	A	B	C	D	E
1	V	12			
2	F	175			
3	Q	1			
4	k	0.15			
5	dt	0.5			
6	c0	0			
7	t	c	k1	cmid	k2
8	0	0	14.58333	3.645833	12.2857
9	0.5	6.142849	8.411239	8.245659	3.697561
10	1	7.99163	4.337442	9.07599	1.470961
11	1.5	8.72711	2.431707	9.335037	0.733977
12	2	9.094099	1.420097	9.449123	0.403018
13	2.5	9.295607	0.847452	9.50747	0.232245
14	3	9.41173	0.511923	9.539711	0.137445
15	3.5	9.480453	0.311448	9.558315	0.0826
16	4	9.521753	0.190288	9.569325	0.050094
17	4.5	9.546799	0.11656	9.575939	0.030546
18	5	9.562072	0.071509	9.57995	0.018688

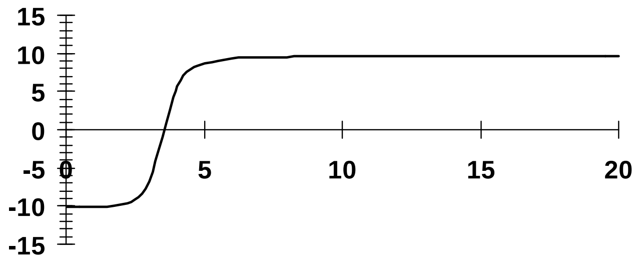
The results are approaching a steady-state value of 9.586267.

<u>Challenge question:</u>

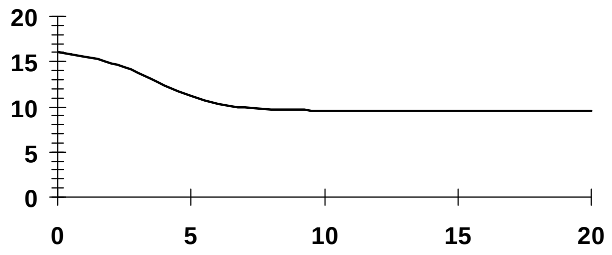
The steady state form (i.e., $dc/dt = 0$) of the equation is $0 = 14.58333 - 0.08333c - 0.15c^2$, which can be solved for 9.586267 and –10.1418. Thus, there is a negative root.

If we put in the initial *y* value as –10.1418 (or higher precision), the solution will stay at the negative root for awhile until roundoff errors may lead to the solution going unstable. If we use an initial value less than the negative root, the solution will also go unstable.

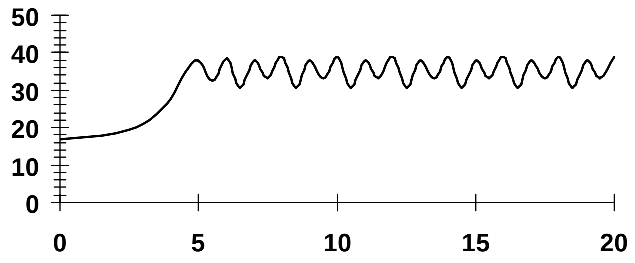
However, if we pick a value that is slightly higher (as per machine precision), it will gravitate towards the positive root. For example if we use –10.14



This kind of behavior will also occur for higher initial conditions. For example, using an initial condition of 16 gives,



However, if we start to use even higher initial conditions, the solution will again go unstable. For example, if we use an initial condition of 17, the result is

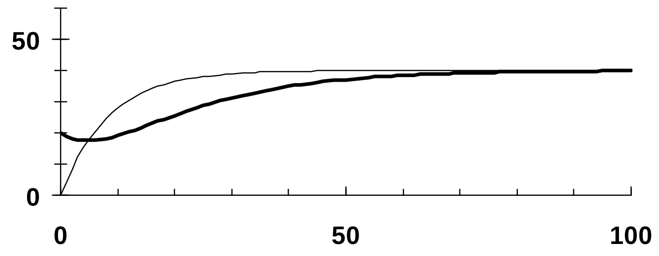


Therefore, it looks like initial conditions roughly in the range from –10.14 up to about 16.5 will yield stable solutions that converge on the steady-state solution of 9.586. Note that this range depends on our choice of step size.

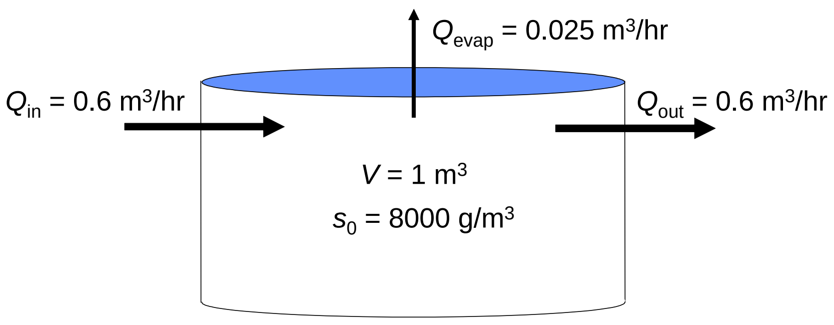
28.4 The first steps of the solution are shown below along with a plot. Notice that a value of the inflow concentration at the end of the interval (***c_{in}-end***) is required to calculate the k_2's correctly.

t	c_{in}	c	k_1	c_{end}	c_{in}-end	k_2	ϕ
0	0	20	-1.2	17.6	8.534886	-0.54391	-0.87195
2	8.534886	18.25609	-0.58327	17.08955	15.24866	-0.11045	-0.34686
4	15.24866	17.56237	-0.13882	17.28472	20.52991	0.194711	0.027944
6	20.52991	17.61826	0.174699	17.96766	24.68428	0.402998	0.288848
8	24.68428	18.19595	0.3893	18.97455	27.95223	0.538661	0.46398
10	27.95223	19.12391	0.529699	20.18331	30.52289	0.620375	0.575037

In the following plot, the inflow concentration (thin line) and the outflow concentration (heavy line) are both shown.



28.5 The system is as depicted below:



(a) The mass of water in the tank can be modeled with a simple mass balance

$$\frac{dV}{dt} = Q_{\text{in}} - Q_{\text{out}} - Q_{\text{evap}} = 0.6 - 0.6 - 0.025 = -0.025$$

With the initial condition that $V = 1$ m^3 at $t = 0$, this equation can be integrated to yield,

$$V = 1 - 0.025t$$

Thus, the time to empty the tank ($M = 0$) can be calculated as $t = 1/0.025 = 40$ hrs.

(b) The concentration in the tank over this period can be computed in several ways. The simplest is to compute the mass of salt in the tank over time by solving the following differential equation:

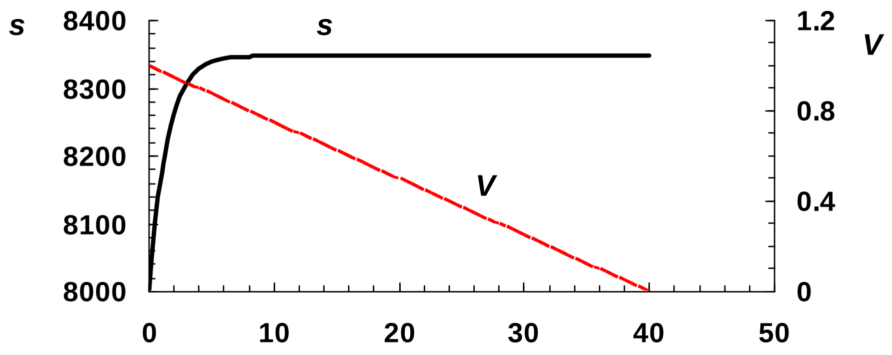
$$\frac{dm}{dt} = Q_{\text{in}} s_{\text{in}} - Q_{\text{out}} s \qquad (1)$$

where m = the mass of salt in the tank. The salt concentration in the tank, s, is the ratio of the mass of salt to the volume of water

$$s = \frac{m}{V} = \frac{m}{1 - 0.025t} \qquad (2)$$

Thus, we can integrate Eq. 1 to determine the mass at each time step and then use Eq. 2 to determine the corresponding salt concentration. The first few steps of the solution of the ODE using Euler's method are tabulated below. In addition, a graph of the entire solution is also displayed. Note that the concentration asymptotically approaches a constant value of 8,347.826 g/m^3 as the solution evolves.

t	***Vol***	***m***	***s***	***dm/dt***
0	1	8000	8000	0
0.5	0.9875	8000	8101.266	-60.7595
1	0.975	7969.62	8173.969	-104.382
1.5	0.9625	7917.429	8225.901	-135.54
2	0.95	7849.659	8262.799	-157.679
2.5	0.9375	7770.819	8288.874	-173.324



Recognize that a singularity occurs at t = 40, because the tank would be totally empty at this point.

28.6 A heat balance for the sphere can be written as

$$\frac{dH}{dt} = hA(T_a - T)$$

The heat gain can be transformed into a volume loss by considering the latent heat of fusion.

$$\frac{dV}{dt} = -\frac{hA}{\rho L_f}(T_a - T) \qquad (1)$$

where ρ = density ≅ 0.917 kg/m^3 and L_f = latent heat of fusion ≅ 333 kJ/kg. The volume and area of a sphere are computed by

$$V = \frac{4}{3}\pi r^3 \qquad A = 4\pi r^2 \qquad (2)$$

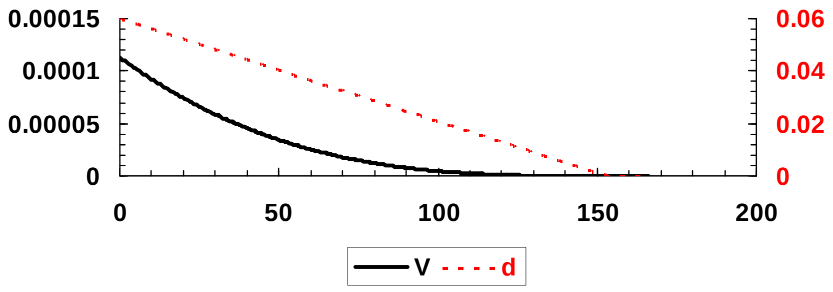
These can be combined with (1) to yield,

$$\frac{dV}{dt} = \frac{h4\pi\left(\frac{3}{4}\frac{V}{\pi}\right)^{2/3}}{\rho L_f}(T - T_a)$$

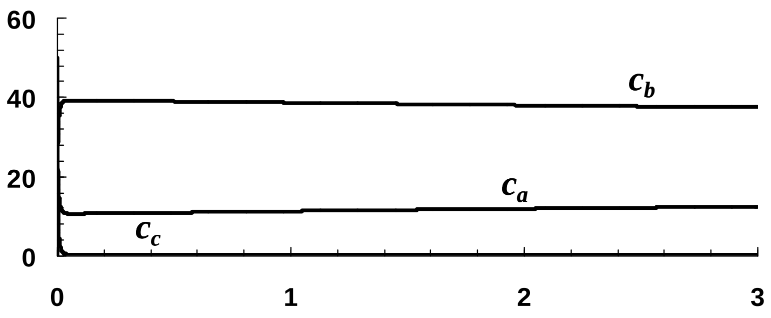
where $T = 0^{\circ}$C. This equation can be integrated along with the initial condition,

$$V_0 = \frac{4}{3}\pi(0.03)^3 = 0.000113 \text{ m}^3$$

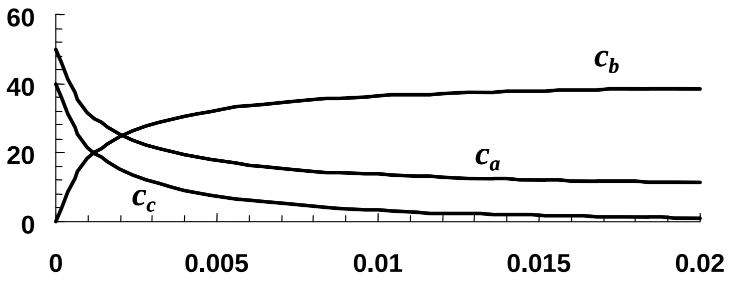
to yield the resulting volume as a function of time. This result can be converted into diameter using (2). Both results are shown on the following plot which indicates that it takes about 150 s for the ice to melt.



28.7 The system for this problem is stiff. Thus, the use of a simple explicit Runge-Kutta scheme would involve using a very small time step in order to maintain a stable solution. A solver designed for stiff systems was used to generate the solution shown below. Two views of the solution are given. The first is for the entire solution domain.



In addition, we can enlarge the initial part of the solution to illustrate the fast transients that occur as the solution moves from its initial conditions to its dominant trajectories.



28.8 Several methods could be used to obtain a solution for this problem (e.g., finite-difference, shooting method, finite-element). The finite-difference approach is straightforward:

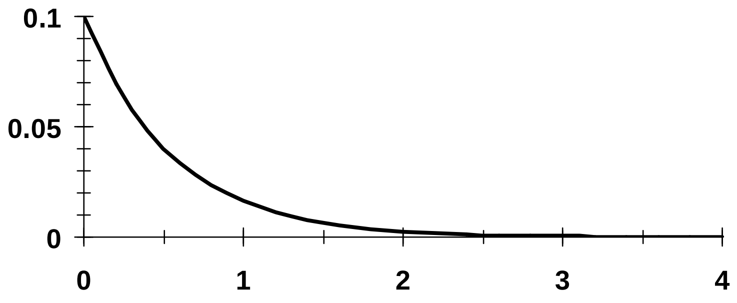
$$D\frac{A_{i-1} - 2A_i + A_{i+1}}{\Delta x^2} - kA_i = 0$$

Substituting parameter values and collecting terms gives

$$-1.5\times10^{-6} A_{i-1} + (3\times10^{-6} + 5\times10^{-6}\Delta x^2)A_i - 1.5\times10^{-6} A_{i+1} = 0$$

Using a Δx = 0.2 cm this equation can be written for all the interior nodes. The resulting linear system can be solved with an approach like the Gauss-Seidel method. The following table and graph summarize the results.

x	*A*	*x*	*A*	*x*	*A*	*x*	*A*
0	0.1						
0.2	0.069544	1.2	0.011267	2.2	0.001779	3.2	0.000257
0.4	0.048359	1.4	0.007814	2.4	0.001224	3.4	0.000166
0.6	0.033621	1.6	0.005415	2.6	0.000840	3.6	9.93E-05
0.8	0.023368	1.8	0.003748	2.8	0.000574	3.8	4.65E-05
1	0.016235	2	0.002591	3	0.000389	4	0



28.9 (a) The temperature of the body can be determined by integrating Newton's law of cooling to give,

$$T(t) = T_o e^{-Kt} + T_a(1 - e^{-Kt})$$

This equation can be solved for *K*,

$$K = -\frac{1}{t}\ln\frac{T(t) - T_a}{T_o - T_a}$$

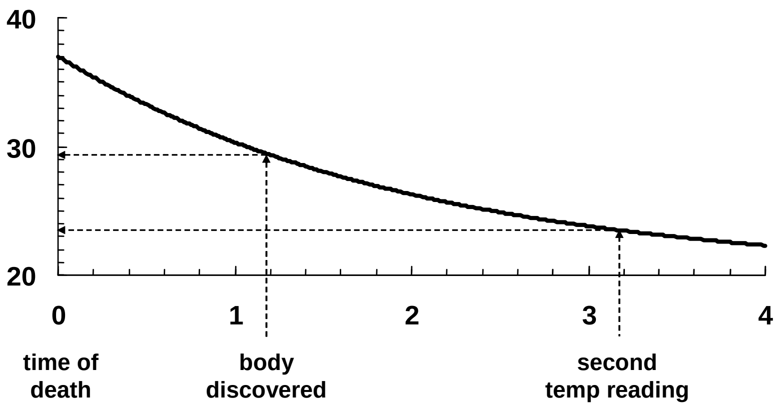
Substituting the values yields

$$K = -\frac{1}{2}\ln\frac{23.5 - 20}{29.5 - 20} = 0.499264\,/\,\text{hr}$$

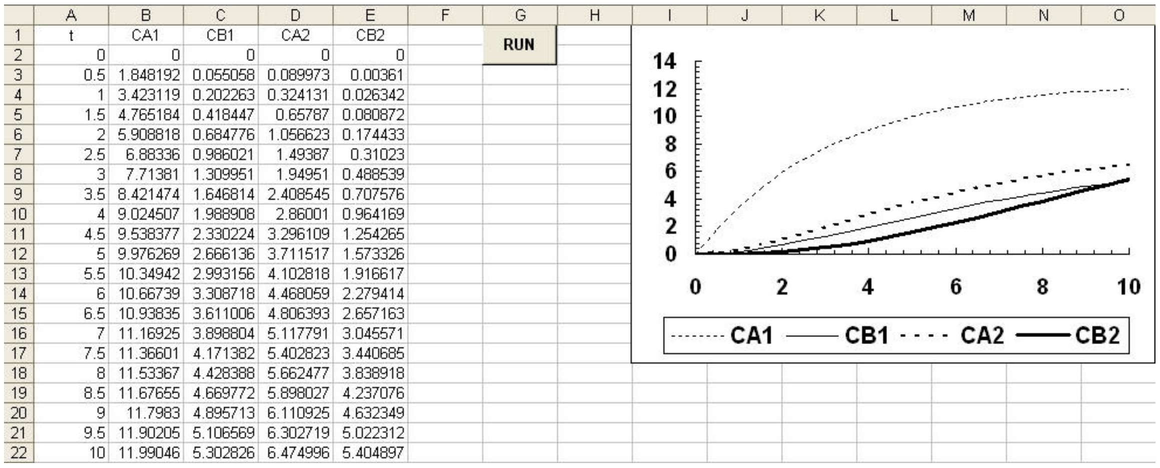
The time of death can then be computed as

$$t_d = -\frac{1}{K}\ln\frac{T(t_d) - T_a}{T_o - T_a} = -\frac{1}{0.499264}\ln\frac{37 - 20}{29.5 - 20} = -1.16556\text{ hr}$$

Thus, the person died 1.166 hrs prior to being discovered. The non-self-starting Heun yielded the following time series of temperature. For convenience, we have redefined the time of death as t = 0.:



28.10 The classical 4th order RK method yields

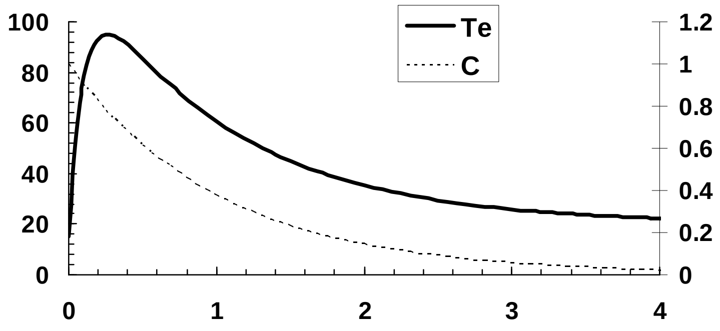


	A	B	C	D	E
1	t	CA1	CB1	CA2	CB2
2	0	0	0	0	0
3	0.5	1.848192	0.055058	0.089973	0.00361
4	1	3.423119	0.202263	0.324131	0.026342
5	1.5	4.765184	0.418447	0.65787	0.080872
6	2	5.908818	0.684776	1.056623	0.174433
7	2.5	6.88336	0.986021	1.49387	0.31023
8	3	7.71381	1.309951	1.94951	0.488539
9	3.5	8.421474	1.646814	2.408545	0.707576
10	4	9.024507	1.988908	2.86001	0.964169
11	4.5	9.538377	2.330224	3.296109	1.254265
12	5	9.976269	2.666136	3.711517	1.573326
13	5.5	10.34942	2.993156	4.102818	1.916617
14	6	10.66739	3.308718	4.468059	2.279414
15	6.5	10.93835	3.611006	4.806393	2.657163
16	7	11.16925	3.898804	5.117791	3.045571
17	7.5	11.36601	4.171382	5.402823	3.440685
18	8	11.53367	4.428388	5.662477	3.838918
19	8.5	11.67655	4.669772	5.898027	4.237076
20	9	11.7983	4.895713	6.110925	4.632349
21	9.5	11.90205	5.106569	6.302719	5.022312
22	10	11.99046	5.302826	6.474996	5.404897

28.11 The classical 4th order RK method yields

t	*C*	*Te*
0	1	15
0.0625	0.941261	60.79675
0.125	0.885808	82.90298
0.1875	0.83356	92.37647
0.25	0.784373	95.1878
0.3125	0.738086	94.54859
0.375	0.694535	92.18087
0.4375	0.653562	89.00406
0.5	0.615016	85.50575
0.5625	0.578753	81.94143
0.625	0.544638	78.4421
0.6875	0.512542	75.07218
0.75	0.482346	71.86061
0.8125	0.453936	68.8176
0.875	0.427205	65.94362
0.9375	0.402055	63.23421

```
        1     0.378391       60.68253
        •
        •
        •
```



28.12 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

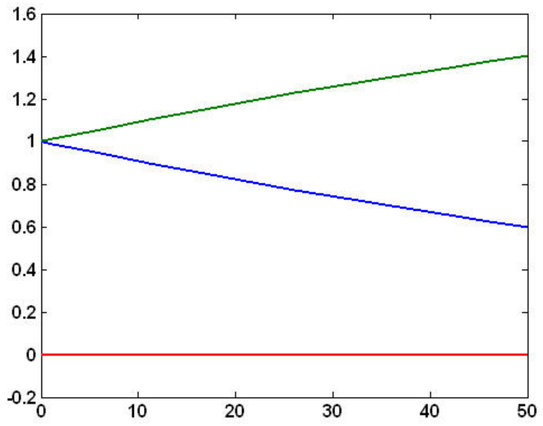
```
>> tspan=[0,50];
>> y0=[1,1,0];
>> [t,y]=ode45(@dcdtstiff,tspan,y0);
```

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)
```



28.13 Centered differences can be substituted for the derivatives to give

$$D\frac{c_{a,i+1}-2c_{a,i}+c_{a,i-1}}{\Delta x^2}=0 \qquad 0\le x<L$$
$$D_f\frac{c_{a,i+1}-2c_{a,i}+c_{a,i-1}}{\Delta x^2}-kc_{a,i}=0 \qquad L\le x<L+L_f$$

Collecting terms

$$-\frac{D}{\Delta x^2}c_{a,i-1}+\frac{2D}{\Delta x^2}c_{a,i}-\frac{D}{\Delta x^2}c_{a,i+1}=0 \qquad 0\le x<L$$
$$-\frac{D_f}{\Delta x^2}c_{a,i-1}+\left(\frac{2D_f}{\Delta x^2}+k\right)c_{a,i}-\frac{D_f}{\Delta x^2}c_{a,i+1}=0 \qquad L\le x<L+L_f$$

The boundary conditions can be developed. For the first node (i = 1),

$$\frac{2D}{\Delta x^2}c_{a,1}-\frac{D}{\Delta x^2}c_{a,2}=\frac{D}{\Delta x^2}c_{a0}$$

For the last,

$$-\frac{2D_f}{\Delta x^2}c_{a,n-1}+\left(\frac{2D_f}{\Delta x^2}+k\right)c_{a,n}$$

A special equation is also required at the interface between the diffusion layer and the biofilm ($x = L$). A flux balance can be written around this node as

$$D\frac{c_{a,i-1}-c_{a,i}}{\Delta x}+D_f\frac{c_{a,i+1}-c_{a,i}}{\Delta x}-k\frac{\Delta x}{2}c_{a,i}=0$$

Collecting terms gives

$$-\frac{D}{\Delta x}c_{a,i-1}+\left(\frac{D+D_F}{\Delta x}+k\frac{\Delta x}{2}\right)c_{a,i}-\frac{D_f}{\Delta x}c_{a,i+1}=0$$

Substituting the parameters gives

first node:

$$160,000c_{a,1}-80,000c_{a,2}=8,000,000$$

interior nodes (diffusion layer):

$$-80,000c_{a,i-1}+160,000c_{a,i}-80,000c_{a,i+1}=0$$

boundary node (i = 6):

$$-80,000c_{a,5}+121,000c_{a,6}-40,000c_{a,7}=0$$

interior nodes (biofilm):

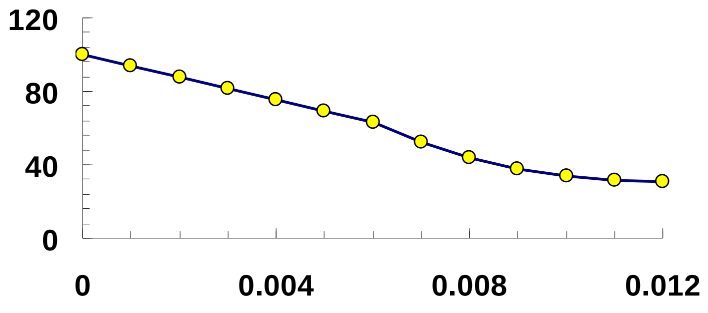
$$-40,000c_{a,i-1} + 82,000c_{a,i} - 40,000c_{a,i+1} = 0$$

last node:

$$-80,000c_{a,n-1} + 82,000c_{a,n} = 0$$

The solution is

x	*c_a*
0	100.0000
0.001	93.8274
0.002	87.6549
0.003	81.4823
0.004	75.3097
0.005	69.1372
0.006	62.9646
0.007	52.1936
0.008	44.0322
0.009	38.0725
0.01	34.0164
0.011	31.6611
0.012	30.8889



28.14 Substitute centered difference for the derivatives,

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

Collecting terms gives

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

Substituting the parameter values yields

$$-55c_{i-1} + 102c_i - 45c_{i+1} = 0$$

For the inlet node (i = 1), we must use a finite difference approximation for the first derivative. We use a second-order version (recall Table 19.3) so that the accuracy is comparable to the centered differences we employ for the interior nodes,

$$Uc_{\text{in}} = Uc_1 - D\frac{-c_3 + 4c_2 - 3c_1}{2\Delta x}$$

which can be solved for

$$\left(\frac{3D}{2\Delta x^2} + \frac{U}{\Delta x}\right)c_1 - \left(\frac{2D}{\Delta x^2}\right)c_2 + \left(\frac{D}{2\Delta x^2}\right)c_3 = \frac{U}{\Delta x}c_{\text{in}}$$

Substituting the parameters gives

$$85c_1 - 100c_2 + 25c_3 = 1000$$

For the outlet node (i = 10), the zero derivative condition implies that $c_{11} = c_9$,

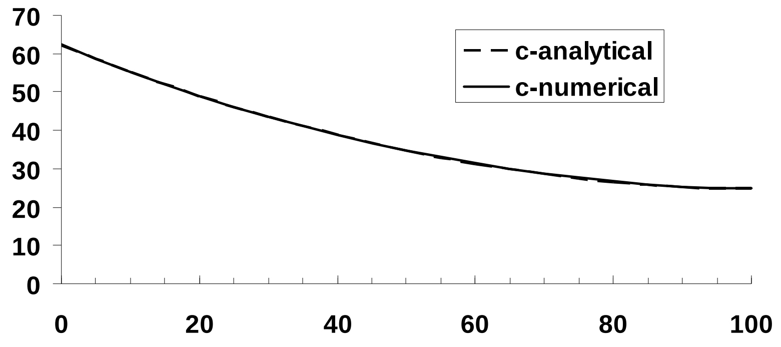
$$-\left(\frac{2D}{\Delta x^2}\right)c_9 + \left(\frac{2D}{\Delta x^2} + k\right)c_{10} = 0$$

Substituting the parameters gives

$$-100c_9 + 102c_{10} = 0$$

The tridiagonal system can be solved. Here are the results together with the analytical solution. As can be seen, the results are quite close.

x	***c*-numerical**	***c*-analytical**
0	63.6967	62.1767
10	56.4361	55.0818
20	50.0702	48.8634
30	44.5150	43.4390
40	39.7038	38.7437
50	35.5881	34.7300
60	32.1394	31.3708
70	29.3528	28.6615
80	27.2516	26.6258
90	25.8945	25.3223
100	25.3868	24.8552



28.15 Centered differences can be substituted for the derivatives to give

$$D\frac{c_{a,i+1}-2c_{a,i}+c_{a,i-1}}{\Delta x^2}-U\frac{c_{a,i+1}-c_{a,i-1}}{2\Delta x}-k_1c_{a,i}=0$$
$$D\frac{c_{b,i+1}-2c_{b,i}+c_{b,i-1}}{\Delta x^2}-U\frac{c_{b,i+1}-c_{b,i-1}}{2\Delta x}+k_1c_{a,i}-k_2c_{b,i}=0$$
$$D\frac{c_{c,i+1}-2c_{c,i}+c_{c,i-1}}{\Delta x^2}-U\frac{c_{c,i+1}-c_{c,i-1}}{2\Delta x}+k_2c_{b,i}=0$$

Collecting terms gives

$$-50c_{a,i-1}+83c_{a,i}-30c_{a,i+1}=0$$
$$-50c_{b,i-1}+81c_{b,i}-30c_{b,i+1}=3c_{a,i}$$
$$-50c_{c,i-1}+80c_{c,i}-30c_{c,i+1}=c_{b,i}$$

For the inlet node (i = 1), we must use a finite difference approximation for the first derivative. We use a second-order version (recall Table 19.3) so that the accuracy is comparable to the centered differences we employ for the interior nodes. For example, for reactant A,

$$Uc_{a,\text{in}}=Uc_{a,1}-D\frac{-c_{a,3}+4c_{a,2}-3c_{a,1}}{2\Delta x}$$

which can be solved for

$$\left(\frac{3D}{2\Delta x^2}+\frac{U}{\Delta x}\right)c_{a,1}-\left(\frac{2D}{\Delta x^2}\right)c_{a,2}+\left(\frac{D}{2\Delta x^2}\right)c_{a,3}=\frac{U}{\Delta x}c_{a,\text{in}}$$

Because the condition does not include reaction rates, similar equations can be written for the other nodes. Substituting the parameters gives

$$80c_{a,1}-80c_{a,2}+20c_{a,3}=200$$
$$80c_{b,1}-80c_{b,2}+20c_{b,3}=0$$
$$80c_{c,1}-80c_{c,2}+20c_{c,3}=0$$

For the outlet node (i = 10), the zero derivative condition implies that $c_{11}=c_9$,

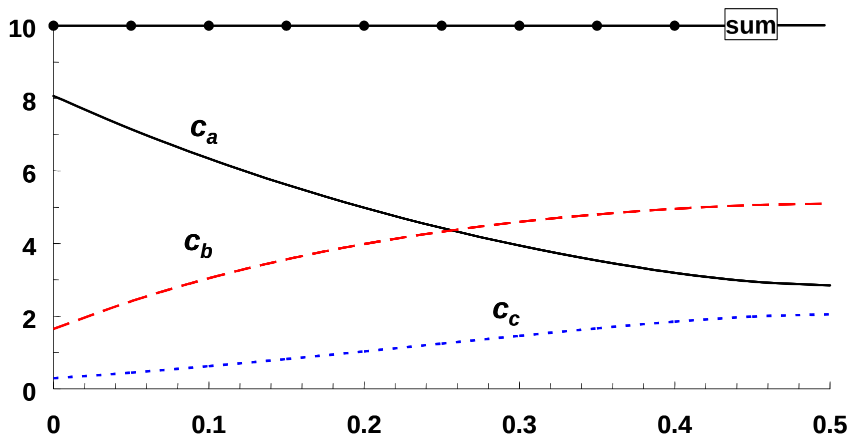
$$-\left(\frac{2D}{\Delta x^2}\right)c_9+\left(\frac{2D}{\Delta x^2}+k\right)c_{10}=0$$

Again, because the condition does not include reaction rates, similar equations can be written for the other nodes. Substituting the parameters gives

$$-80c_{a,9}+83c_{a,10}=0$$
$$-80c_{b,9}+81c_{b,10}=3c_{a,10}$$
$$-80c_{c,9}+80c_{c,10}=c_{b,10}$$

Notice that because the reactions are in series, we can solve the systems for each reactant separately in sequence. The result is

x	c_a	c_b	c_c	sum
0	8.0646	1.6479	0.2875	10
0.05	7.1492	2.4078	0.4430	10
0.1	6.3385	3.0397	0.6218	10
0.15	5.6212	3.5603	0.8184	10
0.2	4.9878	3.9846	1.0276	10
0.25	4.4309	4.3258	1.2433	10
0.3	3.9459	4.5955	1.4587	10
0.35	3.5320	4.8035	1.6644	10
0.4	3.1955	4.9572	1.8473	10
0.45	2.9542	5.0591	1.9867	10
0.5	2.8474	5.1021	2.0505	10

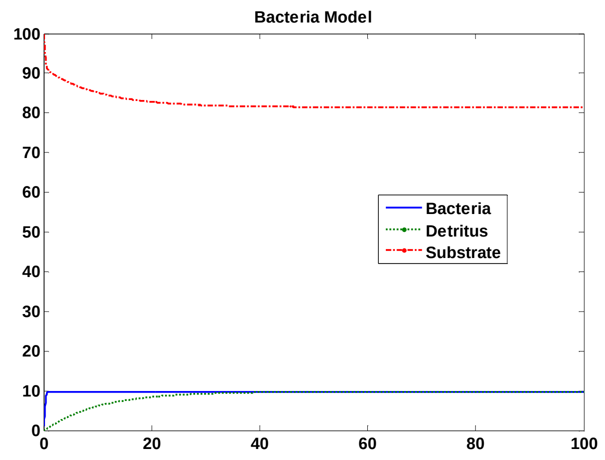


28.16 The differential equations can be implemented with a MATLAB function:

```
function yp=bacteriaNM6(t,y,umax,K,Ks,kd,ke,kh)
growth=umax*(1-y(1)/K)*y(3)/(Ks+y(3))*y(1);
yp=[growth-kd*y(1)-kd*y(1);kd*y(1)-kh*y(2);ke*y(1)+kh*y(2)-growth];
```

A script can then be used to obtain the solution:

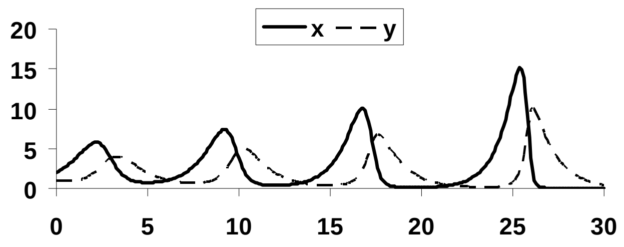
```
tspan=[0 100];y0=[1 0 100];
umax=10;K=10;Ks=10;kd=0.1;ke=0.1;kh=0.1;
[t y] = ode45(@bacteriaNM6,tspan,y0,[],umax,K,Ks,kd,ke,kh);
plot(t,y(:,1),t,y(:,2),t,y(:,3))
title('Bacteria Model');legend('Bacteria','Detritus','Substrate')
```



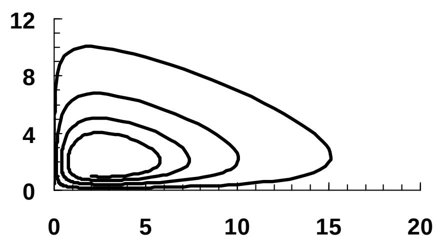
28.17 (a) The first few steps of Euler’s method are shown in the following table

t	*x*	*y*
0	2	1
0.1	2.12	0.98
0.2	2.249744	0.963928
0.3	2.389598	0.951871
0.4	2.539874	0.943959
0.5	2.700807	0.940369

A plot of the entire simulation is shown below:



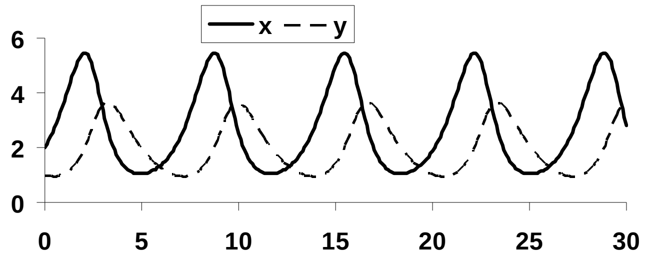
Notice that because the Euler method is lower order, the peaks are increasing, rather than repeating in a stable manner as time progresses. This result is reinforced when a state-space plot of the calculation is displayed.



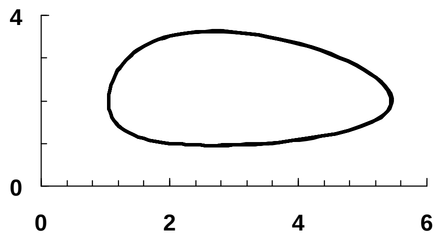
(b) The first few steps of the Heun method is shown in the following table

t	*x*	*y*
0	2	1
0.1	2.124872	0.981964
0.2	2.259707	0.968014
0.3	2.404809	0.958274
0.4	2.560393	0.95292
0.5	2.72655	0.952178

A plot of the entire simulation is shown below:



Notice that in contrast to the Euler method, the peaks are stable manner as time progresses. This result is also reinforced when a state-space plot of the calculation is displayed.



(c) The first few steps of the 4th-order RK method is shown in the following table

t	*x*	*y*
0	2	1
0.1	2.124862	0.982012
0.2	2.259682	0.968111
0.3	2.404758	0.958421
0.4	2.560303	0.953116
0.5	2.726403	0.952425

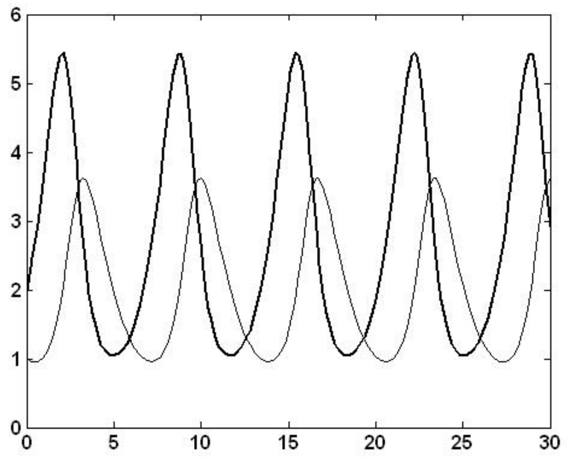
The results are quite close to those obtained with the Heun method in part (*b*). In fact, both the time series and state-space plots are indistinguishable from each other.

(d) In order to solve the problem in MATLAB, a function is first developed to hold the ODEs,

```
function yp=predprey(t,y)
yp=[1.2*y(1)-0.6*y(1)*y(2);-0.8*y(2)+0.3*y(1)*y(2)];
```

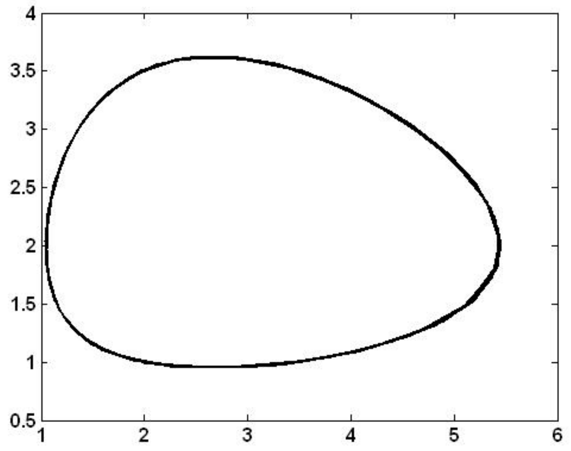
Then, an ODE solver like the function ode45 can be implemented as in

```
>> tspan=[0,30];
>> y0=[2,1];
>> [t,y]=ode45(@predprey,tspan,y0);
>> plot(t,y)
```



The state-space plot can be generated as

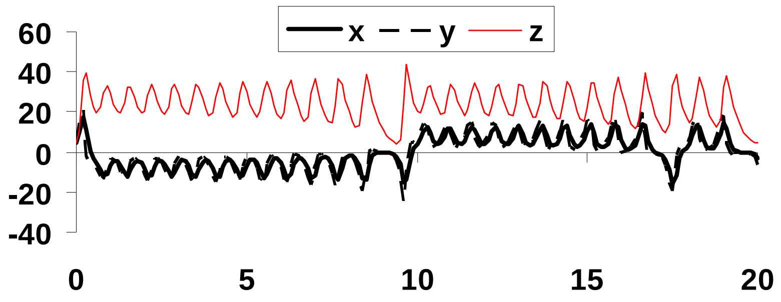
```
>> plot(y(:,1),y(:,2))
```



28.18 Using the step size of 0.1, **(a)** and **(b)** both give unstable results.

(c) The 4th-order RK method yields a stable solution. The first few values are shown in the following table. A time series plot of the results is also shown below. Notice how after about t = 6, this solution diverges from the double precision version in Fig. 28.9.

t	*x*	*y*	*z*
0	5	5	5
0.1	9.781469	17.07946	10.43947
0.2	17.70297	20.8741	35.89687
0.3	10.81088	-2.52924	39.30745
0.4	0.549577	-5.54419	28.07462
0.5	-3.16461	-5.84129	22.36889

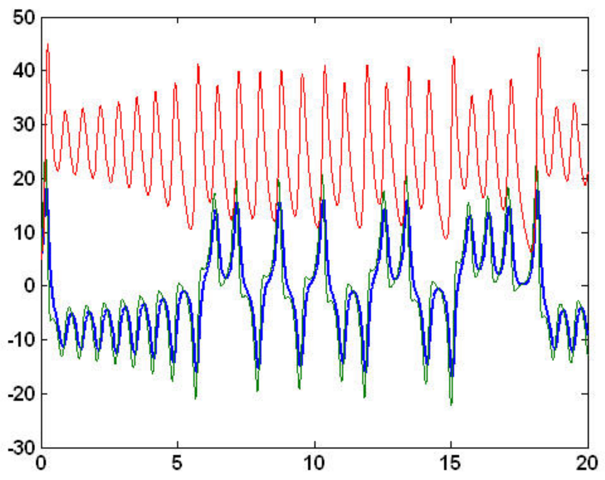


(d) In order to solve the problem in MATLAB, a function is first developed to hold the ODEs,

```
function yp=lorenz(t,y)
yp=[-10*y(1)+10*y(2);28*y(1)-y(2)-y(1)*y(3);-2.666667*y(3)+y(1)*y(2)];
```

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

```
>> tspan=[0,20];
>> y0=[5,5,5];
>> [t,y]=ode45(@lorenz,tspan,y0);
>> plot(t,y)
```



28.19 The second-order equation can be expressed as a pair of first-order equations,

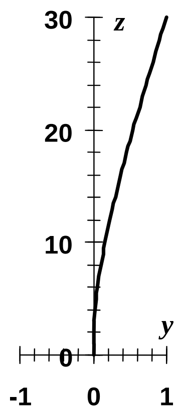
$$\frac{dy}{dz} = w$$

$$\frac{dw}{dz} = \frac{f}{2EI}(L-z)^2$$

We used Euler's method with h = 1 to obtain the solution:

z	*y*	*w*	*dy/dz*	*dw/dz*
0	0	0	0	0.00432
1	0	0.00432	0.00432	0.004037
2	0.00432	0.008357	0.008357	0.003763
3	0.012677	0.01212	0.01212	0.003499

4	0.024797	0.015619	0.015619	0.003245
5	0.040416	0.018864	0.018864	0.003
•				
•				
•				
26	0.8112	0.04524	0.04524	7.68E-05
27	0.85644	0.045317	0.045317	4.32E-05
28	0.901757	0.04536	0.04536	1.92E-05
29	0.947117	0.045379	0.045379	4.8E-06
30	0.992496	0.045384	0.045384	0

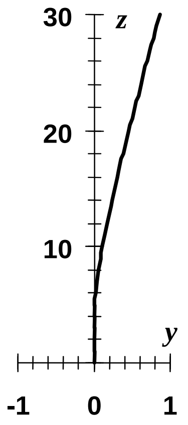


28.20 The second-order equation can be expressed as a pair of first-order equations,

$$\frac{dy}{dz} = w \qquad \frac{dw}{dz} = \frac{200ze^{-2z/30}}{(5+z)2EI}(L-z)^2$$

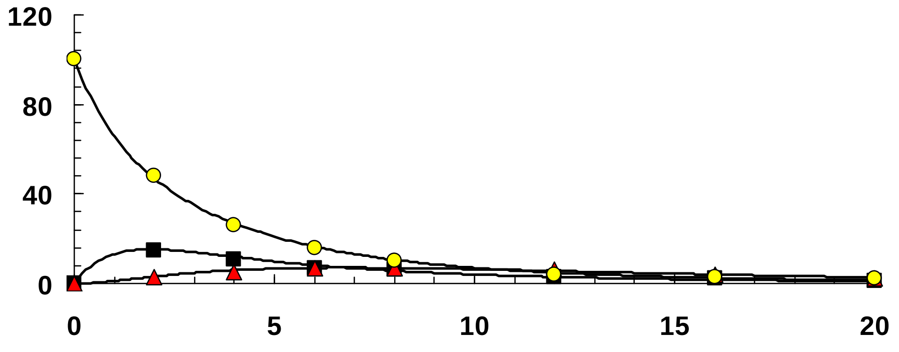
We used Euler's method with *h* = 1 to obtain the solution:

z	*f*(*z*)	*y*	*w*	*dy*/*dz*	*dw*/*dz*
0	0	0	0	0	0
1	31.18357	0	0	0	0.002098
2	50.0099	0	0.002098	0.002098	0.003137
3	61.40481	0.002098	0.005235	0.005235	0.003581
4	68.08252	0.007333	0.008816	0.008816	0.003682
5	71.65313	0.016148	0.012498	0.012498	0.003583
•					
•					
•					
26	29.63907	0.700979	0.040713	0.040713	3.79E-05
27	27.89419	0.741693	0.040751	0.040751	2.01E-05
28	26.24164	0.782444	0.040771	0.040771	8.4E-06
29	24.67818	0.823216	0.04078	0.04078	1.97E-06
30	23.20033	0.863995	0.040782	0.040782	0



28.21 This problem was solved using the Excel spreadsheet in a fashion similar to the last example in Sec. 28.1. We set up Euler's method to solve the 3 ODEs using guesses for the diffusion coefficients. Then we formed a column containing the squared residuals between our predictions and the measured values. Adjusting the diffusion coefficients with the Solver tool minimized the sum of the squares. At first, we assumed that the diffusion coefficients were zero. For this case the Solver did not converge on a credible answer. We then made guesses of 1×10^6 for both. This magnitude was based on the fact that the volumes were of this order of magnitude. The resulting simulation did not fit the data very well, but was much better than when we had guessed zero. When we used Solver, it converged on $E_{12} = 1.243\times10^6$ and $E_{13} = 2.264\times10^6$ which corresponded to a sum of the squares of residuals of 7.734. Some of the Euler calculations are displayed below along with a plot of the fit.

t	*c1*	*c2*	*c3*	*dc1/dt*	*dc2/dt*	*dc3/dt*
0	0	0	100	22.63911	0	-45.2782
0.1	2.2639108	0	95.47218	19.91464	0.351651	-42.203
0.2	4.2553743	0.035165	91.25187	17.46867	0.655521	-39.3905
0.3	6.0022409	0.100717	87.31283	15.27375	0.916677	-36.816
0.4	7.5296162	0.192385	83.63123	13.30513	1.139684	-34.4575
0.5	8.8601294	0.306353	80.18548	11.54045	1.328649	-32.2948

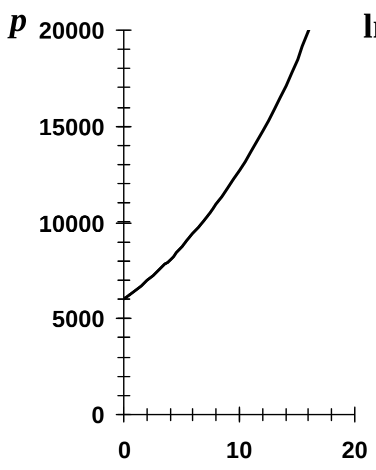


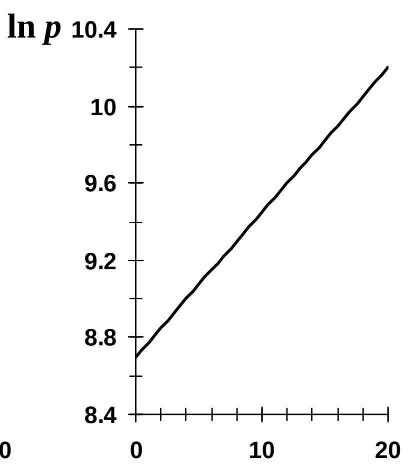
It should be noted that we made up the "measurements" for this problem using the 4th-order RK method with values for diffusive mixing of $E_{12} = 1\times10^6$ and $E_{13} = 2\times10^6$. We then used a random number generator to add some error to this "data."

28.22 The Heun method without corrector can be used to compute

t	p	$k1$	p_{end}	$k2$	ϕ
0	6000	450	6225	466.875	458.4375
0.5	6229.219	467.1914	6462.814	484.7111	475.9512
1	6467.194	485.0396	6709.714	503.2286	494.1341
1.5	6714.261	503.5696	6966.046	522.4535	513.0115
2	6970.767	522.8075	7232.171	542.4128	532.6102
•					
•					
•					
18	23137.43	1735.308	24005.09	1800.382	1767.845
18.5	24021.36	1801.602	24922.16	1869.162	1835.382
19	24939.05	1870.429	25874.26	1940.57	1905.499
19.5	25891.8	1941.885	26862.74	2014.705	1978.295
20	26880.94	2016.071	27888.98	2091.673	2053.872

The results can be plotted. In addition, linear regression can be used to fit a straight line to ln*p* versus *t* to give ln*p* = 8.6995 + 0.075*t*. Thus, as would be expected from a first-order model, the slope is equal to the growth rate of the population.

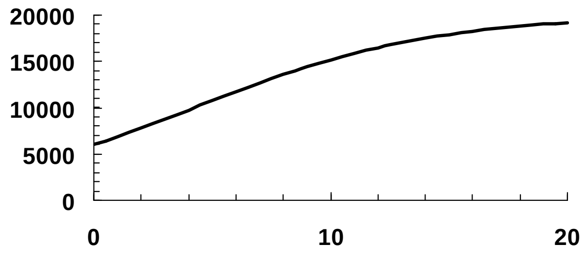




28.23 The Heun method can be used to compute

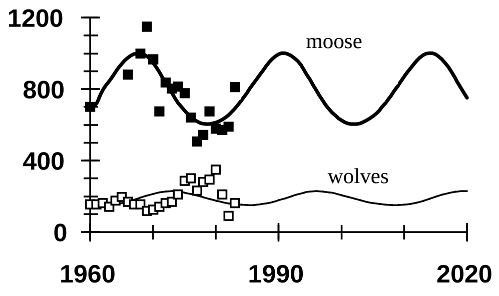
t	p	$k1$	p_{end}	$k2$	ϕ
0	6000	840	6420	871.836	855.918
0.5	6427.959	872.4052	6864.162	901.6652	887.0352
1	6871.477	902.1234	7322.538	928.312	915.2177
1.5	7329.085	928.6622	7793.417	951.3099	939.986
•					
•					
•					
18	18798.01	225.95	18910.99	205.9432	215.9466
18.5	18905.98	206.8344	19009.4	188.3068	197.5706
19	19004.77	189.1412	19099.34	172.02	180.5806
19.5	19095.06	172.7988	19181.46	157.008	164.9034
20	19177.51	157.7327	19256.38	143.1946	150.4637

The results can be plotted.

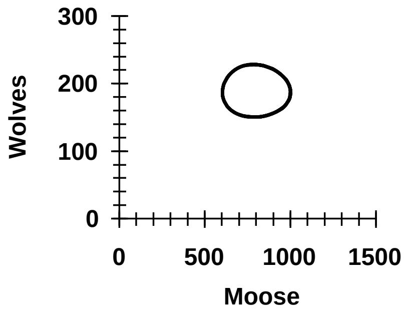


The curve is s-shaped. This shape occurs because initially the population is increasing exponentially since p is much less than $p_{\max}$. However, as p approaches $p_{\max}$, the growth rate decreases and the population levels off.

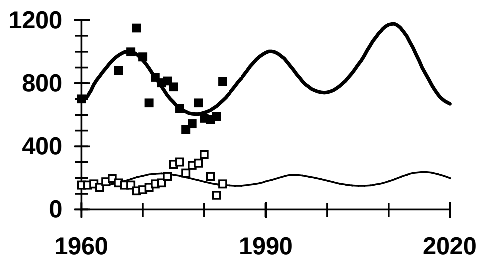
28.24 (a) Nonlinear regression (e.g., using the Excel Solver option) can be used to minimize the sum of the squares of the residuals between the data and the simulation. The resulting estimates are: a = 0.32823, b = 0.01231, c = 0.22445, and d = 0.00029. The fit is:

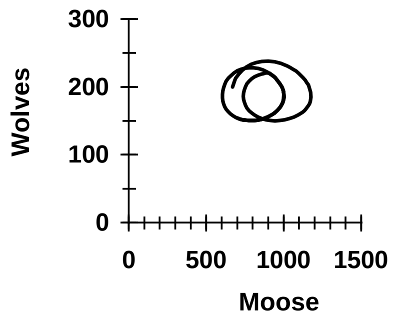


(b) The results in state space are,

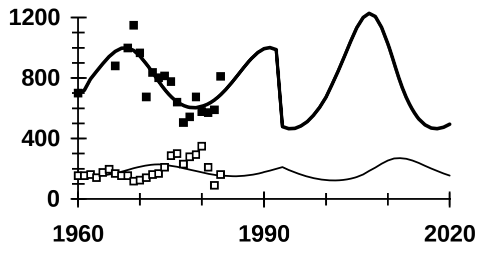


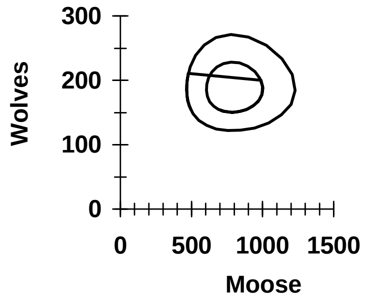
(c)





(d)



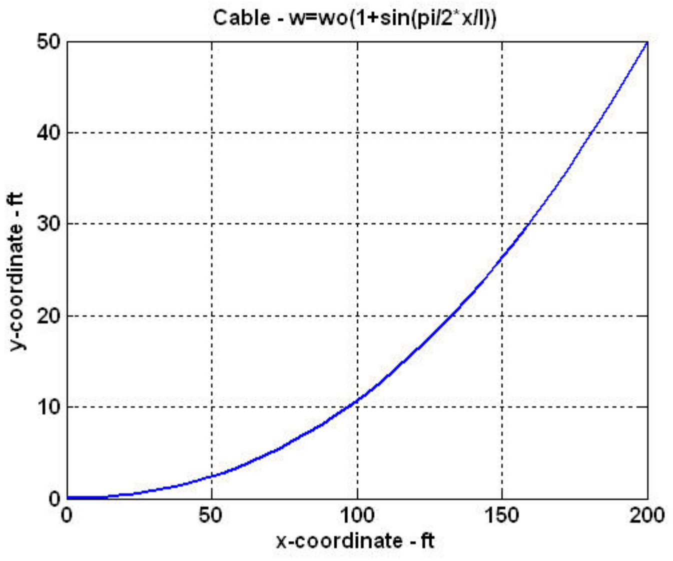


28.25 <u>Main Program:</u>

```
% Hanging static cable - w=w(x)
% Parabolic solution w=w(x)
% CUS Units (lb,ft,s)
% w = wo(1+sin(pi/2*x/l)
% Independent Variable x, xs=start x, xf=end x
% initial conditions [y(1)=cable y-coordinate, y(2)=cable slope];
es=0.5e-7;
xspan=[0,200];
ic=[0 0];
global wToP
wToP=0.0025;
[x,y]=ode45(@slp,xspan,ic);
yf(1)=y(length(x));
wTo(1)=wToP;
ea(1)=1;
wToP=0.002;
[x,y]=ode45(@slp,xspan,ic);
yf(2)=y(length(x));
wTo(2)=wToP;
ea(2)=abs( (yf(2)-yf(1))/yf(2) );
for k=3:10
  wTo(k)=wTo(k-1)+(wTo(k-1)-wTo(k-2))/(yf(k-1)-yf(k-2))*(50-yf(k-1));
  wToP=wTo(k);
  [x,y]=ode45(@slp,xspan,ic);
  yf(k)=y(length(x));
  ea(k)=abs( (yf(k)-yf(k-1))/yf(k) );
  if (ea(k)<=es)
    plot(x,y(:,1)); grid;
    xlabel('x-coordinate - ft'); ylabel('y-coordinate - ft');
    title('Cable - w=wo(1+sin(pi/2*x/l))');
    break
  end
end
```

Function 'slp':

```
function dxy=slp(x,y)
global wToP
dxy=[y(2);(wToP)*(1+sin(pi/2*x/200))];
```



28.26 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,

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

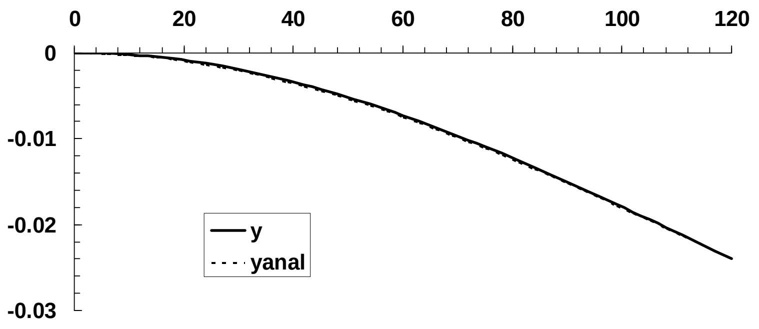
These equations can be simply integrated with any of the one-step techniques from this part of the book. Before determining the solution, we should first express all of the parameters in common units. We can do this by expressing all the length dimensions in either feet or inches. Because the deflection should be small, we will use inches:

L =120 in $\qquad$ P = 1 kip $\qquad$ E = 30,000 ksi $\qquad$ I = 800 in^4

The following shows the results of using the simple Euler's method with h = 0.25 to obtain the solution. Note that we have included the analytical solution in the last column.

x	*y*	*z*	*dydx*	*dzdx*	*yanal*
0	0	0	0	-5.0000E-06	0
3	0	-1.5000E-05	-1.5000E-05	-4.8750E-06	-2.2313E-05
6	-4.5000E-05	-2.9625E-05	-2.9625E-05	-4.7500E-06	-8.8500E-05
9	-1.3388E-04	-4.3875E-05	-4.3875E-05	-4.6250E-06	-1.9744E-04
12	-2.6550E-04	-5.7750E-05	-5.7750E-05	-4.5000E-06	-3.4800E-04
•					
•					
•					
108	-2.0318E-02	-3.0375E-04	-3.0375E-04	-5.0000E-07	-2.0412E-02
111	-2.1229E-02	-3.0525E-04	-3.0525E-04	-3.7500E-07	-2.1305E-02
114	-2.2145E-02	-3.0638E-04	-3.0638E-04	-2.5000E-07	-2.2202E-02
117	-2.3064E-02	-3.0713E-04	-3.0713E-04	-1.2500E-07	-2.3100E-02
120	-2.3985E-02	-3.0750E-04	-3.0750E-04	0.000E+00	-2.4000E-02

Although there is a discrepancy, the results are fairly close. As shown below, the agreement between the approaches can be seen when the numerical and analytical solutions are plotted on the same graph. Note that if a higher-order approach like the 4th-order RK method had been used, the results would be almost indistinguishable.



28.27 This is a boundary-value problem because the values of only one of the variables are known at two separate points; i.e., at $x = 0$, $y = 0$ and at $x = L$, $y = 0$. We can either use finite differences or the shooting method to obtain results.

Before determining the solutions, we should express all of the parameters in common units. We can do this by expressing all the length dimensions in either feet or inches. Because the deflection should be small, we will use inches:

L =120 in $\quad$ w = 0.0833333 kip/in $\quad$ E = 30,000 ksi $\quad$ $I = 800$ in^4

(a) Substituting the finite difference approximation for the second derivative gives

$$EI\frac{y_{i-1}-2y_i+y_{i+1}}{(\Delta x)^2}=\frac{wLx_i}{2}-\frac{wx_i^2}{2}$$

which can be expressed as

$$y_{i-1}-2y_i+y_{i+1}=\frac{(\Delta x)^2}{EI}\left(\frac{wLx_i}{2}-\frac{wx_i^2}{2}\right)$$

Substituting the parameter values with Δx = 24 inches yields

$$y_{i-1} - 2y_i + y_{i+1} = 1.2\times10^{-4} x_i - 1\times10^{-6} x_i^2$$

Writing this equation for the four internal nodes gives

$$\begin{bmatrix} -2 & 1 & & \\ 1 & -2 & 1 & \\ & 1 & -2 & 1 \\ & & 1 & -2 \end{bmatrix} \begin{Bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{Bmatrix} = \begin{Bmatrix} 2.304\times10^{-3} \\ 3.456\times10^{-3} \\ 3.456\times10^{-3} \\ 2.304\times10^{-3} \end{Bmatrix}$$

These equations can be solved for the displacements. The results together with the analytical solution are tabulated as

x	*y* (FD)	*y* (analytical)
0	0	0
24	-0.00576	-0.005568
48	-0.009216	-0.008928
72	-0.009216	-0.008928
96	-0.00576	-0.005568
120	0	0

Thus, the results are pretty close. If a finer grid is used the results would be improved.

(b) In order to implement the shooting method, the second-order differential equation is first expressed as a pair of first-order ODEs,

$$\frac{dy}{dx} = z$$

$$\frac{dz}{dx} = \frac{wLx}{2EI} - \frac{wx^2}{2EI}$$

These equations can be integrated using two guesses for the initial condition of *z*. For our cases we used $z(0) = -1\times10^{-4}$ and $z(0) = -2\times10^{-4}$. We solved the ODEs with Euler's method using a step size of 3 in. The results are

z(0)	-1×10^{-4}	-2×10^{-4}
y(10)	1.64821875E-02	4.48218750E-03

These values can then be used to derive the correct initial condition that corresponds with *y*(120) = 0. Using linear interpolation, the result is

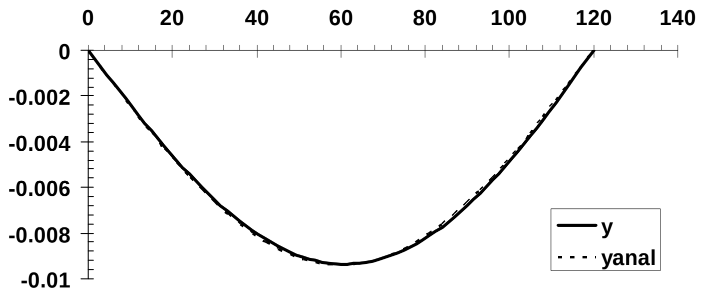
$$z(0) = -1\times10^{-4} + \frac{-2\times10^{-4} - (-1\times10^{-4})}{4.4821875\times10^{-3} - 1.64821875\times10^{-2}}(0 - 1.64821875\times10^{-2}) = -2.3735156\times10^{-4}$$

Using this value, the results of the final shot can be tabulated along with the analytical solution:

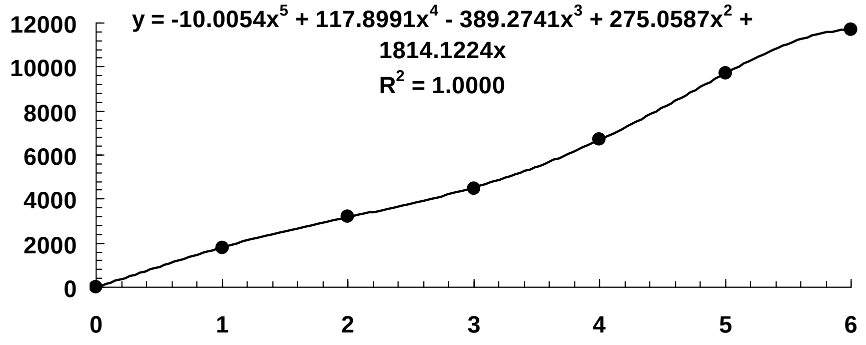
x	*y*	*z*	*dydx*	*dzdx*	*yanal*
0	0	-2.3735E-04	-2.3735E-04	0	0
3	-7.1205E-04	-2.3735E-04	-2.3735E-04	6.0938E-07	-7.4907E-04
6	-1.4241E-03	-2.3552E-04	-2.3552E-04	1.1875E-06	-1.4927E-03
9	-2.1307E-03	-2.3196E-04	-2.3196E-04	1.7344E-06	-2.2256E-03

12	-2.8266E-03	-2.2676E-04	-2.2676E-04	2.2500E-06	-2.9430E-03
•					
•					
•					
108	-3.0426E-03	2.4499E-04	2.4499E-04	2.2500E-06	-2.9430E-03
111	-2.3076E-03	2.5174E-04	2.5174E-04	1.7344E-06	-2.2256E-03
114	-1.5524E-03	2.5695E-04	2.5695E-04	1.1875E-06	-1.4927E-03
117	-7.8152E-04	2.6051E-04	2.6051E-04	6.0938E-07	-7.4907E-04
120	0.0000E+00	2.6234E-04	2.6234E-04	0.0000E+00	0.0000E+00

Note that although there is a discrepancy, the results are fairly close. In particular, notice that the end displacement is effectively zero. As shown below, the close agreement between the approaches can be seen when the numerical and analytical solutions are plotted on the same graph. In fact, if a higher-order approach like the 4th-order RK method had been used, the results would be almost indistinguishable.



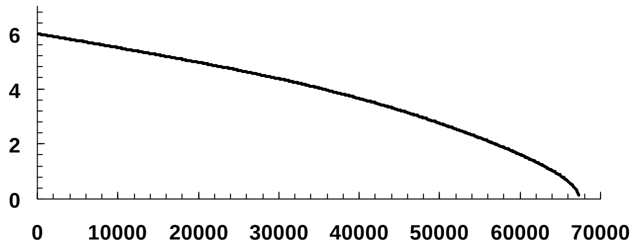
28.28 This problem can be approached in a number of ways. The simplest way is to fit the area-depth data with polynomial regression. A fifth-order polynomial with a zero intercept yields a perfect fit:



This polynomial can then be substituted into the differential equation to yield

$$\frac{dh}{dt} = -\frac{\pi d^2}{4(-10.0054h^5 + 117.8991h^4 - 389.2741h^3 + 275.0587h^2 + 1814.1224h)}\sqrt{2g(h+e)}$$

This equation can then be integrated numerically. This is a little tricky because a singularity occurs as the lake's depth approaches zero. Therefore, the software to solve this problem should be designed to terminate just prior to this occurring. For example, the software can be designed to terminate when a negative area is detected. As displayed below, the results indicate that the reservoir will empty in a little over 67,300 s.



28.29 The parameters can be substituted into force balance equations to give

$$\begin{aligned}
\left(0.45-\omega^2\right)X_1 \quad -0.2 \quad X_2 \quad &= 0\\
-0.24 \quad X_1+\left(0.42-\omega^2\right) X_2 \quad -0.18 \quad X_3 &= 0\\
-0.225 \quad X_2 \ +\left(0.225-\omega^2\right) X_3 &= 0
\end{aligned}$$

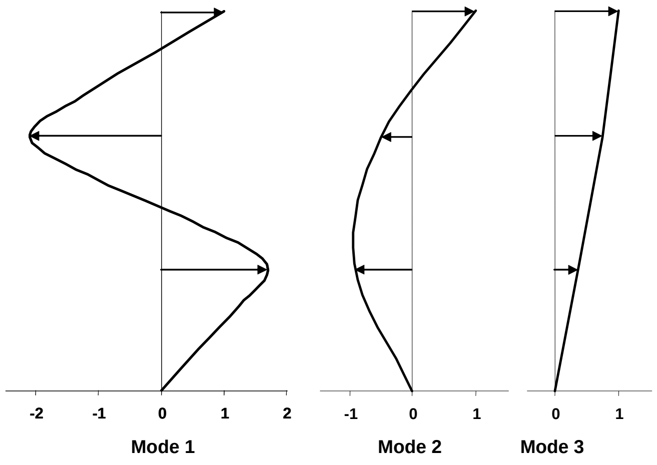
A MATLAB session can be conducted to evaluate the eigenvalues and eigenvectors as

```
>> A = [0.450 -0.200 0.000;-0.240 0.420 -0.180;0.000 -0.225 0.225];
>> [v,d] = eig(A)
v =
   -0.5879   -0.6344    0.2913
    0.7307   -0.3506    0.5725
   -0.3471    0.6890    0.7664
d =
    0.6986         0         0
         0    0.3395         0
         0         0    0.0569
```

Therefore, the eigenvalues are 0.6986, 0.3395 and 0.0569. The corresponding eigenvectors are (normalizing so that the amplitude for the third floor is one),

$$\begin{Bmatrix}1.693748\\-2.10516\\1\end{Bmatrix} \qquad \begin{Bmatrix}-0.92075\\-0.50885\\1\end{Bmatrix} \qquad \begin{Bmatrix}0.380089\\0.746999\\1\end{Bmatrix}$$

A graph can be made showing the three modes



28.30

Errata: Use a value of the infiltration rate of *N* = 0.0001 m/d. The following solution reflects this change, which should appear in the second printing.

(a) The second-order equation can be reexpressed as a pair of first-order equations:

$$\frac{dh}{dx} = z$$
$$\frac{dz}{dx} = -\frac{N}{K\bar{h}} = -\frac{0.0001}{1(7.5)} = -0.0000133333$$

These can be stored in a function

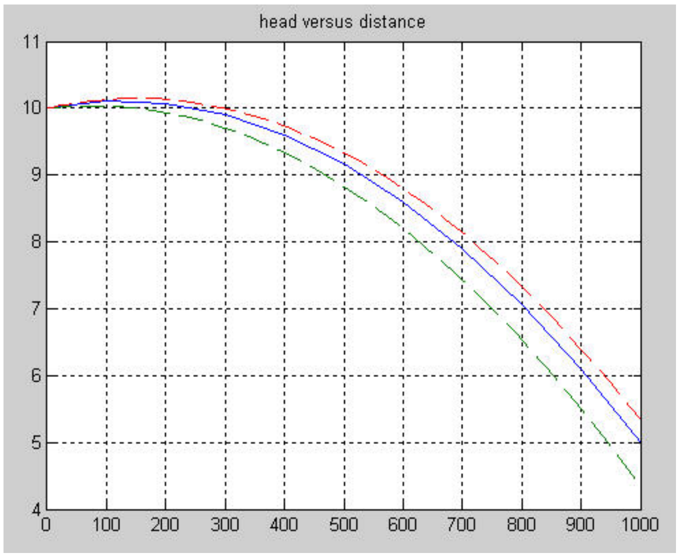
```
function dy=prob2218sys(x,y)
hb=7.5;N=0.0001;K=1;
dy=[y(2);-N/K/hb];
```

The solution was then generated with the following script. Note that we have generated a plot of all the shots:

```
xi=0;xf=1000;
za1=0.001;za2=0.002;ha=10;hb=5;
[x1,h1]=ode45(@prob2218sys,[xi xf],[ha za1]);
hb1=h1(length(h1));
[x2,h2]=ode45(@prob2218sys,[xi xf],[ha za2]);
hb2=h2(length(h2));
za=za1+(za2-za1)/(hb2-hb1)*(hb-hb1);
[x,h]=ode45(@prob2218sys,[xi:(xf-xi)/10:xf],[ha za]);
plot(x,h(:,1),x1,h1(:,1),'--',x2,h2(:,1),'--')
grid;title('head versus distance')
disp('results:')
fprintf('1st shot:  za1 = %8.4g hb1 = %8.4g\n',za1,hb1)
fprintf('2nd shot:  za2 = %8.4g hb2 = %8.4g\n',za2,hb2)
fprintf('Final shot: za = %8.4g   h = %8.4g\n',za,h(length(h)))
z=[x';h(:,1)'];
fprintf('\n     x         h\n');
fprintf('%6d   %10.5f\n',z);
```

The results are

```
results:
1st shot:  za1 =    0.001 hb1 =    4.333
2nd shot:  za2 =    0.002 hb2 =    5.333
Final shot: za = 0.001667   h =        5
     x         h
     0     10.00000
   100     10.10000
   200     10.06667
   300      9.90000
   400      9.60000
   500      9.16667
   600      8.60000
   700      7.90000
   800      7.06667
   900      6.10000
  1000      5.00000
```



(b) We can substitute a centered difference for the second derivative to give

$$K\bar{h}\frac{h_{i-1} - 2h_i + h_{i+1}}{(\Delta x)^2} + N = 0$$

Collecting terms,

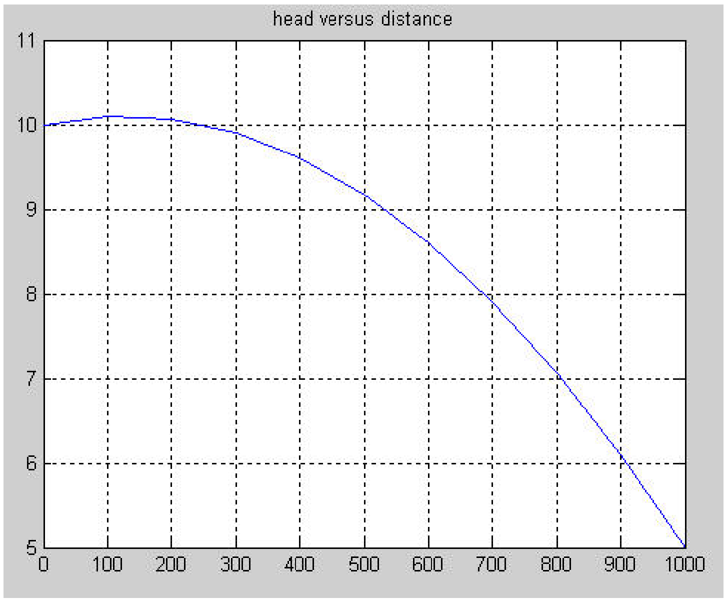
$$-h_{i-1} + 2h_i - h_{i+1} = \frac{N(\Delta x)^2}{K\bar{h}}$$

Substituting the parameter values gives a tridiagonal system of linear algebraic equations. These can be solved with MATLAB as in the following script,

```
h0=10;hn=5;hb=(h0+hn)/2;N=0.0001;K=1;dx=100;
L=1000;n=L/dx-1;
rhs=N*dx^2/K/hb;
a=-1;b=2;c=-1;
A = b*diag(ones(n,1)) + c*diag(ones(n-1,1),1) + a*diag(ones(n-1,1),-1);
b = rhs*ones(n,1);
b(1)=b(1)+h0;
b(n)=b(n)+hn;
h=A\b;
h=[h0 h' hn];
x=[0:dx:L];
z=[x;h];
disp('Results:')
fprintf('\n     x         h\n');
fprintf('%6d   %10.5f\n',z);
plot(x,h);grid;title('head versus distance')
```

The output is

```
Results:
     x         h
     0     10.00000
   100     10.10000
   200     10.06667
   300       9.90000
   400       9.60000
   500       9.16667
   600       8.60000
   700       7.90000
   800       7.06667
   900       6.10000
  1000       5.00000
```



28.31

Errata: Use a value of the infiltration rate of N = 0.0001 m/d. The following solution reflects this change, which should appear in the second printing.

(a) The product rule can be used to evaluate the derivative

$$Kh\frac{d^2h}{dx^2}+K\left(\frac{dh}{dx}\right)^2+N=0$$

We can then define a new variable, $z = dh/dx$ and express the second-order ODE as a pair of first-order ODEs,

$$\frac{dh}{dx}=z$$

$$\frac{dz}{dx}=-\frac{1}{h}z^2-\frac{N}{Kh}$$

These equations can be solved iteratively. First,an M-file can be developed to compute the right-hand sides of these equations,

```
function dy=dydxGW(x,y)
dy=[y(2);-1/y(1)*y(2)^2-0.0001/y(1)];
```

Next, we can build a function to hold the residual that we will try to drive to zero as

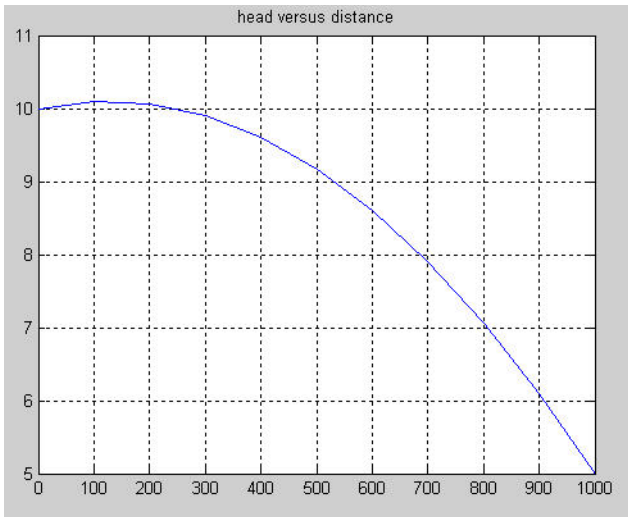
```
function r=resGW(za)
[x,y]=ode45(@dydxGW,[0 1000],[10 za]);
r=y(length(x),1)-5;
```

We can then find the root with the `fzero` function (note that we obtain the initial guess from the linear solution obtained in Prob. 28.30),

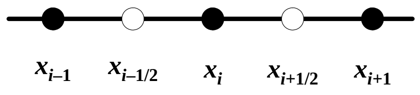
```
>> format long
>> za=fzero(@resGW, 0.0001667)

za =
   0.00125000235608
```

Thus, we see that if we set the initial trajectory *z*(0) = 0.00125, the residual function will be driven to zero and the temperature boundary condition, *T*(1000) = 5 should be satisfied. This can be developing a script to generate the entire solution and plotting head versus distance,

```
[x,h]=ode45(@dydxGW,[0:100:1000],[10 za]);
z=[x';h(:,1)'];
fprintf('\n     x         h\n');
fprintf('%6d   %10.5f\n',z);
plot(x,y(:,1))
     x         h
     0     10.00000
   100     10.07472
   200     10.04988
   300      9.92472
   400      9.69536
   500      9.35414
   600      8.88820
   700      8.27648
   800      7.48332
   900      6.44206
  1000      5.00000
```



(b) A nodal scheme for this problem is shown below:



We can substitute a centered difference for the outer first derivative to give

$$\frac{\left(Kh\frac{dh}{dx}\right)_{i+1/2} - \left(Kh\frac{dh}{dx}\right)_{i-1/2}}{\Delta x} + N = 0 \tag{1}$$

We can then apply centered differences to evaluate the remaining derivatives

$$\left(Kh\frac{dh}{dx}\right)_{i+1/2} = Kh_{i+1/2}\frac{h_{i+1} - h_i}{\Delta x}$$

$$\left(Kh\frac{dh}{dx}\right)_{i-1/2} = Kh_{i-1/2}\frac{h_i - h_{i-1}}{\Delta x}$$

which can be substituted into (1) to give
‘

$$K\frac{\frac{h_i + h_{i+1}}{2}\frac{h_{i+1} - h_i}{\Delta x} - \frac{h_i + h_{i-1}}{2}\frac{h_i - h_{i-1}}{\Delta x}}{\Delta x} + N = 0$$

Collecting terms yields

$$(h_i + h_{i+1})(h_{i+1} - h_i) - (h_i + h_{i-1})(h_i - h_{i-1}) = -\frac{2\Delta x^2}{K}N$$

The terms on the left-hand side can be multiplied out to give

$$h_i h_{i+1} - h_{i+1}h_i - h_i h_i + h_{i+1}h_{i+1} - h_i h_i + h_i h_{i-1} - h_{i-1}h_i + h_{i-1}h_{i-1} = -\frac{2\Delta x^2}{K}N$$

Cancelling and collecting terms gives

$$h_{i-1}^2 - 2h_i^2 + h_{i+1}^2 = -\frac{2\Delta x^2}{K}N$$

An iterative solution similar to Gauss-Seidel can then be developed as

$$h_i = \sqrt{\frac{h_{i-1}^2 + h_{i+1}^2}{2} + \frac{\Delta x^2}{K}N}$$

The results are

x	***h***
0	10
100	10.07472
200	10.04988
300	9.924717

400	9.69536
500	9.354143
600	8.888194
700	8.276473
800	7.483315
900	6.442049
1000	5

Note that these values are quite similar to those obtained with the shooting method in part **(a)**.

28.32

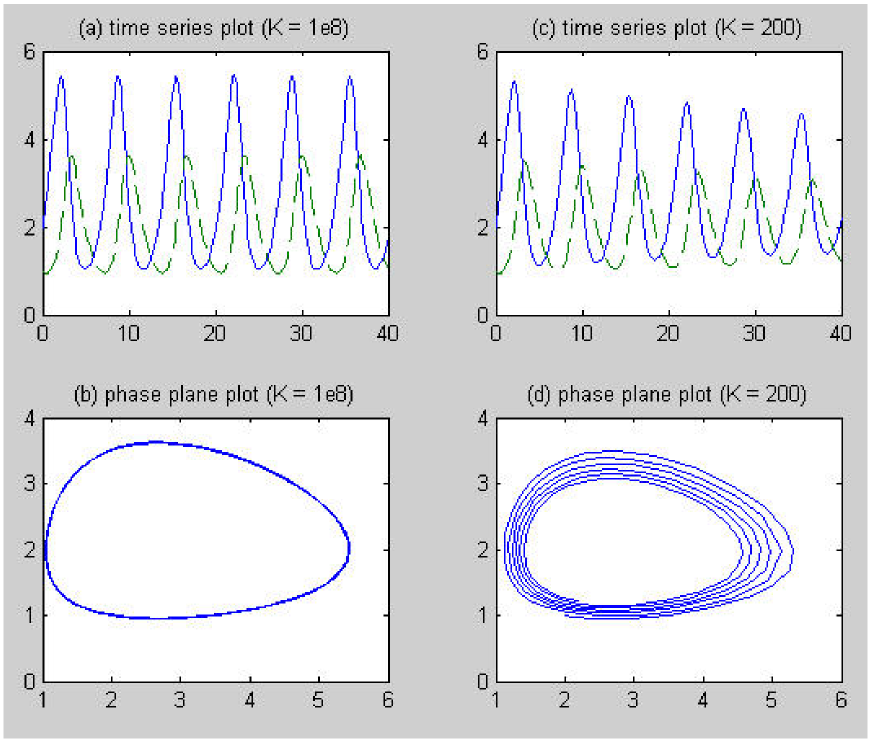
Errata: The carrying capacity for Part (b) should be set to *K* = 200.

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,tspan,y0,[],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)')
```



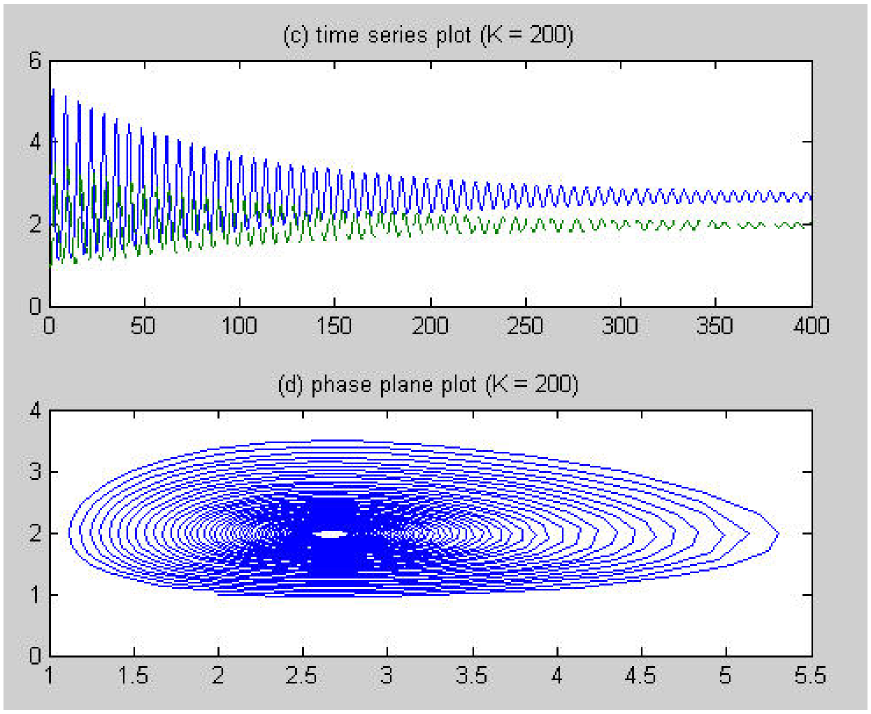
Two things are indicated by these plots:

1. The period of the oscillations seems to be unaffected by the introduction of a carrying capcity 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.

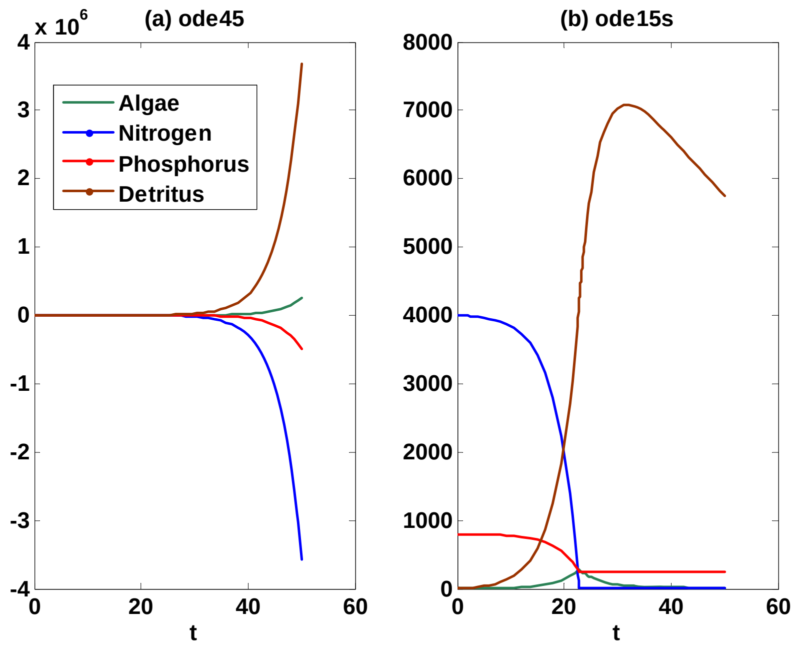


28.33 [Errata for first printing: Solve the equations from *t* = 0 to 50. The value of r_{pc} should be changed to 0.025, use a value of r_{ca} = 40, and set the initial value for phosphorus to *p* = 800.]

The following M-files solve the problem and generate plots of the results:

```
function yp=algaeNM6(t,y,kg,kd,ks,kh,rnc,rpc,rna,rpa,rca,ksp,ksn)
kgnp=kg*min(y(2)/(ksn+y(2)),y(3)/(ksp+y(3)));
yp=[(kgnp-kd-ks)*y(1);rnc*kh*y(4)-rna*kgnp*y(1); ...
rpc*kh*y(4)-rpa*kgnp*y(1);rca*kd*y(1)-kh*y(4)];

tspan=[0 50];y0=[1 4000 800 0];
kg=0.5;kd=0.1;ks=0.15;kh=0.025;rnc=0.18;
rpc=0.025;rna=7.2;rpa=1;rca=40;ksp=2;ksn=15;
```

```
[t y] = ode45(@algaeNM6,tspan,y0,[],kg,kd,ks,kh,rnc,rpc,rna,rpa,rca,ksp,ksn);
[t2 y2] = ...
ode15s(@algaeNM6,tspan,y0,[],kg,kd,ks,kh,rnc,rpc,rna,rpa,rca,ksp,ksn);
subplot(1,2,1);plot(t,y(:,1),t,y(:,2),t,y(:,3),t,y(:,4))
title('Algae Model');legend('Algae','Nitrogen','Phosphorus','Detritus')
xlabel('t');title('(a) ode45')
subplot(1,2,2);plot(t2,y2(:,1),t2,y2(:,2),t2,y2(:,3),t2,y2(:,4))
title('Algae Model');legend('Algae','Nitrogen','Phosphorus','Detritus')
xlabel('t');title('(b) ode15s')
```



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

$$\frac{dS}{dt} = -aSI + \rho R \qquad \frac{dI}{dt} = aSI - rI \qquad \frac{dR}{dt} = rI - \rho R$$

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

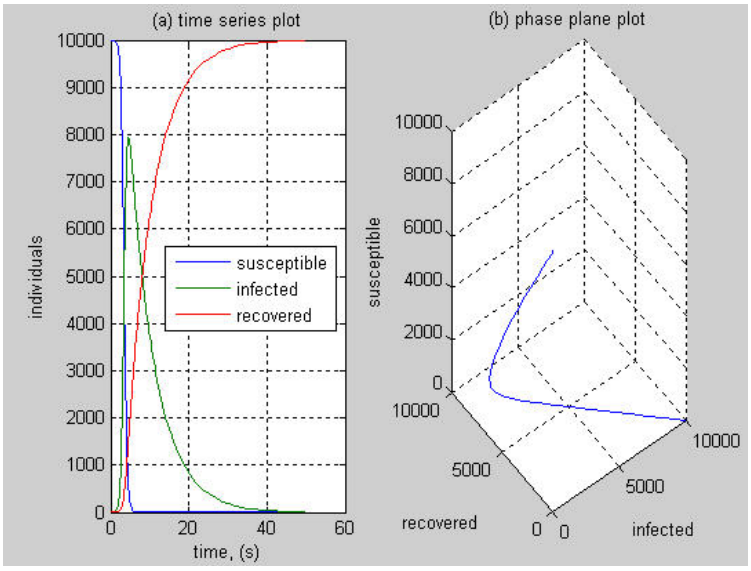
```
function dy=epidemic(t,y,a,r,rho)
dy=[-a*y(1)*y(2)+rho*y(3);a*y(1)*y(2)-r*y(2);r*y(2)-rho*y(3)];
```

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

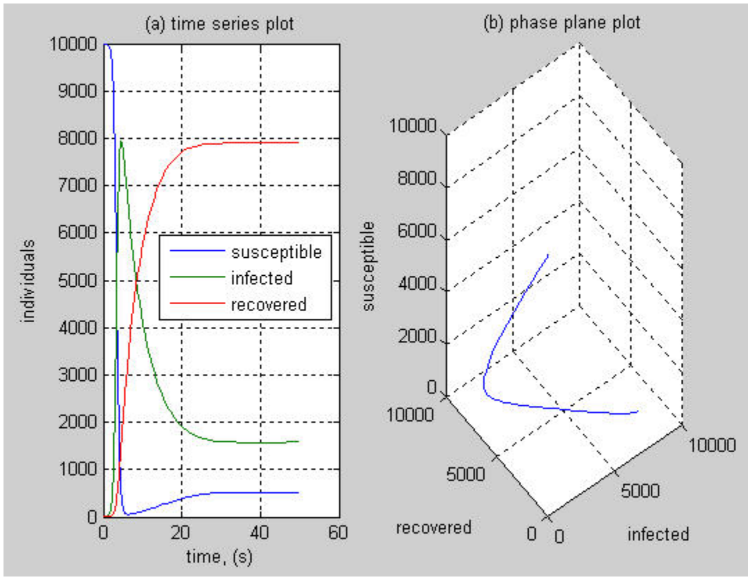
```
tspan = [0 50]; y0 = [10000 1 0];
[t,y]=ode23s(@epidemic,tspan,y0,[],0.002/7,0.15,0);
subplot(1,2,1),plot(t,y)
xlabel('time, (s)')
ylabel('individuals')
title('(a) time series plot'),grid
legend('susceptible','infected','recovered')
subplot(1,2,2),plot3(y(:,3),y(:,2),y(:,1))
xlabel('infected'),ylabel('recovered'),zlabel('susceptible')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



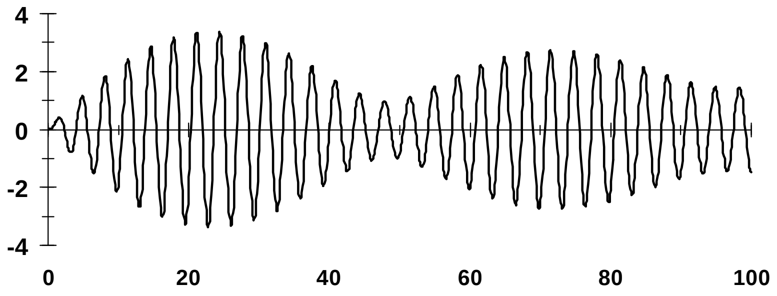
28.35 The second-order equation can be expressed as a pair of first-order equations,

$$\frac{dq}{dt} = i$$

$$\frac{di}{dt} = -0.025i - 4q + \sin 1.8708t$$

The parameters can be substituted and the system solved with the 4th-order RK method in double-precision with *h* = 0.1. A table showing the first few steps and a graph of the entire solution are shown below.

t	*i*	*q*	*k11*	*k12*	*imid*	*qmid*	*k21*	*k22*	*imid*	*qmid*	*k31*	*k32*	*iend*	*qend*	*k41*	*k42*	*phi1*	*phi2*
0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0934	0.0000	0.0047	0.0000	0.0933	0.0047	0.0093	0.0005	0.1839	0.0093	0.0929	0.0031
0.1	0.0093	0.0003	0.1843	0.0093	0.0185	0.0008	0.2734	0.0185	0.0230	0.0012	0.2714	0.0230	0.0364	0.0026	0.3542	0.0364	0.2713	0.0214
0.2	0.0364	0.0025	0.3548	0.0364	0.0542	0.0043	0.4324	0.0542	0.0580	0.0052	0.4287	0.0580	0.0793	0.0083	0.4972	0.0793	0.4290	0.0567
0.3	0.0793	0.0081	0.4978	0.0793	0.1042	0.0121	0.5580	0.1042	0.1072	0.0133	0.5530	0.1072	0.1346	0.0188	0.6017	0.1346	0.5536	0.1061
0.4	0.1347	0.0187	0.6021	0.1347	0.1648	0.0255	0.6399	0.1648	0.1667	0.0270	0.6338	0.1667	0.1981	0.0354	0.6583	0.1981	0.6346	0.1659



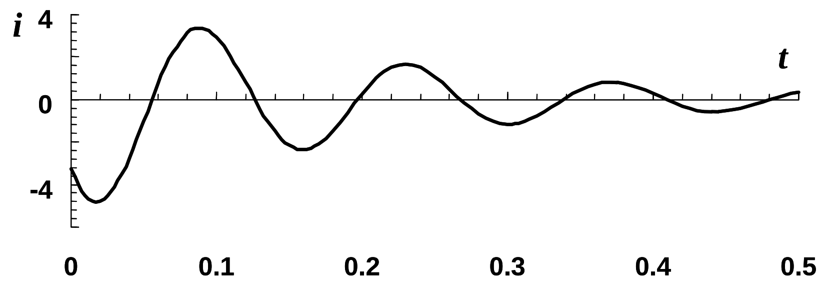
28.36 The second-order equation can be expressed as a pair of first-order equations,

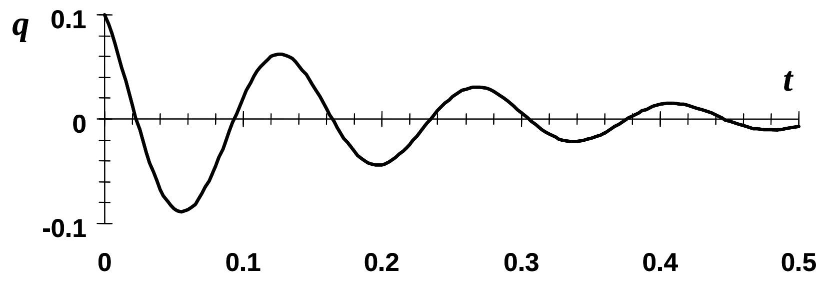
$$\frac{dq}{dt} = i$$

$$\frac{di}{dt} = -\frac{R}{L}i - \frac{q}{CL}$$

The parameters can be substituted and the system solved with the 4th-order RK method with *h* = 0.005. A table showing the first few steps and a graph of the entire solution are shown below.

t	*i*	*q*	*k11*	*k12*	*imid*	*qmid*	*k21*	*k22*	*imid*	*qmid*	*k31*	*k32*	*iend*	*qend*	*k41*	*k42*	*phi1*	*phi2*
0.000	-3.282	0.100	-167.18	-3.282	-3.699	0.092	-146.60	-3.699	-3.648	0.091	-145.02	-3.648	-4.007	0.082	-123.45	-4.007	-145.6	-3.664
0.005	-4.010	0.082	-123.26	-4.010	-4.318	0.072	-100.13	-4.318	-4.260	0.071	-99.17	-4.260	-4.506	0.060	-75.70	-4.506	-99.60	-4.279
0.010	-4.508	0.060	-75.499	-4.508	-4.696	0.049	-51.07	-4.696	-4.635	0.049	-50.74	-4.635	-4.761	0.037	-26.61	-4.761	-50.96	-4.655
0.015	-4.763	0.037	-26.396	-4.763	-4.828	0.025	-1.92	-4.828	-4.767	0.025	-2.21	-4.767	-4.774	0.013	21.39	-4.774	-2.21	-4.788
0.020	-4.774	0.013	21.594	-4.774	-4.720	0.001	44.92	-4.720	-4.661	0.001	44.07	-4.661	-4.553	-0.010	66.00	-4.553	44.26	-4.681





28.37 The equation can be solved analytically as

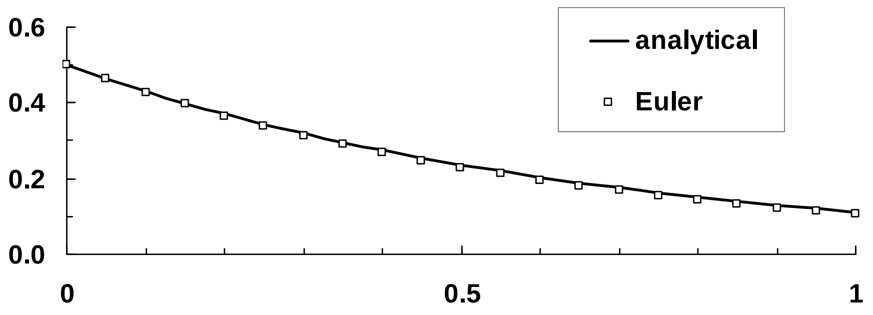
$$\frac{di}{dt} = -\frac{R}{L} i$$
$$\frac{di}{i} = -\frac{R}{L} dt$$
$$\ln i = -(R/L)t + C$$
$$C = \ln 0.5$$
$$\ln(i/0.5) = -(R/L)t$$
$$i = 0.5e^{-2t}$$

The numerical solution can be obtained by expressing the equation as

$$\frac{di}{dt} = -1.5i$$

and using Euler's method with $h = 0.05$ to solve for the current. Some selected values are shown, along with a plot of both the analytical and numerical result. A better match would be obtained by using a smaller step or a higher-order method.

t	*analytical*	*Euler*	*dildt*
0	0.500000	0.500000	-0.750000
0.05	0.463872	0.462500	-0.693750
0.1	0.430354	0.427813	-0.641719
0.15	0.399258	0.395727	-0.593590
0.2	0.370409	0.366047	-0.549071
0.25	0.343645	0.338594	-0.507890

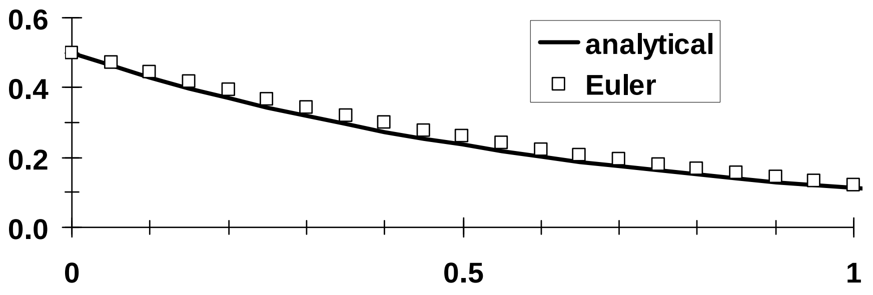


28.38 The numerical solution can be obtained by expressing the equation as

$$\frac{di}{dt} = -1.5(i - i^3)$$

and using Euler's method with h = 0.05 to solve for the current. Some selected values are shown, along with a plot of the numerical result. Note that the table and plot also show the analytical solution for the linear case computed in Prob. 28.19.

t	*analytical*	*Euler*	*dildt*
0	0.500000	0.500000	-0.562500
0.05	0.463872	0.471875	-0.550207
0.1	0.430354	0.444365	-0.534931
0.15	0.399258	0.417618	-0.517175
0.2	0.370409	0.391759	-0.497451
0.25	0.343645	0.366887	-0.476253



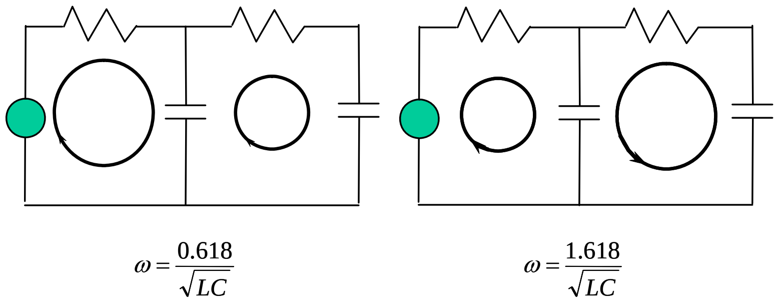
28.39 Using an approach similar to Sec. 28.3, the system can be expressed in matrix form as

$$\begin{bmatrix} 1-\lambda & -1 \\ -1 & 2-\lambda \end{bmatrix} \begin{Bmatrix} i_1 \\ i_2 \end{Bmatrix} = \{0\}$$

A package like MATLAB can be used to evaluate the eigenvalues and eigenvectors as in

```
>> a=[1 -1;-1 2];
>> [v,d]=eig(a)
v =
    0.8507   -0.5257
    0.5257    0.8507
d =
    0.3820         0
         0    2.6180
```

Thus, we can see that the eigenvalues are λ = 0.382 and 2.618 or natural frequencies of $\omega = 0.618/\sqrt{LC}$ and $1.618/\sqrt{LC}$. The eigenvectors tell us that these correspond to oscillations that coincide (0.8507 0.5257) and which run counter to each other (–0.5257 0.8507).


$\omega = \frac{0.618}{\sqrt{LC}}$
$\omega = \frac{1.618}{\sqrt{LC}}$

28.40 A centered differences can be substituted for the derivative to give

$$\frac{V_{i+1} - 2V_i + V_{i-1}}{\Delta x^2} = -\frac{\rho_v}{\varepsilon}$$

Collecting variables and substituting parameters gives

$$-V_{i+1} + 2V_i - V_{i-1} = 60$$

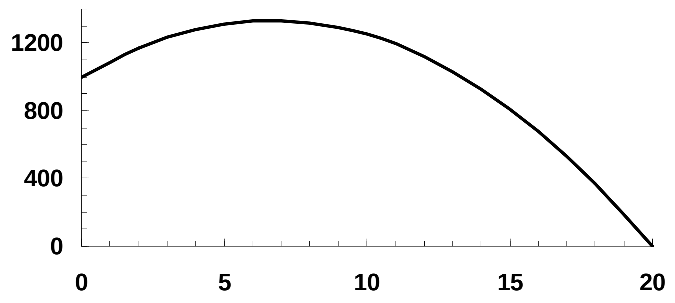
The equations for the first and last node reflect the boundary conditions

$$2V_1 - V_2 = 1060$$

$$-V_8 + 2V_9 = 60$$

The equations can be solved for

x	*V*
0	1000
2	1170
4	1280
6	1330
8	1320
10	1250
12	1120
14	930
16	680
18	370
20	0



28.41 The differential equations to be solved are

linear:

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

$$\frac{dv}{dt} = -\frac{9.81}{1}\theta$$

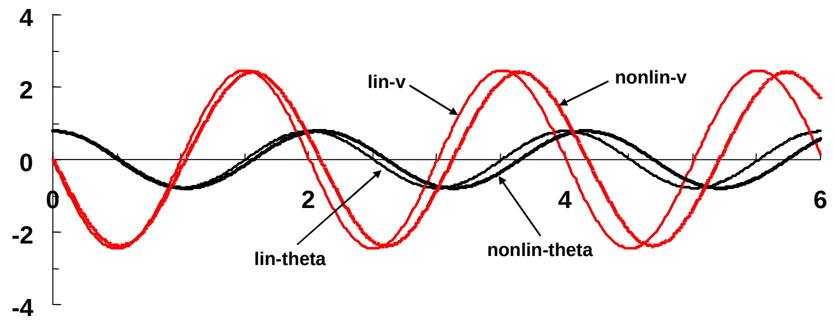
nonlinear:

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

$$\frac{dv}{dt} = -\frac{9.81}{1}\sin\theta$$

A few steps for the 4th-order RK solution of the nonlinear system are contained in the following table and a plot of both solutions is shown below.

t	*theta*	*v*	k_{11}	k_{12}	*theta*	*v*	k_{21}	k_{22}	*theta*	*v*	k_{31}	k_{32}	*theta*	*v*	k_{41}	k_{42}	ϕ_1	ϕ_2
0	0.785	0.000	0.000	-6.937	0.785	-0.035	-0.035	-6.937	0.785	-0.035	-0.035	-6.936	0.785	-0.069	-0.069	-6.934	-0.035	-6.936
0.01	0.785	-0.069	-0.069	-6.934	0.785	-0.104	-0.104	-6.932	0.785	-0.104	-0.104	-6.931	0.784	-0.139	-0.139	-6.927	-0.104	-6.931
0.02	0.784	-0.139	-0.139	-6.927	0.783	-0.173	-0.173	-6.922	0.783	-0.173	-0.173	-6.921	0.782	-0.208	-0.208	-6.915	-0.173	-6.921
0.03	0.782	-0.208	-0.208	-6.915	0.781	-0.242	-0.242	-6.908	0.781	-0.242	-0.242	-6.907	0.780	-0.277	-0.277	-6.898	-0.242	-6.907
0.04	0.780	-0.277	-0.277	-6.898	0.778	-0.311	-0.311	-6.888	0.778	-0.311	-0.311	-6.887	0.777	-0.346	-0.346	-6.876	-0.311	-6.888



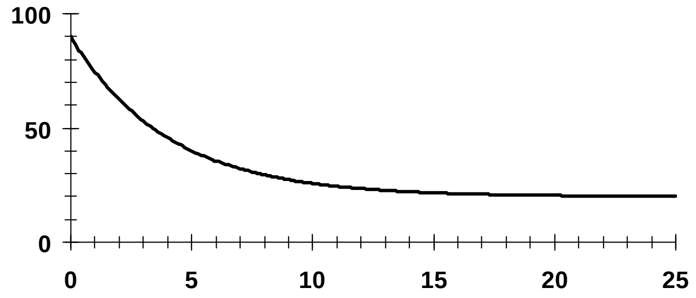
28.42 The differential equation to be solved is

$$\frac{dT}{dt} = 0.25(20 - T)$$

A few steps for the 2nd-order RK solution (Heun without iteration) are shown in the following table and a plot of temperature versus time is shown below.

t	*T*	k_{11}	T_{end}	k_{21}	ϕ
0	90	-17.5	88.25	-17.0625	-17.2813
0.1	88.27188	-17.068	86.56508	-16.6413	-16.8546

0.2	86.58641	-16.6466	84.92175	-16.2304	-16.4385
0.3	84.94256	-16.2356	83.31900	-15.8297	-16.0327
0.4	83.33929	-15.8348	81.75581	-15.4390	-15.6369
0.5	81.77560	-15.4439	80.23121	-15.0578	-15.2509



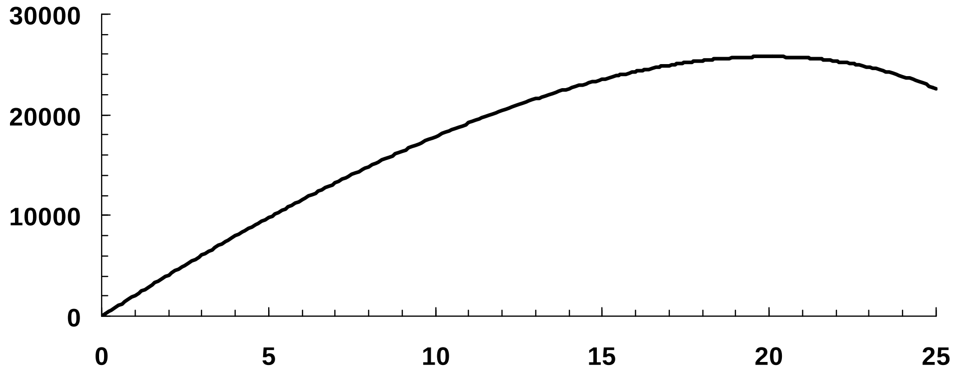
The temperature will drop to 40°C in approximately 5 minutes.

28.43 The differential equation to be solved is

$$\frac{dQ}{dt} = 0.5(12)\frac{100(20-2.5)(20-t)}{100-2.5t}$$

A few steps for the 2nd-order RK solution (Heun without iteration) are shown in the following table and a plot of heat flow versus time is shown below.

t	Q	$k11$	t_{end}	$k21$	ϕ
0	0	2100	0.1	2094.737	2097.368
0.1	209.7368	2094.737	0.2	2089.447	2092.092
0.2	418.946	2089.447	0.3	2084.131	2086.789
0.3	627.625	2084.131	0.4	2078.788	2081.459
0.4	835.7709	2078.788	0.5	2073.418	2076.103
0.5	1043.381	2073.418	0.6	2068.02	2070.719



28.44 The differential equations to be solved are

nonlinear:

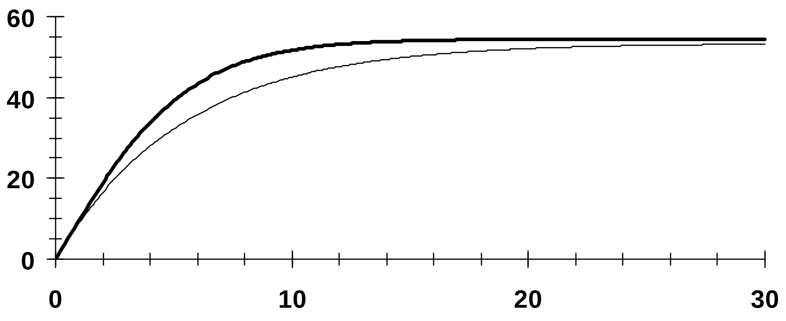
$$\frac{dv}{dt} = 9.8 - \frac{0.225}{68.1}v^2$$

linear:

$$\frac{dv}{dt} = 9.8 - \frac{12.5}{68.1}v$$

A few steps for the solution (Euler) are shown in the following table, which also includes the analytical solution from Example 1.1. A plot of the result is also shown below. Note, the nonlinear solution is the bolder line.

	linear		***nonlinear***		***analytical***
t	***v***	***dvdt***	***v***	***dvdt***	***v***
0	0	9.8	0	9.8	0
0.1	0.98	9.620117	0.98	9.796827	0.971061
0.2	1.942012	9.443537	1.959683	9.787312	1.92446
0.3	2.886365	9.270197	2.938414	9.771473	2.860518
0.4	3.813385	9.100039	3.915561	9.749345	3.779552
0.5	4.723389	8.933005	4.890496	9.720979	4.681871



28.45 The differential equations to be solved are

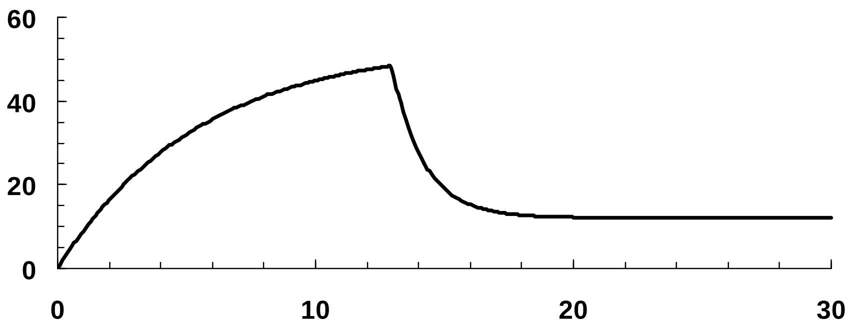
$t < 13$ s: $$\frac{dv}{dt} = 9.8 - \frac{12.5}{68.1}v$$

$t \geq 13$ s: $$\frac{dv}{dt} = 9.8 - \frac{55}{68.1}v$$

The first few steps for the solution (Heun) are shown in the following table, along with the steps when the parachute opens. A plot of the result is also shown below.

t	***v***	***k*11**	***t_{end}***	***v_{end}***	***k*21**	***ϕ***
0	0	9.8	0.1	0.98	9.620117	9.710059
0.1	0.971006	9.621768	0.2	1.933183	9.445157	9.533463
0.2	1.924352	9.446778	0.3	2.86903	9.273379	9.360079
0.3	2.86036	9.274971	0.4	3.787857	9.104725	9.189848
0.4	3.779345	9.106288	0.5	4.689974	8.939138	9.022713
0.5	4.681616	8.940673	0.6	5.575683	8.776563	8.858618
•						
•						
•						
12.8	48.2953	0.935223	12.9	48.38883	0.918057	0.92664
12.9	48.38797	0.918215	13.0	48.47979	-29.354	-14.2179
13	46.96618	-28.1316	13.1	44.15302	-25.8596	-26.9956
13.1	44.26662	-25.9513	13.2	41.67149	-23.8554	-24.9033

13.2	41.77629	-23.94	13.3	39.38228	-22.0065	-22.9733
13.3	39.47896	-22.0846	13.4	37.2705	-20.301	-21.1928



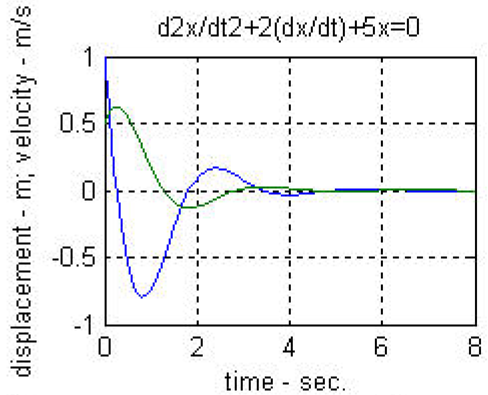
28.46 This problem can be solved with MATLAB:

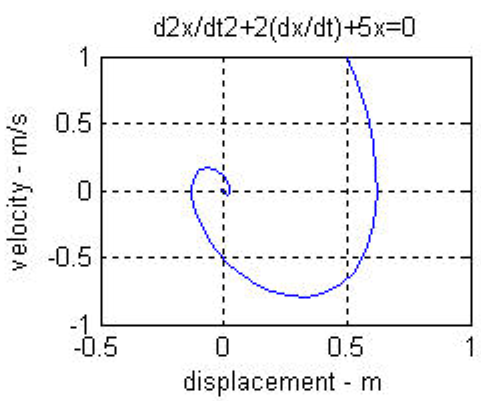
```
%Damped spring mass system
%mass:  m=1 kg
%damping, nonlinear: a abs(dx/dt) (dx/dt), a=2 N/(m/s)^2
%spring, nonlinear: bx^3, b=5 N/m^3
% MATLAB 5 version

%Independent Variable t, tspan=[tstart tstop]
%initial conditions [x(1)=velocity, x(2)=displacement];

t0=0;
tf=8;
tspan=[0 8]; ic=[1 0.5];

% a) linear solution
[t,x]=ode45('kl',tspan,ic);
subplot(221)
plot(t,x); grid; xlabel('time - sec.');
ylabel('displacement - m; velocity - m/s');
title('d2x/dt2+2(dx/dt)+5x=0')
subplot(222)
%Phase-plane portrait
plot(x(:,2),x(:,1)); grid;
xlabel('displacement - m'); ylabel('velocity - m/s');
title('d2x/dt2+2(dx/dt)+5x=0');

% b) nonlinear spring
[t,x]=ode45('nlk',tspan,ic);
subplot(223)
plot(t,x); grid;
xlabel('time - sec.');  ylabel('displacement - m; velocity - m/s');
title('d2x/dt2+2(dx/dt)+5x^3=0')
%Phase-plane portrait
subplot(224)
plot(x(:,2),x(:,1)); grid;
xlabel('displacement - m'); ylabel('velocity - m/s');
title('d2x/dt2+2(dx/dt)+5x=0');
pause

% c) nonlinear damping
[t,x]=ode45('nlc',tspan,ic);
subplot(221)
plot(t,x); grid;
```

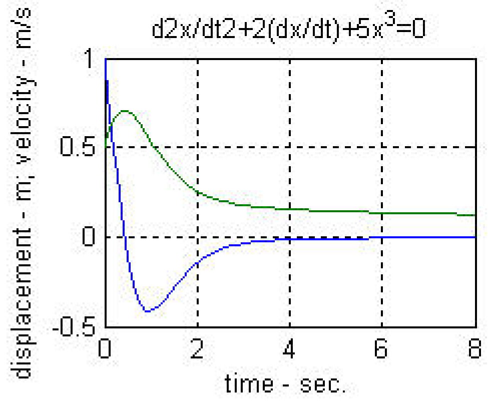
```
xlabel('time - sec.');  ylabel('displacement - m; velocity - m/s');
title('d2x/dt2+2abs(dx/dt)(dx/dt)+5x=0')
%Phase-plane portrait
subplot(222)
plot(x(:,2),x(:,1)); grid;
xlabel('displacement - m'); ylabel('velocity - m/s');
title('d2x/dt2+2abs(dx/dt)(dx/dt)+5x=0');

% d) nonlinear damping and spring
[t,x]=ode45('nlck',tspan,ic);
subplot(223)
plot(t,x); grid;
xlabel('time - sec.');  ylabel('displacement - m; velocity - m/s');
title('d2x/dt2+2abs(dx/dt)(dx/dt)+5x^3=0')
%Phase-plane portrait
subplot(224)
plot(x(:,2),x(:,1)); grid;
xlabel('displacement - m'); ylabel('velocity - m/s');
title('d2x/dt2+2abs(dx/dt)(dx/dt)+5x^3=0');
```

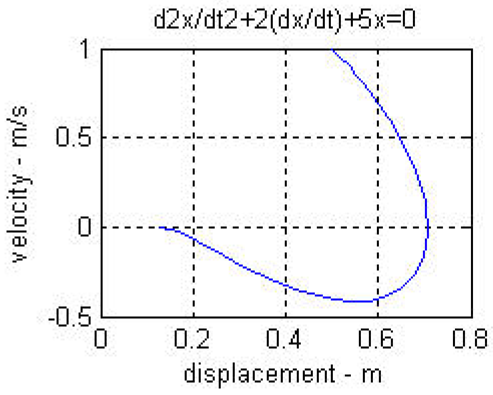
<u>Functions:</u>

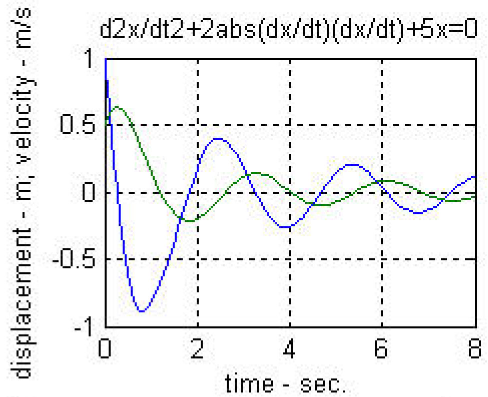
```
%Damped spring mass system - m d2x/dt2 + c dx/dt + k x =0
%mass:     m=1 kg
%    linear-    c=2 N/(m/s)
%    linear-    k=5 N/m
%x(1)=velocity, x(2)=displacement

function dx=kl(t,x);
dx=[-2*x(1)-5*x(2); x(1)];

%Damped spring mass system - m d2x/dt2 + c dx/dt + k x =0
%mass:     m=1 kg
%damping:  linear-    c=2  N/(m/s)
%spring:   nonlinear- kx=bx^3,   b=5 N/m^3

function dx=nlk(t,x);
dx=[-2*x(1)-5*x(2).*x(2).*x(2); x(1)];

%Damped spring mass system - m d2x/dt2 + c dx/dt + k x =0
%mass:     m=1 kg
%damping:  nonlinear- c dx/dt = a sgn(dx/dt) (dx/dt)^2,  a=2 N/(m/s)^2
%spring:   linear-    kx=5x
%x(1)=velocity, x(2)=dispacement

function dx=nlc(t,x);
dx=[-2*abs(x(1))*x(1)-5*x(2); x(1)];

%Damped spring mass system - m d2x/dt2 + c dx/dt + k x =0
%mass:     m=1 kg
%damping:  nonlinear- c dx/dt = a sgn(dx/dt) (dx/dt)^2,  a=2 N/(m/s)^2
%spring:   nonlinear- k x = bx^3,   b=5 N/m^3
%x(1)=velocity, x(2)=dispacement

function dx=nlck(t,x);
dx=[-2*abs(x(1)).*x(1)-5*x(2).*x(2).*x(2); x(1)];
```

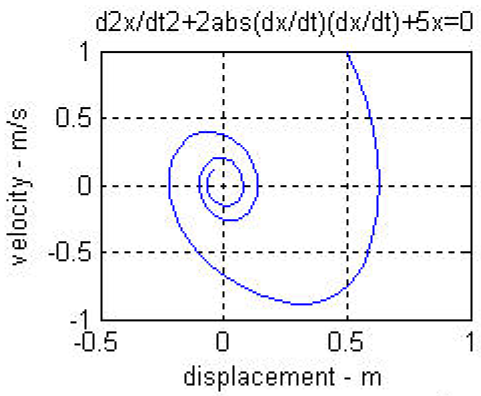

d2x/dt2+2(dx/dt)+5x=0
displacement - m; velocity - m/s
time - sec.
1
0.5
0
-0.5
-1
0
2
4
6
8

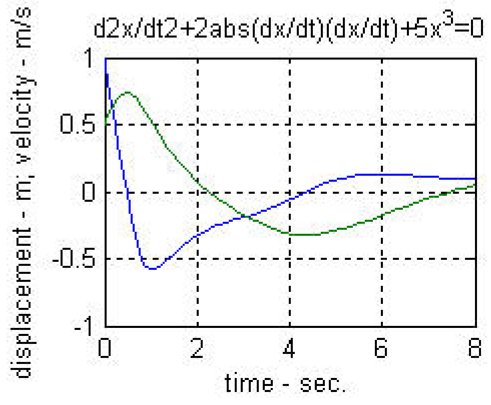

d2x/dt2+2(dx/dt)+5x=0
velocity - m/s
displacement - m
1
0.5
0
-0.5
-1
-0.5
0
0.5
1

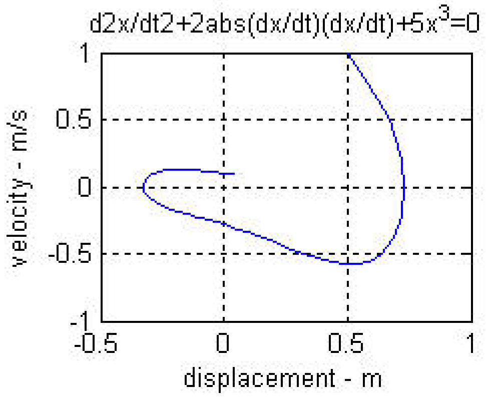

d2x/dt2+2(dx/dt)+5x^3=0
displacement - m; velocity - m/s
time - sec.
1
0.5
0
-0.5
0
2
4
6
8


d2x/dt2+2(dx/dt)+5x=0
velocity - m/s
displacement - m
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8


d2x/dt2+2abs(dx/dt)(dx/dt)+5x=0
displacement - m; velocity - m/s
time - sec.
1
0.5
0
-0.5
-1
0
2
4
6
8


d2x/dt2+2abs(dx/dt)(dx/dt)+5x=0
velocity - m/s
displacement - m
1
0.5
0
-0.5
-1
-0.5
0
0.5
1


d2x/dt2+2abs(dx/dt)(dx/dt)+5x^3=0
displacement - m; velocity - m/s
time - sec.
1
0.5
0
-0.5
-1
0
2
4
6
8

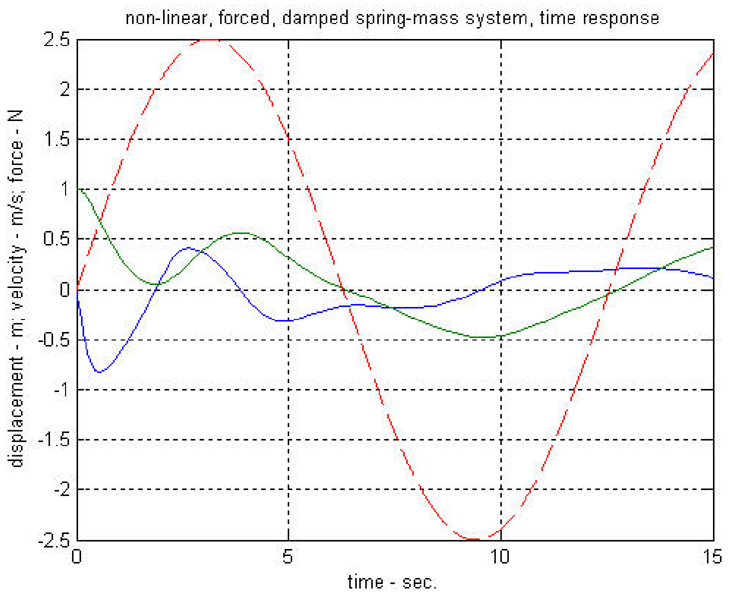

d2x/dt2+2abs(dx/dt)(dx/dt)+5x^3=0
velocity - m/s
displacement - m
1
0.5
0
-0.5
-1
-0.5
0
0.5
1

28.47 This problem can be solved with MATLAB:

```
%Forced damped spring-mass system w/ material damping
%mass:  m=2 kg
%damping, nonlinear air: a abs(dx/dt) dx/dt, a=5 N/(m/s)^2
%spring, linear: kx = 6x
%forcing function: F=Fo(sin(wt)),  Fo=2.5 N, w=0.5 rad/s
%Independent Variable t, tspan=[tstart tstop]
%initial conditions [x(1)=velocity, x(2)=displacement];

tspan=[0 15]; ic=[0 1];
[t,x]=ode45('nlF',tspan,ic);
ts=0:.01:15;
Sin=2.5*sin(0.5*ts);
plot(t,x,ts,Sin,'--'); grid; xlabel('time - sec.');
ylabel('displacement - m; velocity - m/s; force - N');
title('non-linear, forced, damped spring-mass system, time response')
```

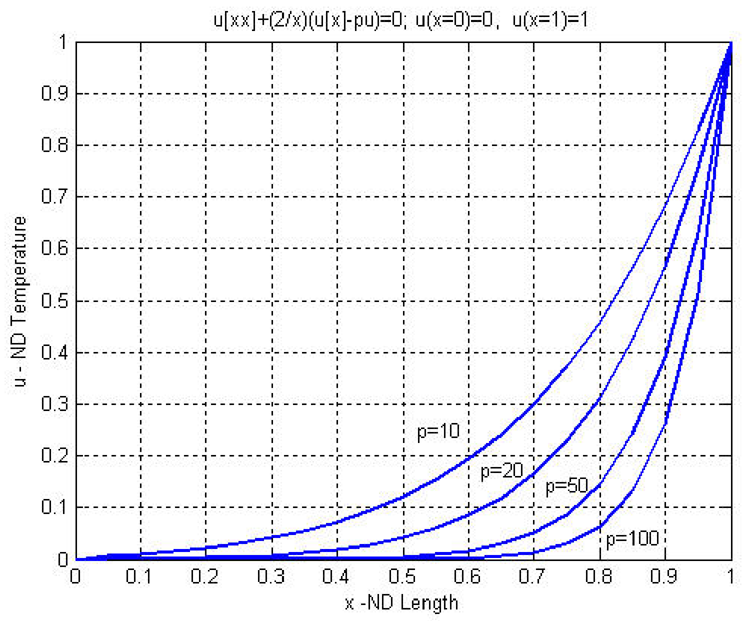
Function nlF:

```
%Forced damped spring-mass system w/ material damping
%mass:  m=2 kg
%damping, nonlinear air: a abs(dx/dt) (dx/dt), a=5 N/(m/s)^2
%spring, linear: kx = 6x
%forcing function: F=Fo(sin(wt),  Fo=2.5 N, w=0.5 rad/s
% x(1)= velocity, x(2)= displacement

function dx=nlF(t,x);
dx=[-2.5*abs(x(1)).*x(1)-3*x(2)+1.25*sin(0.5*t); x(1)];
```



28.48 This problem can be solved with MATLAB:

```
%   ODE Boundary Value Problem
%   Tapered conical cooling fin
%      u[xx]+(2/x)(u[x]-pu)=0
%      BC. u(x=0)=0  u(x=1)=1
%   i=spatial index, from 1 to R
%                 numbering for points is i=1 to i=R for (R-1) dx spaces
%   u(i=1)=0 and u(i=R)=1

R=21;
%Constants
dx=1/(R-1);
dx2=dx*dx;
%Parameters
p(1)=10; p(2)=20; p(3)=50; p(4)=100;
%sizing matrices
u=zeros(1,R); x=zeros(1,R);
a=zeros(1,R); b=zeros(1,R); c=zeros(1,R); d=zeros(1,R);
ba=zeros(1,R); ga=zeros(1,R);
%Independent Variable
x=0:dx:1;
%Boundary Conditions
u(1)=0; u(R)=1;

for k=1:4;
  %/Coefficients
  b(2)=-2-2*p(k)*dx2/dx;
  c(2)=2;
  for i=3:R-2,
    a(i)=1-dx/(dx*(i-1));
    b(i)=-2-2*p(k)*dx2/(dx*(i-1));
    c(i)=1+1/(i-1);
  end
  a(R-1)=1-dx/(dx*(R-2));
  b(R-1)=-2-2*p(k)*dx2/(dx*(R-2));
  d(R-1)=-(1+1/(R-2));
       %Solution by Thomas Algorithm
  ba(2)=b(2);
  ga(2)=d(2)/b(2);
  for i=3:R-1,
    ba(i)=b(i)-a(i)*c(i-1)/ba(i-1);
    ga(i)=(d(i)-a(i)*ga(i-1))/ba(i);
  end
  %back substitution step
  u(R-1)=ga(R-1);
  for i=R-2:-1:2,
    u(i)=ga(i)-c(i)*u(i+1)/ba(i);
  end
  %Plot
  plot(x,u)
  title('u[xx]+(2/x)(u[x]-pu)=0; u(x=0)=0,  u(x=1)=1')
  xlabel('x -ND Length')
  ylabel('u - ND Temperature')
  hold on
end
grid
hold off
gtext('p=10');gtext('p=20');gtext('p=50');gtext('p=100');
```



28.49 The second-order equation can be expressed as a pair of first-order equations,

$$\frac{dx}{dt} = v \qquad \frac{dv}{dt} = -\frac{c}{m}v - \frac{k_1}{m}x - \frac{k_3}{m}x^3 + \frac{P}{m}\cos(\omega t)$$

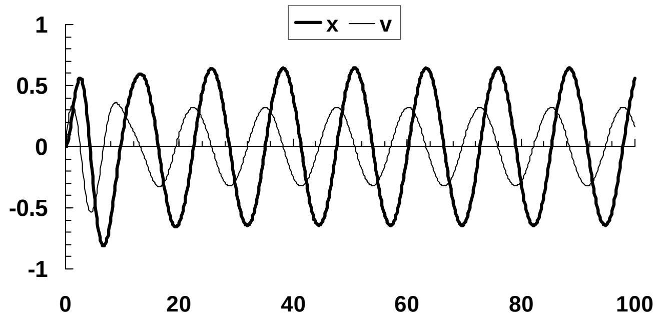
Substituting the parameter values:

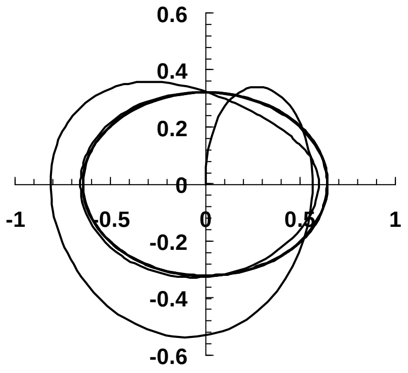
$$\frac{dx}{dt} = v \qquad \frac{dv}{dt} = -0.4v - k_1 x - k_3 x^3 + 0.5\cos(0.5t)$$

(a) Linear (k_1 = 1; k_3 = 0):

$$\frac{dx}{dt} = v$$
$$\frac{dv}{dt} = -0.4v - x + 0.5\cos(0.5t)$$

Here are the results of solving these equations with the initial condition, $x = v = 0$, using the Heun method without corrector and a step-size h = 0.125.

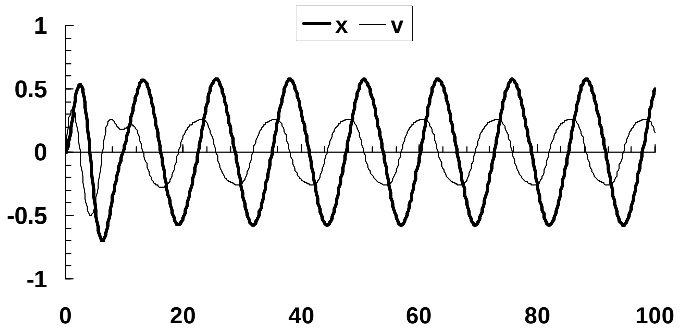


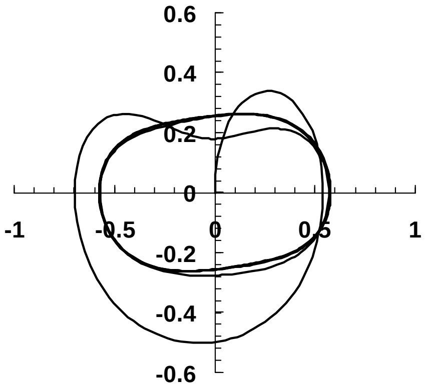


(b) Nonlinear ($k_1 = 1$; $k_3 = 0.5$):

$$\frac{dx}{dt} = v$$
$$\frac{dv}{dt} = -0.4v - x - 0.5x^3 + 0.5\cos(0.5t)$$

Here are the results of solving these equations with the initial condition, $x = v = 0$, using the Heun method without corrector and a step-size $h = 0.125$.





28.50 The following MATLAB M-file can be set up to compute the right-hand-sides of the ODEs,

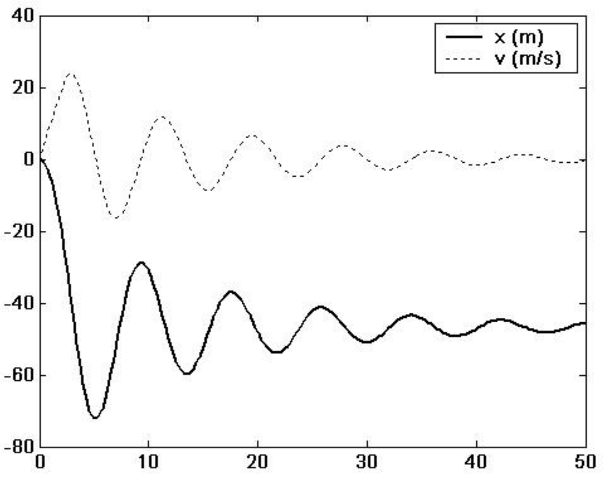
```
function dydt = bungee(t,y,L,cd,m,k,gamma)
```

```
g = 9.81;
cord = 0;
if y(1) > L %determine if the cord exerts a force
  cord = k/m*(y(1)-L)+gamma/m*y(2);
end
dydt = [y(2); g - sign(y(2))*cd/m*y(2)^2 - cord];
```

We can use `ode45` to obtain the solutions and display them on a plot,

```
>> [t,y]=ode45(@bungee,[0 50],[0 0],[],30,0.25,68.1,40,8);
>> plot(t,-y(:,1),'-',t,y(:,2),':')
>> legend('x (m)','v (m/s)')
```

As in the following figure, we have reversed the sign of distance for the plot so that negative distance is in the downward direction.



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

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

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

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

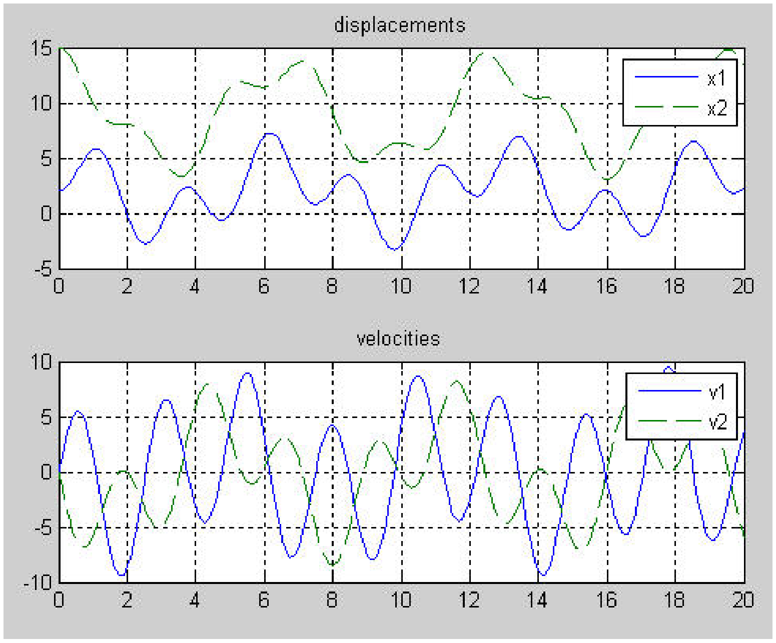
We can develop an M-file to compute them,

```
function dx = dxdtProb2117(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(@dxdtProb2117,[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')
```


displacements
x1
x2
velocities
v1
v2