CHAPTER 22

22.1 **(a)** The analytical solution can be derived by separation of variables

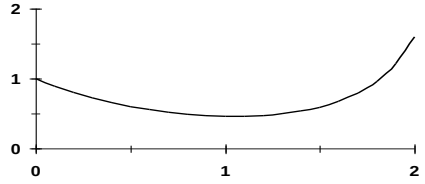
$$\int \frac{dy}{y} = \int (t^2 - 1.1)\, dt$$

$$\ln y = \frac{t^3}{3} - 1.1t + C$$

Substituting the initial conditions yields $C = 0$. Taking the exponential gives the final result

$$y = e^{t^3/3 - 1.1t}$$

The result can be plotted as



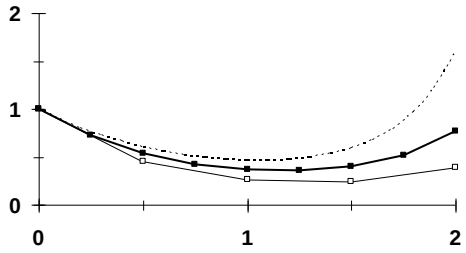
(b) Euler's method with $h = 0.5$

x	y	dy/dx
0	1	–1.1
0.5	0.45	–0.3825
1	0.25875	–0.02588
1.5	0.245813	0.282684
2	0.387155	1.122749

Euler's method with $h = 0.25$ gives

x	y	dy/dx
0	1	–1.1
0.25	0.725	–0.75219
0.5	0.536953	–0.45641
0.75	0.422851	–0.22728
1	0.36603	–0.0366
1.25	0.356879	0.165057
1.5	0.398143	0.457865
1.75	0.51261	1.005997
2	0.764109	2.215916

The results can be plotted along with the analytical solution as



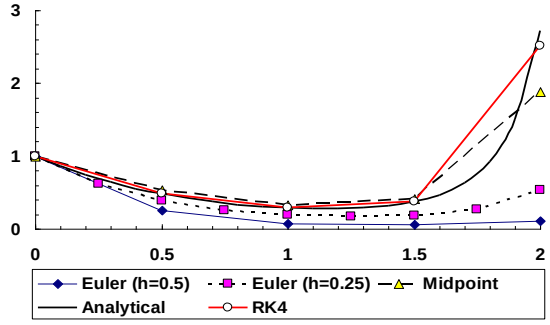
(c) Midpoint method $(h = 0.5)$

t	y	dy/dt	t_m	y_m	dy_m/dt
0	1	–1.1	0.25	0.725	–0.75219
0.5	0.623906	–0.53032	0.75	0.491326	–0.26409
1	0.491862	–0.04919	1.25	0.479566	0.221799
1.5	0.602762	0.693176	1.75	0.776056	1.52301
2	1.364267	3.956374	2.25	2.35336	9.32519

(d) RK4 $(h = 0.5)$

t	y	k_1	t_m	y_m	k_2	y_m	k_3	t_e	y_e	k_4	ϕ
0	1	–1.1	0.25	0.725	–0.7522	0.8120	–0.8424	0.5	0.5788	–0.4920	–0.7969
0.5	0.6016	–0.5113	0.75	0.4737	–0.2546	0.5379	–0.2891	1	0.4570	–0.0457	–0.2741
1	0.4645	–0.0465	1.25	0.4529	0.2095	0.5169	0.2391	1.5	0.5841	0.6717	0.2537
1.5	0.5914	0.6801	1.75	0.7614	1.4943	0.9649	1.8937	2	1.5382	4.4609	1.9861
2	1.5845	4.5949	2.25	2.7332	10.8302	4.2920	17.0071	2.5	10.0880	51.9532	18.7038

All the solutions can be presented graphically as



22.2 **(a)** The analytical solution can be derived by the separation of variables,

$$\int \frac{dy}{\sqrt{y}} = \int (1+2x)\,dx$$

The integrals can be evaluated to give,

$$2\sqrt{y} = x + x^2 + C$$

Substituting the initial conditions yields $C = 2$. Substituting this value and rearranging gives

$$y = \left(\frac{x^2 + x + 2}{2}\right)^2$$

Some selected value can be computed as

x	y
0	1.000000
0.25	1.336914
0.5	1.890625
0.75	2.743164
1	4.000000

(b) Euler's method:

$$y(0.25) = y(0) + f(0,1)h$$
$$f(0,1) = (1+2(0))\sqrt{1} = 1$$
$$y(0.25) = 1 + 1(0.25) = 1.25$$

$$y(0.5) = y(0.25) + f(0.25,1.25)0.25$$
$$f(0.25,1.25) = (1+2(0.25))\sqrt{1.25} = 1.67705$$
$$y(0.5) = 1.25 + 1.67705\ \ (0.25) = 1.66926$$

The remaining steps can be implemented and summarized as

x	y	dy/dx
0	1.00000	1.00000
0.25	1.25000	1.67705
0.5	1.66926	2.58400
0.75	2.31526	3.80400
1	3.26626	5.42184

(c) Heun's method:

Predictor:

$$k_1 = (1+2(0))\sqrt{1} = 1$$
$$y(0.25) = 1+1(0.25) = 1.25$$
$$k_2 = (1+2(0.25))\sqrt{1.25} = 1.6771$$

Corrector:

$$y(0.25) = 1+\frac{1+1.6771}{2}0.25 = 1.33463$$

The remaining steps can be implemented and summarized as

x	y	k_1	x_e	y_e	k_2	dy/dx
0	1.00000	1.0000	0.25	1.25	1.6771	1.3385
0.25	1.33463	1.7329	0.5	1.76785	2.6592	2.1961
0.5	1.88364	2.7449	0.75	2.56987	4.0077	3.3763
0.75	2.72772	4.1290	1	3.75996	5.8172	4.9731
1	3.97099					

(d) Ralston's method:

Predictor:

$$k_1 = (1+2(0))\sqrt{1} = 1$$
$$y(0.16667) = 1+1(0.16667) = 1.16667$$
$$k_2 = (1+2(0.16667))\sqrt{1.16667} = 1.44016$$

Corrector:

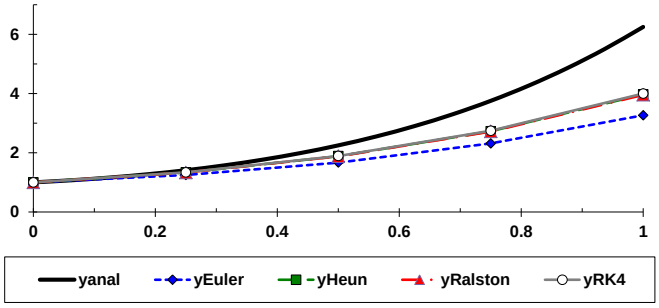
$$y(0.25) = 1+\ 0.25(1)+0.75(1.44016)\ 0.25 = 1.33253$$

The remaining steps can be implemented and summarized as

x	y	k_1	$x+(2/3)h$	$y+(2/3)k_1h$	k_2	dy/dx
0	1.00000	1.00000	0.16667	1.16667	1.44016	1.33012
0.25	1.33253	1.73153	0.41667	1.62112	2.33426	2.18358
0.5	1.87842	2.74111	0.66667	2.33528	3.56571	3.35956
0.75	2.71832	4.12183	0.91667	3.40529	5.22847	4.95181
1	3.95627					

(e) RK4

x	y	k_1	x_m	y_m	k_2	x_m	y_m	k_3	x_e	y_e	k_4	ϕ
0	1.0000	1	0.125	1.1250	1.32583	0.125	1.1657	1.34961	0.25	1.3374	1.73469	1.3476
0.25	1.3369	1.73436	0.375	1.5537	2.18133	0.375	1.6096	2.2202	0.5	1.8919	2.75096	2.2147
0.5	1.8906	2.74997	0.625	2.2343	3.36322	0.625	2.3110	3.42043	0.75	2.7457	4.14253	3.4100
0.75	2.7431	4.14056	0.875	3.2606	4.96574	0.875	3.3638	5.04368	1	4.0040	6.00299	5.0271
1	3.9998											



22.3 (a) Heun's method:

Predictor:

$$k_1 = -1 + 0^2 = -1$$

$$y(0.5) = 1 + (-1)0.5 = 0.5$$

$$k_2 = -0.5 + 0.5^2 = -0.25$$

Corrector:

$$y(0.5) = 1 + \frac{-1-0.25}{2}0.5 = 0.6875$$

The remaining steps can be implemented and summarized as

t	y	k_1	x_{i+1}	y_{i+1}	k_2	dy/dt
0	1	–1.0000	0.5	0.5	–0.2500	–0.625
0.5	0.6875	–0.4375	1	0.46875	0.5313	0.04688
1	0.71094	0.2891	1.5	0.855469	1.3945	0.8418
1.5	1.13184	1.1182	2	1.690918	2.3091	1.71362
2	1.98865	2.0114	2.5	2.994324	3.2557	2.63351
2.5	3.3054	2.9446	3	4.777702	4.2223	3.58345
3	5.09713	3.9029				

(b) As in Part **(a)**, the corrector can be represented as

$$y_{i+1}^1 = 1 + \frac{-1 + (-0.6875 + 0.5^2)}{2} 0.5 = 0.64063$$

The corrector can then be iterated to give

$$y_{i+1}^2 = 1 + \frac{-1 + (-0.64063 + 0.5^2)}{2} 0.5 = 0.65234$$
$$y_{i+1}^3 = 1 + \frac{-1 + (-0.65234 + 0.5^2)}{2} 0.5 = 0.64941$$

The iterations can be continued until the percent relative error falls below 0.1%. This occurs after 6 iterations with the result that $y(0.5) = 0.64996$ with $\varepsilon_a = 0.028\%$. The remaining values can be computed in a like fashion to give

x	*y*	iterations
0	1.0000000	0
0.5	0.6499634	6
1	0.6399316	6
1.5	1.0339067	6
2	1.8705671	5
2.5	3.1725717	5
3	4.9525967	4

(c) Midpoint method

$$k_1 = -(1) + (0)^2 = -1$$
$$y(0.25) = 1 + (-1)(0.25) = 0.75$$
$$k_2 = -(0.75) + 0.25^2 = -0.6875$$
$$y(0.5) = 1 + (-0.6875)0.5 = 0.65625$$

The remainder of the computations can be implemented in a similar fashion as listed below:

t	*y*	*dy/dt*	t_m	y_m	dy_m/dt
0	1	–1.0000	0.25	0.75	–0.6875
0.5	0.65625	–0.4063	0.75	0.554688	0.0078
1	0.66016	0.3398	1.25	0.745117	0.8174
1.5	1.06885	1.1812	1.75	1.364136	1.6984
2	1.91803	2.0820	2.25	2.438522	2.6240
2.5	3.23002	3.0200	2.75	3.985014	3.5775
3	5.01876	3.9812			

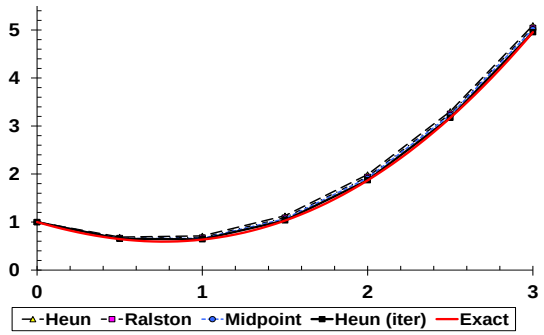
(d) Ralston's method:

$$k_1 = -(1) + (0)^2 = -1$$
$$y(0.3333) = 1 + (-1)(0.3333) = 0.6667$$
$$k_2 = -(0.6667) + .3333^2 = -0.5556$$
$$y(0.25) = 1 + \frac{-1 + 3(-0.5556)}{4} 0.5 = 0.6667$$

The remaining steps can be implemented and summarized as

t	y	k_1	$t + (2/3)h$	$y + (2/3)k_1 h$	k_2	dy/dt
0	1	–1.0000	0.3333	0.666667	–0.5556	–0.6667
0.5	0.66667	–0.4167	0.8333	0.527778	0.1667	0.02083
1	0.67708	0.3229	1.3333	0.784722	0.9931	0.82552
1.5	1.08984	1.1602	1.8333	1.476563	1.8845	1.70345
2	1.94157	2.0584	2.3333	2.627713	2.8167	2.62716
2.5	3.25515	2.9949	2.8333	4.253432	3.7743	3.57947
3	5.04488	3.9551				

All the versions can be plotted as:



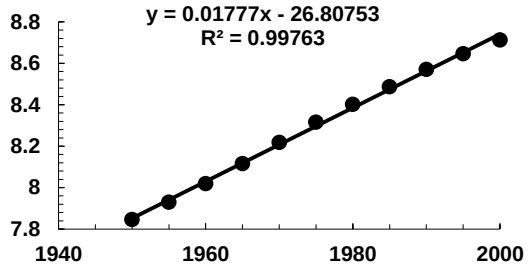
22.4 **(a)** The solution to the differential equation is

$$p = p_0 e^{k_g t}$$

Taking the natural log of this equation gives

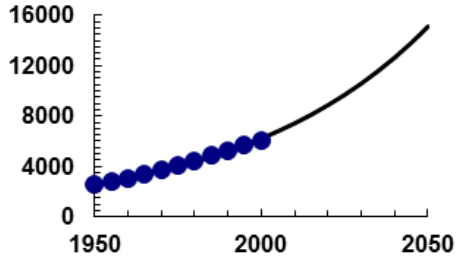
$$\ln p = \ln p_0 + k_g t$$

Therefore, a semi-log plot (ln p versus t) should yield a straight line with a slope of k_g. The plot, along with the linear regression best fit line is shown below. The estimate of the population growth rate is $k_g = 0.01777/\text{yr}$.



(b) The ODE can be integrated with the fourth-order RK method with the results tabulated and plotted below:

t	p	k_1	p_{mid}	k_2	p_{mid}	k_3	p_{end}	k_4	ϕ
1950	2555.0	45.413	2668.5	47.431	2673.6	47.521	2792.6	49.636	47.492
1955	2792.5	49.634	2916.5	51.839	2922.1	51.937	3052.1	54.250	51.906
1960	3052.0	54.247	3187.6	56.657	3193.6	56.764	3335.8	59.292	56.730
1965	3335.6	59.289	3483.9	61.923	3490.5	62.040	3645.8	64.802	62.003
1970	3645.7	64.799	3807.7	67.678	3814.9	67.806	3984.7	70.825	67.765
1975	3984.5	70.821	4161.5	73.968	4169.4	74.108	4355.0	77.407	74.063
1980	4354.8	77.403	4548.3	80.843	4556.9	80.996	4759.8	84.601	80.947
1985	4759.5	84.597	4971.0	88.356	4980.4	88.523	5202.2	92.464	88.470
1990	5201.9	92.460	5433.0	96.568	5443.3	96.751	5685.6	101.058	96.693
1995	5685.3	101.053	5938.0	105.543	5949.2	105.743	6214.1	110.450	105.679
2000	6213.7	110.445	6489.9	115.352	6502.1	115.570	6791.6	120.715	115.501
2005	6791.2	120.709	7093.0	126.073	7106.4	126.311	7422.8	131.935	126.236
2010	7422.4	131.928	7752.2	137.790	7766.9	138.051	8112.7	144.197	137.968
2015	8112.3	144.189	8472.7	150.597	8488.8	150.881	8866.7	157.598	150.791
2020	8866.2	157.590	9260.2	164.593	9277.7	164.904	9690.7	172.246	164.805
2025	9690.2	172.237	10120.8	179.890	10140.0	180.230	10591.4	188.254	180.122
2030	10590.9	188.244	11061.5	196.609	11082.4	196.981	11575.8	205.750	196.862
2035	11575.2	205.740	12089.5	214.882	12112.4	215.288	12651.6	224.873	215.159
2040	12651.0	224.861	13213.1	234.853	13238.1	235.297	13827.4	245.772	235.156
2045	13826.7	245.760	14441.1	256.680	14468.4	257.166	15112.6	268.614	257.011
2050	15111.8								



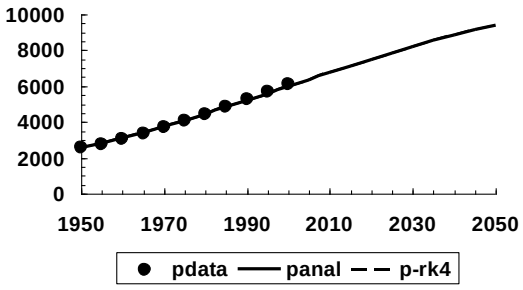
22.5 **(a)** The analytical solution can be used to compute values at times over the range. For example, the value at $t = 1955$ can be computed as

$$p = 2{,}555\frac{12{,}000}{2{,}555 + (12{,}000 - 2{,}555)e^{-0.026(1955-1950)}} = 2{,}826.2$$

Values at the other times can be computed and displayed along with the data in the plot below.

(b) The ODE can be integrated with the fourth-order RK method with the results tabulated and plotted below:

t	p-rk4	k_1	t_m	y_m	k_2	t_m	y_m	k_3	t_e	y_e	k_4	ϕ
1950	2555.0	52.29	1952.5	2685.7	54.20	1952.5	2690.5	54.27	1955.0	2826.3	56.18	54.23
1955	2826.2	56.17	1957.5	2966.6	58.06	1957.5	2971.3	58.13	1960.0	3116.8	59.99	58.09
1960	3116.6	59.99	1962.5	3266.6	61.81	1962.5	3271.1	61.87	1965.0	3425.9	63.64	61.83
1965	3425.8	63.64	1967.5	3584.9	65.36	1967.5	3589.2	65.41	1970.0	3752.8	67.06	65.37
1970	3752.6	67.06	1972.5	3920.3	68.63	1972.5	3924.2	68.66	1975.0	4096.0	70.15	68.63
1975	4095.8	70.14	1977.5	4271.2	71.52	1977.5	4274.6	71.55	1980.0	4453.5	72.82	71.52
1980	4453.4	72.82	1982.5	4635.4	73.97	1982.5	4638.3	73.98	1985.0	4823.3	75.00	73.95
1985	4823.1	75.00	1987.5	5010.6	75.88	1987.5	5012.8	75.89	1990.0	5202.6	76.62	75.86
1990	5202.4	76.62	1992.5	5394.0	77.20	1992.5	5395.5	77.21	1995.0	5588.5	77.63	77.18
1995	5588.3	77.63	1997.5	5782.4	77.90	1997.5	5783.1	77.90	2000.0	5977.8	78.00	77.87
2000	5977.7	78.00	2002.5	6172.7	77.94	2002.5	6172.5	77.94	2005.0	6367.4	77.71	77.91
2005	6367.2	77.71	2007.5	6561.5	77.32	2007.5	6560.5	77.32	2010.0	6753.8	76.77	77.29
2010	6753.7	76.77	2012.5	6945.6	76.06	2012.5	6943.9	76.07	2015.0	7134.0	75.21	76.04
2015	7133.9	75.21	2017.5	7321.9	74.21	2017.5	7319.4	74.23	2020.0	7505.0	73.09	74.20
2020	7504.9	73.09	2022.5	7687.6	71.83	2022.5	7684.5	71.85	2025.0	7864.2	70.47	71.82
2025	7864.0	70.47	2027.5	8040.2	68.98	2027.5	8036.5	69.01	2030.0	8209.1	67.43	68.98
2030	8208.9	67.43	2032.5	8377.5	65.75	2032.5	8373.3	65.80	2035.0	8537.9	64.04	65.76
2035	8537.7	64.05	2037.5	8697.8	62.23	2037.5	8693.3	62.28	2040.0	8849.1	60.41	62.25
2040	8849.0	60.41	2042.5	9000.0	58.50	2042.5	8995.2	58.56	2045.0	9141.8	56.61	58.53
2045	9141.6	56.62	2047.5	9283.1	54.65	2047.5	9278.2	54.72	2050.0	9415.2	52.73	54.68
2050	9415.0											



Thus, the RK4 results are so close to the analytical solution that the two results are indistinguishable graphically.

22.6 We can solve this problem with the M-file `Eulode` (Fig. 22.3). First, we develop a function to compute the derivatives

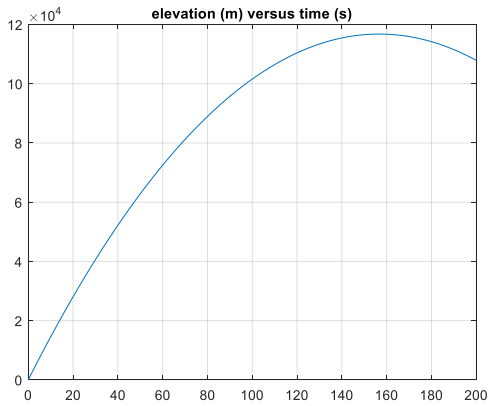
```
function dydt = projectile(t,y,radius)
% y(1) = elevation (m)
% y(2) = velocity (m/s)
global g
dydt = [y(2);-g*radius^2/(radius+y(1))^2];
```

The RK4sys function (Fig. 22.8) can then be used to generate results and display them graphically as invoked by the following script:

```
clear,clc,format compact
global g
h=0.0625;tspan=[0 200];y0=[0 1500];
g=9.81;radius=6.37e6;
[t y] = rk4sys(@projectile,tspan,y0,h,radius);
max_elevation_km=max(y(:,1))/1e3
plot(t,y(:,1))
title('elevation (m) versus time (s)'),grid
```

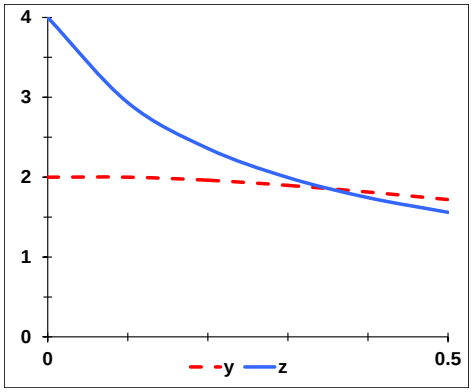
Results:

```
max_elevation_km =
  116.7813
```



22.7 **(a)** Euler's method:

t	y	z	dy/dt	dz/dt
0	2.0000	4.0000	0.00	–10.67
0.1	2.0000	2.9333	–0.38	–5.74
0.2	1.9619	2.3597	–0.65	–3.64
0.3	1.8970	1.9956	–0.83	–2.52
0.4	1.8140	1.7437	–0.95	–1.84



(b) 4th-order RK method:

$$k_{1,1} = f_1(0,2,4) = -2(2) + 4e^{-0} = 0$$

$$k_{1,2} = f_2(0,2,4) = -\frac{2(4)^2}{3} = -10.667$$

$$y(0.05) = 2 + 0(0.05) = 2$$

$$z(0.05) = 4 - 10.6667(0.05) = 3.47$$

$$k_{2,1} = f_1(0.05,2,3.47) = -2(2) + 4e^{-0.05} = -0.195$$
$$k_{2,2} = f_2(0.05,2.3,3.47) = -\frac{2(3.47)^2}{3} = -8.012$$

$$y(0.05) = 2 - 0.195(0.05) = 1.99$$
$$z(0.05) = 4 - 8.012(0.05) = 3.60$$

$$k_{3,1} = f_1(0.05,1.99,3.60) = -2(1.99) + 4e^{-0.05} = -0.176$$
$$k_{3,2} = f_2(0.05,1.99,3.60) = -\frac{1.99(3.6)^2}{3} = -8.595$$

$$y(0.1) = 2 - 0.176(0.1) = 1.98$$
$$z(0.1) = 4 - 8.595(0.1) = 3.14$$

$$k_{4,1} = f_1(0.1,1.98,3.14) = -2(1.98) + 4(3.14)e^{-0.1} = -0.346$$
$$k_{4,2} = f_2(0.1,1.98,3.14) = -\frac{1.98(3.14)^2}{3} = -6.517$$

The k's can then be used to compute the increment functions,

$$\phi_1 = \frac{0 + 2(-0.195 - 0.176) - 0.346}{6} = -0.181$$
$$\phi_2 = \frac{-10.667 + 2(-8.012 - 8.595) - 6.517}{6} = -8.4$$

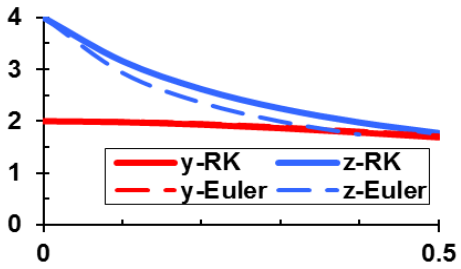
These slope estimates can then be used to make the prediction for the first step

$$y(0.1) = 2 - 0.181(0.1) = 1.982$$
$$z(0.1) = 4 - 8.4(0.1) = 3.160$$

The remaining steps can be taken in a similar fashion and the results summarized as

t	y	z	k_{11}	k_{12}	k_{21}	k_{22}	k_{31}	k_{32}	k_{41}	k_{42}	ϕ_1	ϕ_2
0	2.000	4.000	0.000	–10.667	–0.195	–8.012	–0.176	–8.595	–0.346	–6.517	–0.181	–8.400
0.1	1.982	3.160	–0.344	–6.597	–0.486	–5.246	–0.472	–5.479	–0.594	–4.400	–0.476	–5.408
0.2	1.934	2.619	–0.594	–4.423	–0.694	–3.651	–0.684	–3.760	–0.768	–3.130	–0.686	–3.729
0.3	1.866	2.246	–0.768	–3.138	–0.836	–2.659	–0.829	–2.715	–0.884	–2.317	–0.830	–2.701
0.4	1.783	1.976	–0.884	–2.321	–0.926	–2.005	–0.922	–2.037	–0.955	–1.770	–0.923	–2.029

A plot of these values along with the results for Euler's method can be developed.



22.8 The second-order van der Pol equation can be re-expressed as a system of 2 first-order ODEs,

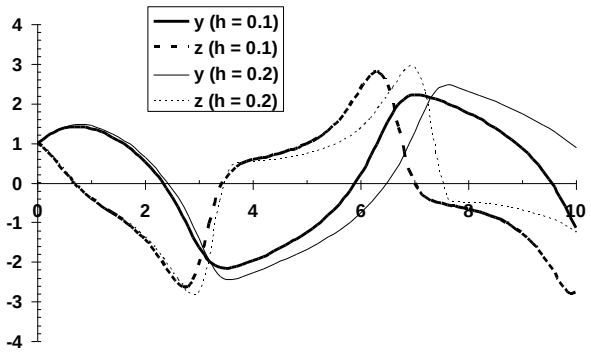
$$\frac{dy}{dt} = z$$
$$\frac{dz}{dt} = (1 - y^2)z - y$$

(a) Euler ($h = 0.2$). Here are the first few steps. The remainder of the computation would be implemented in a similar fashion and the results displayed in the plot below.

t	y $(h = 0.2)$	z $(h = 0.2)$	dy/dt	dz/dt
0	1	1	1	−1
0.2	1.2	0.8	0.8	−1.552
0.4	1.36	0.4896	0.4896	−1.77596
0.6	1.45792	0.1344072	0.134407	−1.6092
0.8	1.4848014	−0.187433	−0.18743	−1.25901

(b) Euler ($h = 0.1$). Here are the first few steps. The remainder of the computation would be implemented in a similar fashion and the results displayed in the plot below.

t	y $(h = 0.1)$	z $(h = 0.1)$	dy/dt	dz/dt
0	1	1	1	−1
0.1	1.1	0.9	0.9	−1.289
0.2	1.19	0.7711	0.7711	−1.51085
0.3	1.26711	0.6200145	0.620015	−1.64257
0.4	1.3291115	0.4557574	0.455757	−1.67847

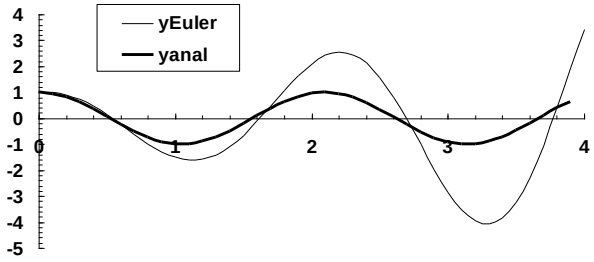


22.9 The second-order equation can be re-expressed as a system of two first-order ODEs,

$$\frac{dy}{dt} = z$$
$$\frac{dz}{dt} = -9y$$

(a) Euler. Here are the first few steps along with the analytical solution. The remainder of the computation would be implemented in a similar fashion and the results displayed in the plot below.

t	y_{Euler}	z_{Euler}	dy/dt	dz/dt	$y_{analytical}$
0	1	0	0	–9	1
0.1	1	–0.9	–0.9	–9	0.955336
0.2	0.91	–1.8	–1.8	–8.19	0.825336
0.3	0.73	–2.619	–2.619	–6.57	0.62161
0.4	0.4681	–3.276	–3.276	–4.2129	0.362358



(b) RK4. Here are the first few steps along with the analytical solution. The remainder of the computation would be implemented in a similar fashion and the results displayed in the plot below.

$$k_{1,1} = f_1(0,1,0) = z = 0$$
$$k_{1,2} = f_2(0,1,0) = -9y = -9(1) = -9$$

$$y(0.05) = 1 + 0(0.05) = 1$$
$$z(0.05) = 0 - 9(0.05) = -0.45$$

$$k_{2,1} = f_1(0.05, 1, -0.45) = -0.45$$
$$k_{2,2} = f_2(0.05, 1, -0.45) = -9(1) = -9$$

$$y(0.05) = 1 + (-0.45)(0.05) = 0.9775$$
$$z(0.05) = 0 - 9(0.05) = -0.45$$

$$k_{3,1} = f_1(0.05, 0.9775, -0.45) = -0.45$$
$$k_{3,2} = f_2(0.05, 0.9775, -0.45) = -9(0.9775) = -8.7975$$

$$y(0.1) = 1 + (-0.45)(0.1) = 0.9550$$
$$z(0.1) = 0 - 8.7975(0.1) = -0.8798$$

$$k_{4,1} = f_1(0.1, 0.9550, -0.8798) = -0.8798$$
$$k_{4,2} = f_2(0.1, 0.9550, -0.8798) = -9(0.9550) = -8.5950$$

The *k*'s can then be used to compute the increment functions,

$$\phi_1 = \frac{0 + 2(-0.45 - 0.45) - 0.8798}{6} = -0.4466$$
$$\phi_2 = \frac{-9 + 2(-9 - 8.7975) - 8.5950}{6} = -8.8650$$

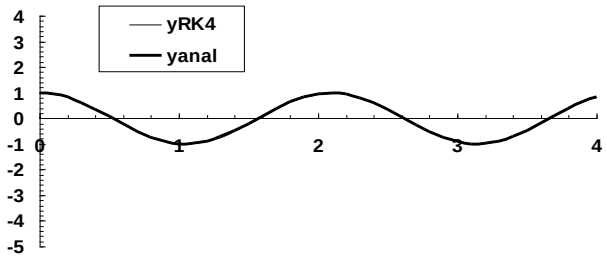
These slope estimates can then be used to make the prediction for the first step

$$y(0.1) = 1 - 0.4466(0.1) = 0.9553$$
$$z(0.1) = 0 - 8.8650(0.1) = -0.8865$$

The remaining steps can be taken in a similar fashion and the first few results summarized as

t	*y*	*z*	*y*-anal
0	1.0000	0.0000	1.00000
0.1	0.9553	–0.8865	0.95534
0.2	0.8253	–1.6938	0.82534
0.3	0.6216	–2.3498	0.62161
0.4	0.3624	–2.7960	0.36236
0.5	0.0708	–2.9924	0.07074

As can be seen, the results agree with the analytical solution closely. A plot of all the values can be developed and indicates the same close agreement.



22.10 A MATLAB M-file for Heun's method with iteration can be developed as

```
function [t,y] = Heun(dydt,tspan,y0,h,es,maxit)
% [t,y] = Heun(dydt,tspan,y0,h):
%   uses the midpoint method to integrate an ODE
% input:
%   dydt = name of the M-file that evaluates the ODE
%   tspan = [ti, tf] where ti and tf = initial and
%           final values of independent variable
%   y0 = initial value of dependent variable
%   h = step size
%   es = stopping criterion (%)
%        optional (default = 0.001)
%   maxit = maximum iterations of corrector
%        optional (default = 50)
%   es = (optional) stopping criterion (%)
%   maxit = (optional) maximum allowable iterations
% output:
%   t = vector of independent variable
%   y = vector of solution for dependent variable

% if necessary, assign default values
if nargin<6, maxit = 50; end  %if maxit blank set to 50
if nargin<5, es = 0.001; end  %if es blank set to 0.001
ti = tspan(1);
tf = tspan(2);
t = (ti:h:tf)';
n = length(t);
% if necessary, add an additional value of t
% so that range goes from t = ti to tf
if t(n)<tf
  t(n+1) = tf;
  n = n+1;
end
y = y0*ones(n,1); %preallocate y to improve efficiency
iter = 0;
for i = 1:n-1
  hh = t(i+1) - t(i);
  k1 = feval(dydt,t(i),y(i));
  y(i+1) = y(i) + k1*hh;
```

```
  while (1)
    yold = y(i+1);
    k2 = feval(dydt,t(i)+hh,y(i+1));
    y(i+1) = y(i) + (k1+k2)/2*hh;
    iter = iter + 1;
    if y(i+1) ~= 0, ea = abs((y(i+1) - yold)/y(i+1)) * 100; end
    if ea <= es | iter >= maxit, break, end
  end
end
plot(t,y)
```

Here is the test of the solution of Prob. 22.5. First, an M-file holding the differential equation is written as

```
function dp = dpdt(t, p)
dp = 0.026*(1-p/12000)*p;
```

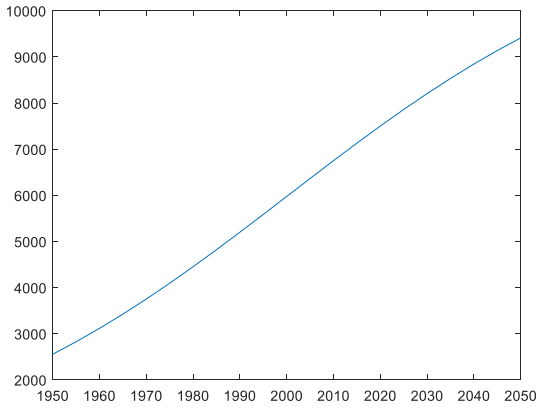
Then the M-file can be invoked as in

```
[t,p]=Heun(@dpdt,[1950 2050],2555,5,0.1);
disp([t,p])
```

Results:

```
  1.0e+03 *
    1.9500    2.5550
    1.9550    2.8261
    1.9600    3.1165
    1.9650    3.4256
    1.9700    3.7523
    1.9750    4.0953
    1.9800    4.4527
    1.9850    4.8222
    1.9900    5.2012
    1.9950    5.5868
    2.0000    5.9759
    2.0050    6.3651
    2.0100    6.7513
    2.0150    7.1313
    2.0200    7.5021
    2.0250    7.8609
    2.0300    8.2056
    2.0350    8.5345
    2.0400    8.8459
    2.0450    9.1386
    2.0500    9.4122
```

The following plot is generated



22.11 A MATLAB M-file for the midpoint method can be developed as

```
function [t,y] = midpoint(dydt,tspan,y0,h)
% [t,y] = midpoint(dydt,tspan,y0,h):
%   uses the midpoint method to integrate an ODE
% input:
%   dydt = name of the M-file that evaluates the ODE
%   tspan = [ti, tf] where ti and tf = initial and
%           final values of independent variable
%   y0 = initial value of dependent variable
%   h = step size
% output:
%   t = vector of independent variable
%   y = vector of solution for dependent variable

ti = tspan(1);
tf = tspan(2);
t = (ti:h:tf)';
n = length(t);
% if necessary, add an additional value of t
% so that range goes from t = ti to tf
if t(n)<tf
  t(n+1) = tf;
  n = n+1;
end
y = y0*ones(n,1); %preallocate y to improve efficiency
for i = 1:n-1
  hh = t(i+1) - t(i);
  k1 = feval(dydt,t(i),y(i));
  ymid = y(i) + k1*hh/2;
  k2 = feval(dydt,t(i)+hh/2,ymid);
  y(i+1) = y(i) + k2*hh;
end
plot(t,y)
```

Here is the test of the solution of Prob. 22.5. First, an M-file holding the differential equation is written as

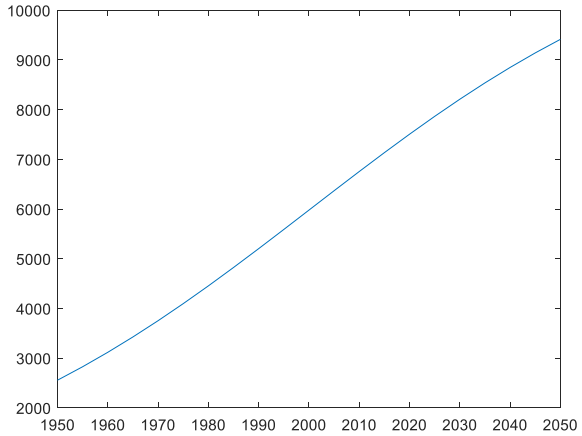
```
function dp = dpdt(t, p)
dp = 0.026*(1-p/12000)*p;
```

Then the M-file can be invoked as in

```
[t,p]=midpoint(@dpdt,[1950 2050],2555,5);
disp([t,p])
```

```
1.0e+03 *
    1.9500    2.5550
    1.9550    2.8260
    1.9600    3.1163
    1.9650    3.4253
    1.9700    3.7521
    1.9750    4.0953
    1.9800    4.4529
    1.9850    4.8227
    1.9900    5.2021
    1.9950    5.5881
    2.0000    5.9776
    2.0050    6.3672
    2.0100    6.7538
    2.0150    7.1341
    2.0200    7.5052
    2.0250    7.8643
    2.0300    8.2092
    2.0350    8.5380
    2.0400    8.8491
    2.0450    9.1416
    2.0500    9.4148
```

The following plot is generated



22.12 A MATLAB M-file for the fourth-order RK method can be developed as

```
function [t,y] = rk4(dydt,tspan,y0,h)
```

```
% [t,y] = rk4(dydt,tspan,y0,h):
%   uses the fourth-order Runge-Kutta method to integrate an ODE
% input:
%   dydt = name of the M-file that evaluates the ODE
%   tspan = [ti, tf] where ti and tf = initial and
%           final values of independent variable
%   y0 = initial value of dependent variable
%   h = step size
% output:
%   t = vector of independent variable
%   y = vector of solution for dependent variable

ti = tspan(1);
tf = tspan(2);
t = (ti:h:tf)';
n = length(t);
% if necessary, add an additional value of t
% so that range goes from t = ti to tf
if t(n)<tf
  t(n+1) = tf;
  n = n+1;
end
y = y0*ones(n,1); %preallocate y to improve efficiency
for i = 1:n-1
  hh = t(i+1) - t(i);
  k1 = feval(dydt,t(i),y(i));
  ymid = y(i) + k1*hh/2;
  k2 = feval(dydt,t(i)+hh/2,ymid);
  ymid = y(i) + k2*hh/2;
  k3 = feval(dydt,t(i)+hh/2,ymid);
  yend = y(i) + k3*hh;
  k4 = feval(dydt,t(i)+hh,yend);
  phi = (k1+2*(k2+k3)+k4)/6;
  y(i+1) = y(i) + phi*hh;
end
plot(t,y)
```

Here is the test of the solution of Prob. 22.2. First, an M-file holding the differential equation is written as

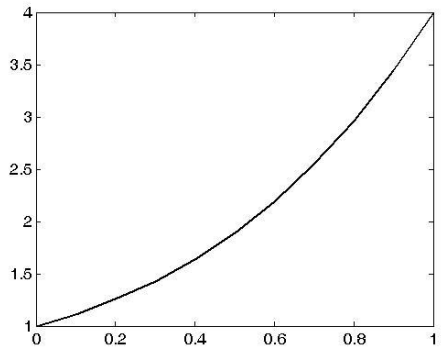
```
function dy = dydx(x, y)
dy = (1+2*x)*sqrt(y);
```

Then the M-file can be invoked as in

```
>> [x,y]=rk4(@dydx,[0 1],1,0.1);
>> disp([x,y])
         0    1.0000
    0.1000    1.1130
    0.2000    1.2544
    0.3000    1.4280
    0.4000    1.6384
    0.5000    1.8906
    0.6000    2.1904
    0.7000    2.5440
```

```
    0.8000    2.9584
    0.9000    3.4410
    1.0000    4.0000
```

The following plot is generated



22.13 The following M-file can be used for any number of simultaneous first-order ODEs:

```
function [tp,yp] = eulersystem(dydt,tspan,y0,h,varargin)
% Euler ode
% [t,y] = eulersystem(dydt,tspan,y0,h,p1,p2,...): integrates
% a system of ODEs with Euler's method
% input:
% dydt = name of the M-file that evaluates the ODEs
% tspan = [ti, tf]; initial and final times with output
% generated at interval of h, or
% = [t0 t1 ... tf]; specific times where solution output
% y0 = initial values of dependent variables
% h = step size
% p1,p2,... = additional parameters used by dydt
% output:
% tp = vector of independent variable
% yp = vector of solution for dependent variables
if nargin < 4,error('at least 4 input arguments required'), end
if any(diff(tspan)<= 0),error('tspan not ascending order'), end
n = length(tspan);
ti = tspan(1);tf = tspan(n);
if n == 2
  t = (ti:h:tf)'; n = length(t);
  if t(n)<tf
    t(n + 1) = tf;
    n = n + 1;
  end
else
  t = tspan;
end
tt = ti; y(1,:) = y0;
np = 1; tp(np) = tt; yp(np,:) = y(1,:);
i = 1;
while(1)
```

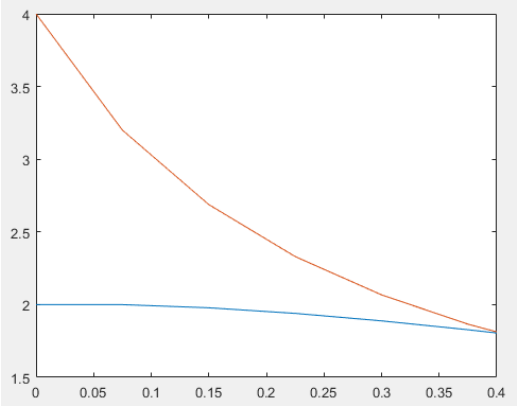
```
tend = t(np + 1);
hh = t(np + 1) - t(np);
if hh > h,hh = h;end
while(1)
  if tt+hh > tend,hh = tend-tt;end
  phi = dydt(tt,y(i,:),varargin{:})';
  y(i + 1,:) = y(i,:) + phi*hh;
  tt = tt + hh;
  i = i + 1;
  if tt >= tend,break,end
end
np = np + 1; tp(np) = tt; yp(np,:) = y(i,:);
if tt >= tf,break,end
end
```

Here is the test of the solution of Prob. 22.7. An M-file holding the differential equations and a script can be written as

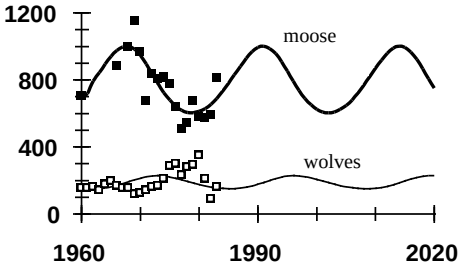
```
function dy = dydtsystem(t, y)
dy = [-2*y(1)+4*exp(-t);-y(1)*y(2)^2/3];
```

```
clear, clc, format compact
[t,y]=eulersystem(@dydtsystem,[0 0.4],[2 4],0.075);
z=[t' y];
disp(z)
plot(t,y)
```

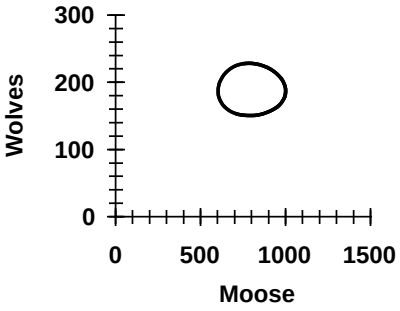
```
         0    2.0000    4.0000
    0.0750    2.0000    3.2000
    0.1500    1.9783    2.6880
    0.2250    1.9398    2.3306
    0.3000    1.8884    2.0672
    0.3750    1.8274    1.8655
    0.4000    1.8047    1.8125
```



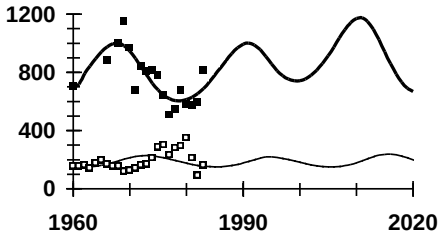
22.14 (a) Non-linear regression can be used to minimize the sum of the squares of the residuals between the data and the simulation. The resulting estimates are: $a = 0.32823$, $b = 0.01231$, $c = 0.22445$, and $d = 0.00029$. The fit is:

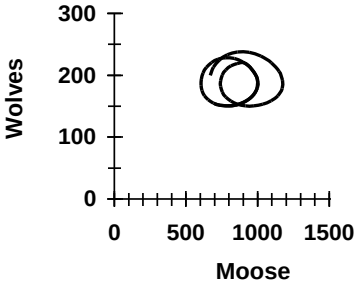


(b) The results in state space are,

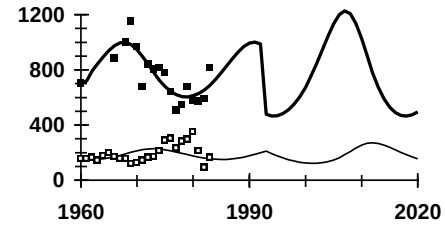


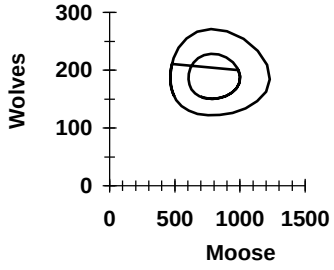
(c)





(d)





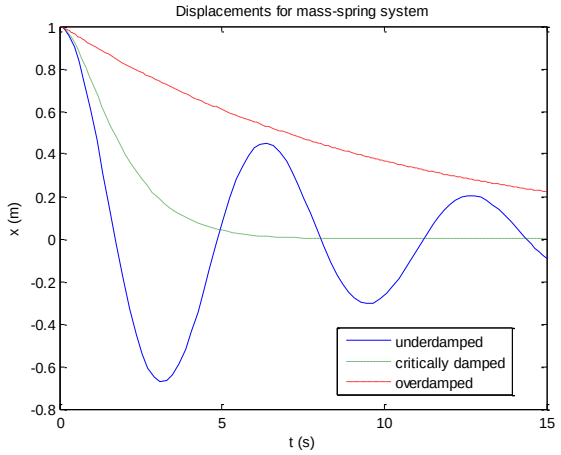
22.15 Function defining the derivative:

```
function dy = dydt(t,y,m,k,c)
dy=[y(2);-(c*y(2)+k*y(1))/m];
end
```

Script to generate plot:

```
k=20;m=20;
tspan=[0,15];y0=[1 0];
[t1,y1]=ode45(@dydt,tspan,y0,[],m,k,5);
[t2,y2]=ode45(@dydt,tspan,y0,[],m,k,40);
[t3,y3]=ode45(@dydt,tspan,y0,[],m,k,200);
plot(t1,y1(:,1),t2,y2(:,1),':',t3,y3(:,1),'--')
legend('underdamped','critically
damped','overdamped','location','best')
title('Displacements for mass-spring system')
xlabel('t (s)'),ylabel('x (m)')
```

Output:



22.16 The volume of the tank can be computed as

$$\frac{dV}{dt} = -CA\sqrt{2gH} \tag{1}$$

This equation cannot be solved because it has 2 unknowns: V and H. The volume is related to the depth of liquid by

$$V = \frac{\pi H^2(3r - H)}{3} \tag{2}$$

Equation (2) can be differentiated to give

$$\frac{dV}{dt} = (2\pi rH - \pi H^2)\frac{dH}{dt} \tag{3}$$

This result can be substituted into Eq. (1) to give an equation with 1 unknown,

$$\frac{dH}{dt} = -\frac{CA\sqrt{2gH}}{2\pi rH - \pi H^2} \tag{4}$$

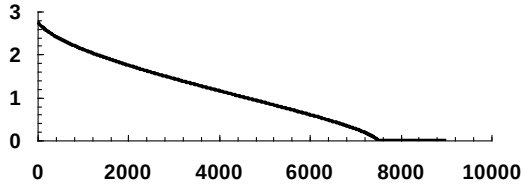
The area of the orifice can be computed as

$$A = \pi(0.015)^2 = 0.000707$$

Substituting this value along with the other parameters ($C = 0.55$, $g = 9.81$, $r = 1.5$) into Eq. (4) gives

$$\frac{dH}{dt} = -0.000548144\frac{\sqrt{H}}{3H - H^2} \tag{5}$$

We can solve this equation with an initial condition of $H = 2.75$ m using the 4th-order RK method with a step size of 6 s. If this is done, the result can be plotted as shown,



The results indicate that the tank empties at between $t = 7482$ and 7488 seconds.

22.17 (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

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

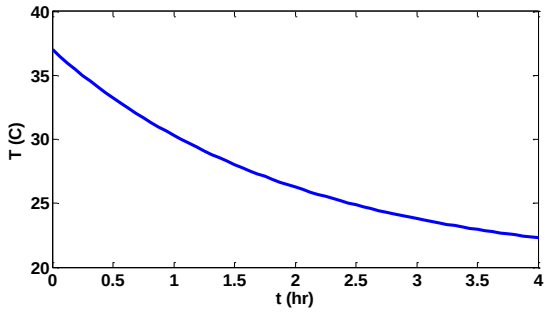
Thus, the person died 1.166 hrs prior to being discovered.

(b) The following function can be developed to hold the ODE:

```
function dy = dTdt(t,T,Ta,K)
dy=-K*(T-Ta);
end
```

The following script then uses the `rk4sys` function (Fig. 22.8) to generate the solution and the plot. For convenience, we have redefined the time of death as $t = 0$.

```
clear,clc,clf
[t,y]=rk4sys(@dTdt,[0 4],37,1/16,20,0.499264);
plot(t,y)
xlabel('t (hr)'),ylabel('T (C)')
```

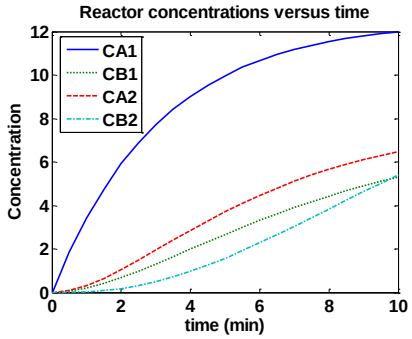


22.18 Function defining the derivatives:

```
function dc = dCdtReactor(t,C,tau,CA0,k)
dc=[1/tau*(CA0-C(1))-k*C(1); -1/tau*C(2)+k*C(1); ...
    1/tau*(C(1)-C(3))-k*C(3); 1/tau*(C(2)-C(4))+k*C(3)];
```

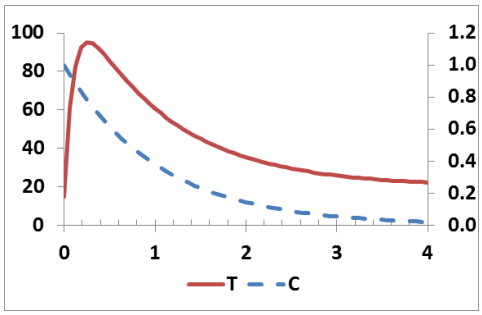
Script to generate plot using the `rk4sys` function (Fig. 22.8):

```
clear,clc,clf
CA0=20;k=0.12;tau=5;
tspan=[0:1/2:10];y0=[0 0 0 0];
[t,C]=rk4sys(@dCdtReactor,tspan,y0,1/16,tau,CA0,k);
plot(t,C(:,1),t,C(:,2),':',t,C(:,3),'--',t,C(:,4),'-.')
legend('CA1','CB1','CA2','CB2','location','best')
title('Reactor concentrations versus time')
xlabel('time (min)'),ylabel('Concentration')
```



22.19 The classical 4th order RK method yields

t	*C*	*T*
0	1.0000	15.0000
0.0625	0.9413	60.7968
0.125	0.8858	82.9030
0.1875	0.8336	92.3765
0.25	0.7844	95.1878
0.3125	0.7381	94.5486
0.375	0.6945	92.1809
0.4375	0.6536	89.0041
0.5	0.6150	85.5057
0.5625	0.5788	81.9414
0.625	0.5446	78.4421
0.6875	0.5125	75.0722
0.75	0.4823	71.8606
0.8125	0.4539	68.8176
0.875	0.4272	65.9436
0.9375	0.4021	63.2342
1	0.3784	60.6825
.		
.		
.		



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

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

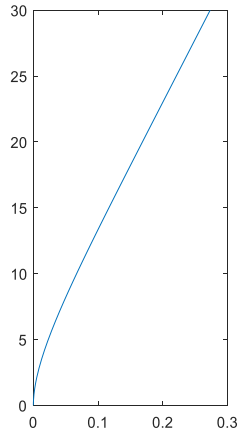
Scripts & function:

```
clear, clc
format compact
[z,y]=ode45(@dySail,[0:1:30],[0 0]);
```

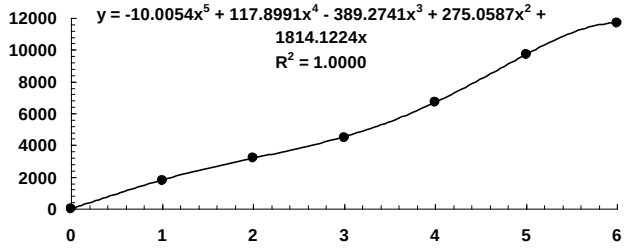
```
[z y]
plot(y(:,1),z)

function dy = dySail(z, y)
f=200/(5+z)*exp(-2*z/30);
L = 30; E = 1.25e8; I = 0.05;
dy = [y(2);f/(2*E*I)*(L-z)^2];

ans =
         0         0         0
    1.0000    0.0013    0.0025
    2.0000    0.0047    0.0043
    3.0000    0.0097    0.0057
    4.0000    0.0159    0.0067
    5.0000    0.0230    0.0075
    6.0000    0.0309    0.0081
    7.0000    0.0393    0.0086
    8.0000    0.0481    0.0090
    9.0000    0.0573    0.0094
   10.0000    0.0668    0.0096
   11.0000    0.0765    0.0098
   12.0000    0.0864    0.0100
   13.0000    0.0964    0.0101
   14.0000    0.1065    0.0102
   15.0000    0.1167    0.0102
   16.0000    0.1270    0.0103
   17.0000    0.1373    0.0103
   18.0000    0.1477    0.0104
   19.0000    0.1581    0.0104
   20.0000    0.1685    0.0104
   21.0000    0.1790    0.0104
   22.0000    0.1894    0.0105
   23.0000    0.1999    0.0105
   24.0000    0.2103    0.0105
   25.0000    0.2208    0.0105
   26.0000    0.2313    0.0105
   27.0000    0.2417    0.0105
   28.0000    0.2522    0.0105
   29.0000    0.2627    0.0105
   30.0000    0.2732    0.0105
```



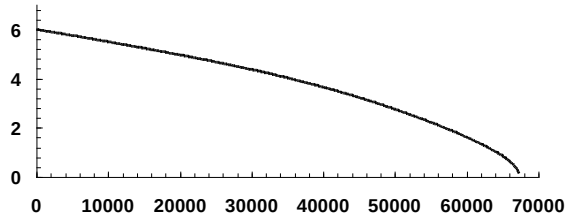
22.21 This problem can be approached in a number of ways. The simplest way is to fit the area-depth data with polynomial regression. A fifth-order polynomial with a zero intercept yields a perfect fit:



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.



22.22 Function defining the derivatives:

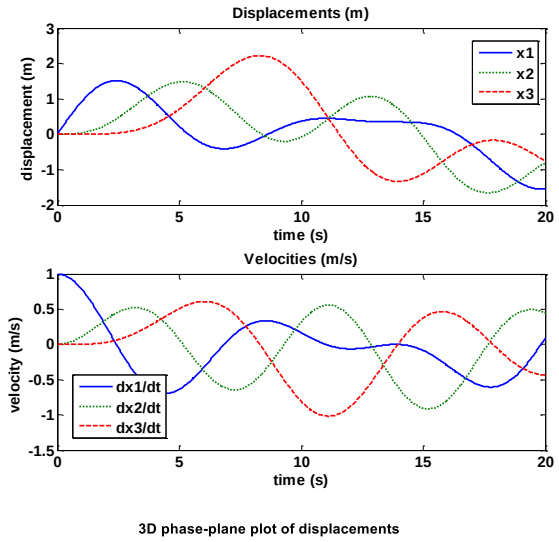
```
function dy = ODEquake(t,y,m1,m2,m3,k1,k2,k3)
dy=[y(4);y(5);y(6);-k1/m1*y(1)+k2/m1*(y(2)-y(1)); ...
    k2/m2*(y(1)-y(2))+k3/m2*(y(3)-y(2)); ...
    k3/m3*(y(2)-y(3))];
```

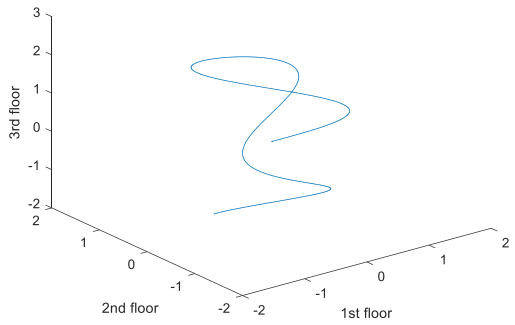
Script to generate plot using the `rk4sys` function (Fig. 22.8):

```
clear,clc,clf
m1=12000;m2=10000;m3=8000;
k1=3000;k2=2400;k3=1800;
tspan=[0:1/32:20];y0=[0 0 0 1 0 0];
[t,y]=rk4sys(@ODEquake,tspan,y0,1/32,m1,m2,m3,k1,k2,k3);
subplot(2,1,1)
plot(t,y(:,1),t,y(:,2),':',t,y(:,3),'--')
```

```
legend('x1','x2','x3','location','best')
title('Displacements (m)')
xlabel('time (s)'),ylabel('displacement (m)')
subplot(2,1,2)
plot(t,y(:,4),t,y(:,5),':',t,y(:,6),'--')
legend('dx1/dt','dx2/dt','dx3/dt','location','best')
title('Velocities (m/s)')
xlabel('time (s)'),ylabel('velocity (m/s)')
figure
plot3(y(:,1),y(:,2),y(:,3))
title('3D phase-plane plot of displacements')
xlabel('1st floor'),ylabel('2nd floor') ,zlabel('3rd floor')
```

Output:

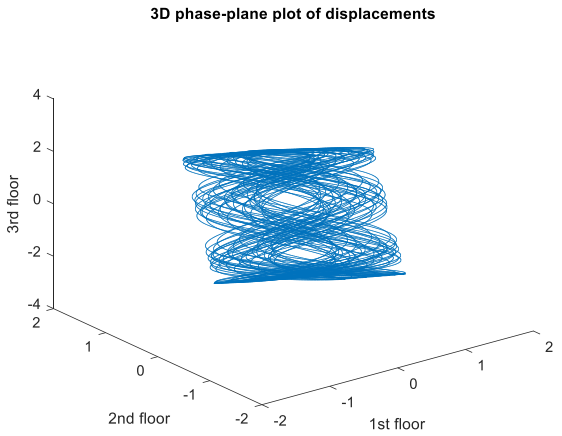




Note that a nicer phase plane plot results if the time of the simulation is extended by modifying the end of the script to

```
figure
tspan=[0:1/32:1000];y0=[0 0 0 1 0 0];
[t,y]=rk4sys(@ODEquake,tspan,y0,1/32,m1,m2,m3,k1,k2,k3);
plot3(y(:,1),y(:,2),y(:,3))
```

```
title('3D phase-plane plot of displacements')
xlabel('1st floor'),ylabel('2nd floor') ,zlabel('3rd floor')
```



22.23 Here is a function to implement the midpoint method:

```
function [tp,yp] = midpoint(dydt,tspan,y0,h,varargin)
% [t,y] = midpoint(dydt,tspan,y0,h):
%   uses the midpoint method to integrate an ODE
% input:
%   dydt = name of the M-file that evaluates the ODE
%   tspan = [ti, tf]; initial and final times with output
%          generated at interval of h, or
%         = [t0 t1 ... tf]; specific times where solution output
%   y0 = initial values of dependent variables
%   h = step size
%   p1,p2,... = additional parameters used by dydt
% output:
%   tp = vector of independent variable
%   yp = vector of solution for dependent variables

if nargin<4,error('at least 4 input arguments required'), end
if any(diff(tspan)<=0),error('tspan not ascending order'), end
n = length(tspan);
ti = tspan(1);tf = tspan(n);
if n == 2
  t = (ti:h:tf)'; n = length(t);
  if t(n)<tf
    t(n+1) = tf;
    n = n+1;
  end
else
  t = tspan;
end
tt = ti; y(1,:) = y0;
np = 1; tp(np) = tt; yp(np,:) = y(1,:);
i=1;
while(1)
  tend = t(np+1);
  hh = t(np+1) - t(np);
```

```
  if hh>h,hh = h;end
  while(1)
    if tt+hh>tend,hh = tend-tt;end
    k1 = dydt(tt,y(i,:),varargin{:})';
    ymid = y(i,:) + k1*hh/2;
    k2 = dydt(tt+hh/2,ymid,varargin{:})';
    y(i+1,:) = y(i,:) + k2*hh;
    tt = tt+hh;
    i=i+1;
    if tt>=tend,break,end
  end
  np = np+1; tp(np) = tt; yp(np,:) = y(i,:);
  if tt>=tf,break,end
end
```

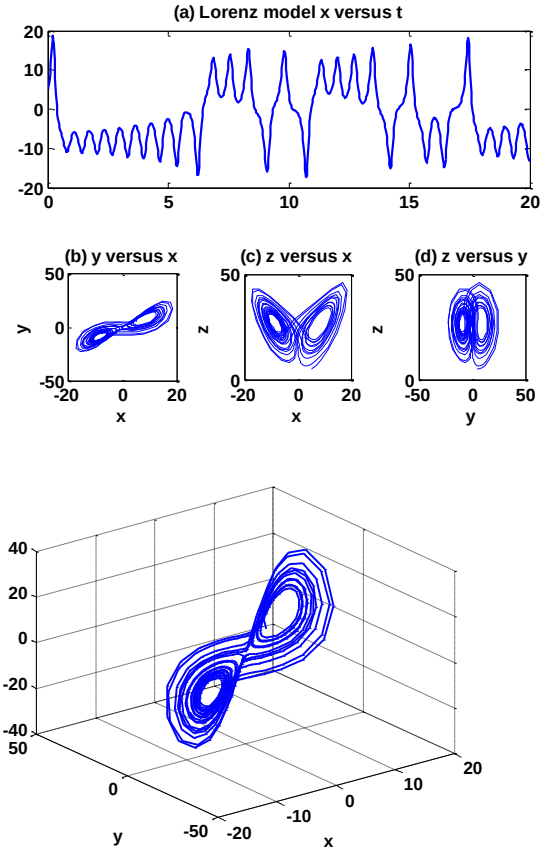
The following function holds the Lorenz ODEs:

```
function yp=lorenz(t,y,sigma,b,r)
yp=[-sigma*y(1)+sigma*y(2);r*y(1)-y(2)-y(1)*y(3);-
b*y(3)+y(1)*y(2)];
```

The following script solves the ODEs and generates the plots:

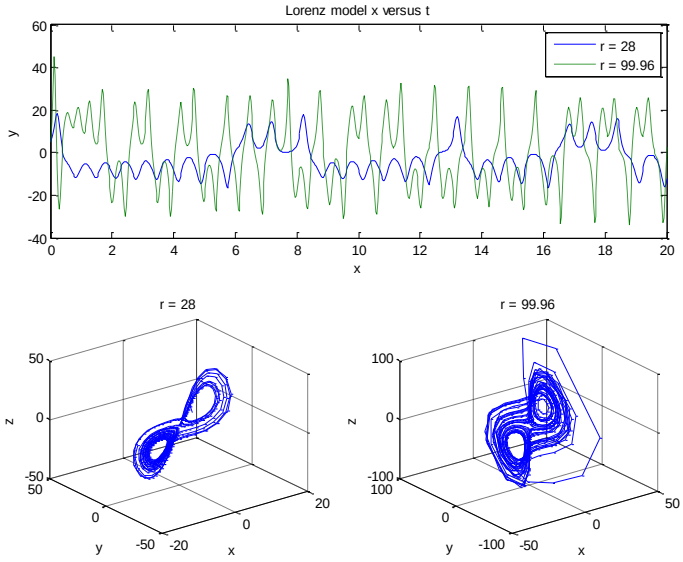
```
clear,clc,clf
tspan=[0 20];y0=[5 5 5];
sigma=10;b=8/3;r=28;
[t y] = midpoint(@lorenz,tspan,y0,0.03125,sigma,b,r);
subplot(2,3,[1 2 3])
plot(t,y(:,1))
title('(a) Lorenz model x versus t');
subplot(2,3,4);plot(y(:,1),y(:,2))
xlabel('x');ylabel('y')
axis square;title('(b) y versus x')
subplot(2,3,5);plot(y(:,1),y(:,3))
xlabel('x');ylabel('z')
axis square;title('(c) z versus x')
subplot(2,3,6);plot(y(:,2),y(:,3))
xlabel('y');ylabel('z')
axis square;title('(d) z versus y')
pause
subplot(1,1,1)
plot3(y(:,1),y(:,2),y(:,2))
xlabel('x');ylabel('y');zlabel('z');grid
```

Here are the results of running the script:



22.24 Script:

```
clear,clc,clf
tspan=[0 20];y0=[5 5 5];
sigma=10;b=8/3;
[t y] = rk4sys(@lorenz,tspan,y0,0.03125,sigma,b,28);
[t1 y1] = rk4sys(@lorenz,tspan,y0,0.03125,sigma,b,99.96);
subplot(2,2,[1 2])
plot(t,y(:,1),t1,y1(:,1),'--')
title('Lorenz model x versus t');
xlabel('x');ylabel('y')
legend('r = 28','r = 99.96')
subplot(2,2,3)
plot3(y(:,1),y(:,2),y(:,2))
title('r = 28');
xlabel('x');ylabel('y');zlabel('z');grid
subplot(2,2,4)
plot3(y1(:,1),y1(:,2),y1(:,2))
title('r = 99.96');
xlabel('x');ylabel('y');zlabel('z');grid
```



22.25

```
clear,clc,clf
tauw=[20 10 5];
kgmax=0.2; Ks=150; kd=0.01; kr=0.01;
Y=0.5; Sin=1000;
for i = 1:length(tauw)
  [t,c]=ode45(@derivs,[0 150],[5
0],[],tauw(i),kgmax,Ks,kd,kr,Y,Sin);
  cT=c(:,1)+c(:,2);
  subplot(3,1,i)
  plot(t,c,t,cT)
  legend('X','S','cT','location','best')
end

function dy=derivs(t,y,tauw,kgmax,Ks,kd,kr,Y,Sin)
dy=[(kgmax*y(2)./(Ks+y(2))-kd-kr-1/tauw).*y(1);...
    -1/Y*kgmax*y(2)./(Ks+y(2)).*y(1)+kd*y(1)+1/tauw*(Sin-y(2))];
End
```