CHAPTER 22

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

$$\int \frac{dy}{y} = \int t^3 - 1.5\, dt$$

The integrals can be evaluated to give,

$$\ln y = \frac{t^4}{4} - 1.5t + C$$

Substituting the initial conditions yields $C = 0$. Substituting this value and taking the exponential gives

$$y = e^{\frac{t^4}{4} - 1.5t}$$

t	*y*
0	**1**
0.25	**0.687961**
0.5	**0.479805**
0.75	**0.351376**
1	**0.286505**
1.25	**0.282339**
1.5	**0.373673**
1.75	**0.755577**
2	**2.718282**

(b) Euler method (h = 0.5):

t	*y*	*dy*/*dt*
0	**1**	-1.5
0.5	**0.25**	-0.34375
1	**0.078125**	-0.03906
1.5	**0.058594**	0.109863
2	**0.113525**	

Euler method (h = 0.25):

t	*y*	*dy*/*dt*
0	**1**	-1.5
0.25	**0.625**	-0.92773
0.5	**0.393066**	-0.54047
0.75	**0.25795**	-0.2781
1	**0.188424**	-0.09421
1.25	**0.164871**	0.074707
1.5	**0.183548**	0.344153
1.75	**0.269586**	1.040434
2	**0.529695**	

(c) Midpoint method (h = 0.5)

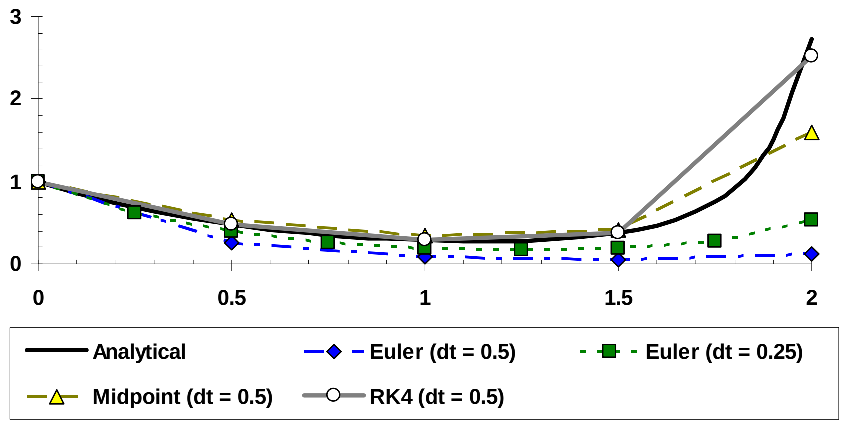
t	*y*	*dy*/*dt*	t_m	y_m	dy_m/dt
0	**1**	-1.5	0.25	0.625	-0.92773

0.5	**0.536133**	-0.73718	0.75	0.351837	-0.37932
1	**0.346471**	-0.17324	1.25	0.303162	0.13737
1.5	**0.415156**	0.778417	1.75	0.60976	2.353292
2	**1.591802**				

(d) RK4 (h = 0.5)

t	y	k_1	t_m	y_m	k_2	t_m	y_m	k_3	t_e	y_e	k_4	ϕ
0	**1.0000**	-1.5000	0.25	0.6250	-0.9277	0.25	0.7681	-1.1401	0.5	0.4300	-0.5912	-1.0378
0.5	**0.4811**	-0.6615	0.75	0.3157	-0.3404	0.75	0.3960	-0.4269	1	0.2676	-0.1338	-0.3883
1	**0.2869**	-0.1435	1.25	0.2511	0.1138	1.25	0.3154	0.1429	1.5	0.3584	0.6720	0.1736
1.5	**0.3738**	0.7008	1.75	0.5489	2.1186	1.75	0.9034	3.4866	2	2.1170	13.7607	4.2786
2	**2.5131**											

All the solutions can be presented graphically as



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

$$\int \frac{dy}{\sqrt{y}} = \int 1 + 4x \, dx$$

The integrals can be evaluated to give,

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

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

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

Some selected value can be computed as

x	y
0	**1**
0.25	**1.336914**
0.5	**1.890625**
0.75	**2.743164**
1	**4**

(b) Euler's method:

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

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

The remaining steps can be implemented and summarized as

x	*y*	*dy/dx*
0	**1**	1
0.25	**1.25**	1.67705
0.5	**1.66926**	2.584
0.75	**2.31526**	3.804
1	**3.26626**	5.42184

(c) Heun's method:

Predictor:

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

Corrector:

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

The remaining steps can be implemented and summarized as

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

(d) Ralston's method:

Predictor:

$$k_1 = (1+2(0))\sqrt{1} = 1$$
$$y(0.1875) = 1 + 1(0.1875) = 1.1875$$
$$k_2 = (1+2(0.1875))\sqrt{1.1875} = 1.49837$$

Corrector:

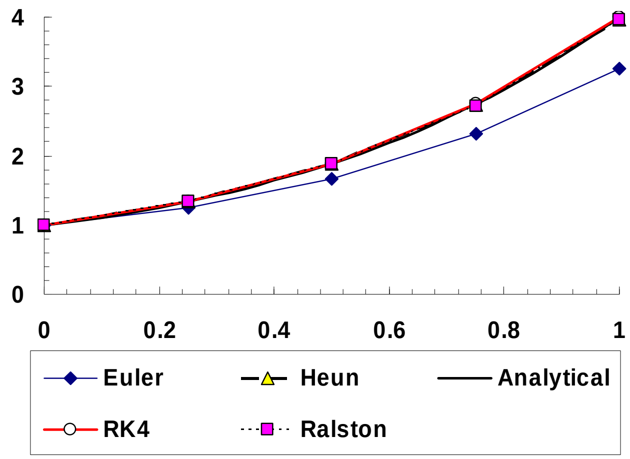
$$y(0.25) = 1 + \frac{1 + 2(1.49837)}{3} 0.25 = 1.33306$$

The remaining steps can be implemented and summarized as

x	y	k_1	$x + 3/4h$	$y + (3/4)k_1h$	k_2	dy/dx
0	**1**	1	0.1875	1.1875	1.49837	1.3322
0.25	**1.33306**	1.73187	0.4375	1.65779	2.41416	2.1867
0.5	**1.87974**	2.74208	0.6875	2.39388	3.67464	3.3638
0.75	**2.72069**	4.12363	0.9375	3.49387	5.37392	4.9572
1	**3.95998**					

(e) RK4

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



22.3 (a) <u>Heun's method:</u>

Predictor:

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

Corrector:

$$y(0.5) = 1 + \frac{-2 + 0.25}{2} 0.5 = 0.5625$$

The remaining steps can be implemented and summarized as

t	y	k_1	x_{i+1}	y_{i+1}	k_2	dy/dt
0	**1**	-2.0000	0.5	0	0.2500	-0.875
0.5	**0.5625**	-0.8750	1	0.125	0.7500	-0.0625
1	**0.53125**	-0.0625	1.5	0.5	1.2500	0.59375
1.5	**0.82813**	0.5938	2	1.125	1.7500	1.17188
2	**1.41406**	1.1719				

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

$$y_{i+1}^1 = 1 + \frac{-2 + (-2(0) + 0.5^2)}{2} 0.5 = 0.5625$$

The corrector can then be iterated to give

$$y_{i+1}^2 = 1 + \frac{-2 + (-2(0.5625) + 0.5^2)}{2} 0.5 = 0.28125$$

$$y_{i+1}^3 = 1 + \frac{-2 + (-2(0.28125) + 0.5^2)}{2} 0.5 = 0.421875$$

The iterations can be continued until the percent relative error falls below 0.1%. This occurs after 12 iterations with the result that $y(0.5) = 0.37491$ with $\varepsilon_a = 0.073\%$. The remaining values can be computed in a like fashion to give

t	y
0	**1.0000000**
0.5	**0.3749084**
1	**0.3334045**
1.5	**0.6526523**
2	**1.2594796**

(c) Midpoint method

$$k_1 = -2(1) + (0)^2 = -2$$
$$y(0.25) = 1 + (-2)(0.25) = 0.5$$
$$k_2 = -2(0.5) + 0.25^2 = -0.9375$$
$$y(0.5) = 1 + (-0.9375)0.5 = 0.53125$$

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

t	y	dy/dt	t_m	y_m	dy_m/dt
0	**1**	-2.0000	0.25	0.5	-0.9375
0.5	**0.53125**	-0.8125	0.75	0.328125	-0.0938
1	**0.48438**	0.0313	1.25	0.492188	0.57813
1.5	**0.77344**	0.7031	1.75	0.949219	1.16406
2	**1.35547**				

(d) Ralston's method:

$$k_1 = -2(1) + (0)^2 = -2$$
$$y(0.375) = 1 + (-2)(0.375) = 0.25$$

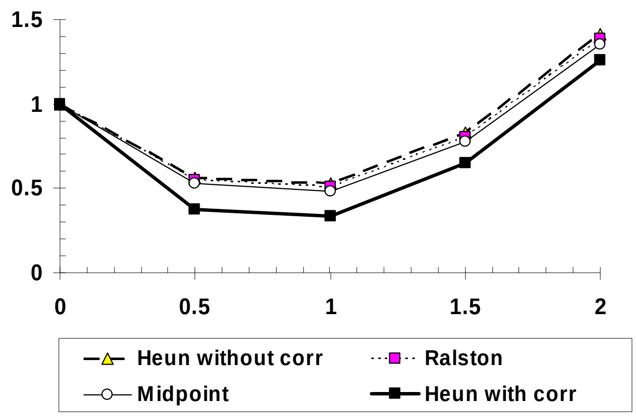
$k_2 = -2(0.25) + 0.375^2 = -0.3594$

$$y(0.25) = 1 + \frac{-2 + 2(-0.3594)}{3} 0.5 = 0.54688$$

The remaining steps can be implemented and summarized as

t	*y*	k_1	$t + 3/4h$	$y + (3/4)k_1h$	k_2	*dy/dt*
0	**1**	-2.0000	0.375	0.25	-0.3594	-0.9063
0.5	**0.54688**	-0.8438	0.875	0.230469	0.3047	-0.0781
1	**0.50781**	-0.0156	1.375	0.501953	0.8867	0.58594
1.5	**0.80078**	0.6484	1.875	1.043945	1.4277	1.16797
2	**1.38477**					

All the versions can be plotted as:



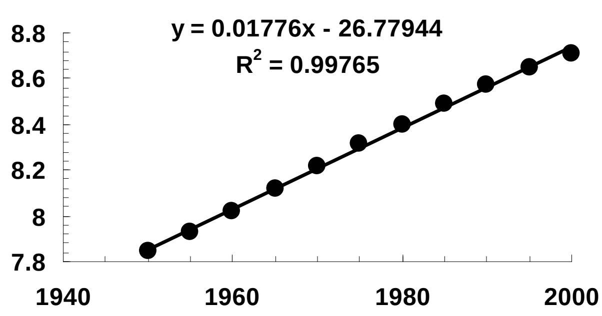
22.4 **(a)** The solution to the differential equation is

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

Taking the natural log of this equation gives

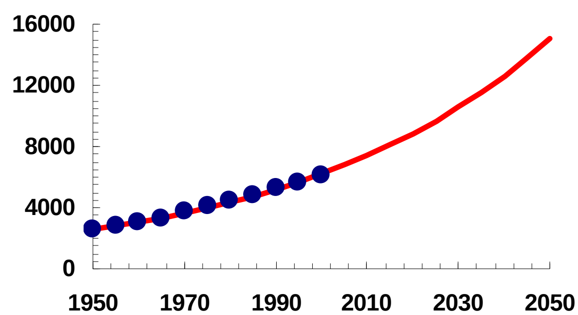
$$\ln p = \ln p_0 + k_g t$$

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 = 0.01776/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	2560.0	45.466	2673.7	47.485	2678.7	47.575	2797.9	49.691	47.546
1955	2797.7	49.688	2922.0	51.895	2927.5	51.993	3057.7	54.305	51.961
1960	3057.5	54.303	3193.3	56.714	3199.3	56.821	3341.6	59.348	56.787
1965	3341.5	59.345	3489.8	61.980	3496.4	62.097	3652.0	64.860	62.060
1970	3651.8	64.856	3813.9	67.736	3821.1	67.864	3991.1	70.883	67.823
1975	3990.9	70.879	4168.1	74.026	4176.0	74.166	4361.7	77.465	74.121
1980	4361.5	77.461	4555.1	80.901	4563.7	81.053	4766.8	84.659	81.005
1985	4766.5	84.655	4978.2	88.413	4987.5	88.580	5209.4	92.521	88.527
1990	5209.2	92.516	5440.4	96.624	5450.7	96.806	5693.2	101.112	96.748
1995	5692.9	101.107	5945.7	105.596	5956.9	105.796	6221.9	110.502	105.732
2000	6221.6	110.496	6497.8	115.402	6510.1	115.620	6799.7	120.764	115.551
2005	6799.3	120.757	7101.2	126.119	7114.6	126.357	7431.1	131.978	126.281
2010	7430.7	131.971	7760.6	137.831	7775.3	138.091	8121.2	144.234	138.008
2015	8120.8	144.227	8481.3	150.630	8497.3	150.915	8875.3	157.628	150.824
2020	8874.9	157.620	9268.9	164.618	9286.4	164.929	9699.5	172.266	164.830
2025	9699.0	172.257	10129.7	179.906	10148.8	180.245	10600.3	188.263	180.137
2030	10599.7	188.254	11070.3	196.612	11091.2	196.983	11584.6	205.746	196.865
2035	11584.0	205.735	12098.4	214.870	12121.2	215.276	12660.4	224.852	215.147
2040	12659.8	224.841	13221.9	234.824	13246.8	235.267	13836.1	245.733	235.126
2045	13835.4	245.720	14449.7	256.630	14477.0	257.115	15121.0	268.552	256.960
2050	15120.2	268.539	15791.5	280.462	15821.4	280.991	16525.2	293.491	280.823



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

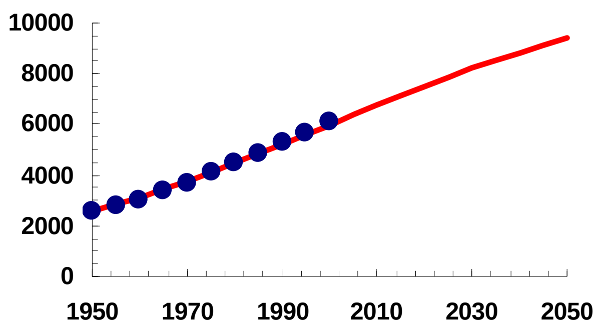
$$p = 2,560 \frac{12,000}{2,560 + (12,000 - 2,560)e^{-0.026(1955-1950)}} = 2,831.54$$

Values at the other times can be computed and displayed along with the data in the plot below. The analytical results are also included as the last column of the table 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	p_m	k_2	p_m	k_3	p_e	k_4	ϕ	p-analytical
1950	2560.00	52.361	2690.9	54.275	2695.7	54.343	2831.7	56.251	54.308	2560
1955	2831.54	56.249	2972.2	58.136	2976.9	58.198	3122.5	60.060	58.163	2831.54
1960	3122.35	60.058	3272.5	61.882	3277.1	61.935	3432.0	63.712	61.901	3122.355
1965	3431.86	63.710	3591.1	65.428	3595.4	65.472	3759.2	67.121	65.439	3431.859
1970	3759.05	67.119	3926.8	68.688	3930.8	68.723	4102.7	70.200	68.690	3759.052
1975	4102.50	70.199	4278.0	71.575	4281.4	71.601	4460.5	72.865	71.569	4102.503
1980	4460.35	72.864	4642.5	74.007	4645.4	74.024	4830.5	75.036	73.994	4460.349
1985	4830.32	75.036	5017.9	75.910	5020.1	75.920	5209.9	76.647	75.890	4830.319

1990	5209.77	76.647	5401.4	77.224	5402.8	77.227	5595.9	77.646	77.199	5209.771
1995	5595.77	77.646	5789.9	77.904	5790.5	77.905	5985.3	78.000	77.877	5595.767
2000	5985.15	78.000	6180.2	77.930	6180.0	77.930	6374.8	77.696	77.902	5985.154
2005	6374.66	77.696	6568.9	77.299	6567.9	77.301	6761.2	76.745	77.273	6374.666
2010	6761.03	76.745	6952.9	76.033	6951.1	76.040	7141.2	75.178	76.011	6761.033
2015	7141.09	75.179	7329.0	74.173	7326.5	74.187	7512.0	73.047	74.158	7141.09
2020	7511.88	73.047	7694.5	71.779	7691.3	71.802	7870.9	70.416	71.771	7511.878
2025	7870.73	70.417	8046.8	68.923	8043.0	68.956	8215.5	67.365	68.924	7870.733
2030	8215.35	67.366	8383.8	65.688	8379.6	65.732	8544.0	63.977	65.697	8215.351
2035	8543.84	63.979	8703.8	62.161	8699.2	62.214	8854.9	60.341	62.178	8543.837
2040	8854.73	60.343	9005.6	58.427	9000.8	58.490	9147.2	56.540	58.453	8854.728
2045	9146.99	56.542	9288.3	54.571	9283.4	54.642	9420.2	52.655	54.604	9146.992
2050	9420.01	52.658	9551.7	50.669	9546.7	50.746	9673.7	48.758	50.708	9420.011



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

22.6 The equations to be integrated are

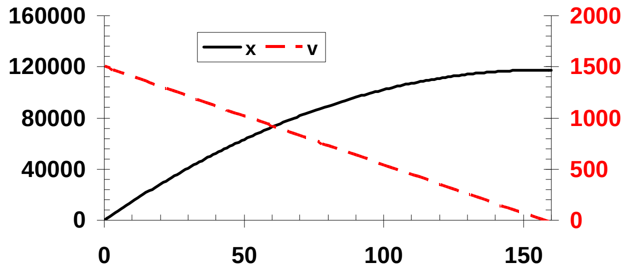
$$\frac{dv}{dt} = -9.81\frac{(6.37\times10^6)^2}{(6.37\times10^6 + x)^2}$$

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

The solution can be obtained with Euler's method with a step size of 1. Here are the results of the first few steps.

t	*v*	*x*	*dv/dt*	*dx/dt*
0	1500.000	0	-9.81000	1500.000
1	1490.190	1500.000	-9.80538	1490.190
2	1480.385	2990.190	-9.80080	1480.385
3	1470.584	4470.575	-9.79624	1470.584
4	1460.788	5941.158	-9.79173	1460.788
5	1450.996	7401.946	-9.78724	1450.996

The entire solution for height and velocity can be plotted as



The maximum height occurs at about 157 s. This result can be made more precise with the following script:

```
clear,clc
format short g
g=9.81;R=6.37e6;v0=1400;
dt=1/1024;t=0;
v=v0;x=0;
while (1)
  if v<=0, break, end
  dxdt=v;dvdt=-g*R^2/(R+x)^2;
  x=x+dxdt*dt;v=v+dvdt*dt;
  t=t+dt;
end
t,x,v

t =
       145.75
x =
  1.0149e+005
v =
   -0.0010876
```

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

t	*y*	*z*	*dy/dt*	*dz/dt*
0	2.0000	4.0000	1.00	-16.00
0.1	2.1000	2.4000	0.32	-6.05
0.2	2.1324	1.7952	-0.17	-3.44
0.3	2.1153	1.4516	-0.53	-2.23
0.4	2.0626	1.2287	-0.77	-1.56

(b) 4th-order RK method:

$$k_{1,1} = f_1(0,2,4) = -2(2) + 5e^{-0} = 1$$

$$k_{1,2} = f_2(0,2,4) = -\frac{2(4)^2}{2} = -16$$

$$y(0.1) = 2 + 1(0.05) = 2.05$$

$$z(0.05) = 4 - 16(0.05) = 3.2$$

$$k_{2,1} = f_1(0.05, 2.05, 3.2) = -2(2.05) + 5e^{-0.05} = 0.656$$

$$k_{2,2} = f_2(0.05, 2.05, 3.2) = -\frac{2.05(3.2)^2}{2} = -10.496$$

$$y(0.05) = 2 + 0.656(0.05) = 2.033$$

$$z(0.05) = 4 - 10.496(0.05) = 3.475$$

$$k_{3,1} = f_1(0.05, 2.033, 3.475) = -2(2.033) + 5e^{-0.05} = 0.691$$

$$k_{3,2} = f_2(0.05, 2.033, 3.475) = -\frac{2.033(3.475)^2}{2} = -12.275$$

$$y(0.1) = 2 + 0.691(0.1) = 2.069$$

$$z(0.1) = 4 - 12.275(0.1) = 2.772$$

$$k_{4,1} = f_1(0.1, 2.069, 2.772) = -2(2.069) + 5e^{-0.1} = 0.386$$

$$k_{4,2} = f_2(0.1, 2.069, 2.772) = -\frac{2.069(2.772)^2}{2} = -7.952$$

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

$$\phi_1 = \frac{1 + 2(0.656 + 0.691) + 0.386}{6} = 0.680$$

$$\phi_2 = \frac{-16 + 2(-10.496 - 12.275) - 7.952}{6} = -11.582$$

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

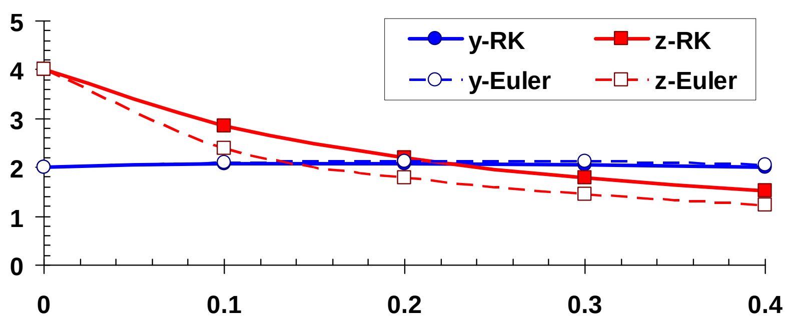
$$y(0.1) = 2 + 0.680(0.1) = 2.0680$$

$$z(0.1) = 4 - 11.582(0.1) = 2.8418$$

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

t	***y***	***z***
0	2.0000	4.0000
0.1	2.0680	2.8418
0.2	2.0827	2.1938
0.3	2.0576	1.7874
0.4	2.0036	1.5126

A plot of these values can be developed.



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

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

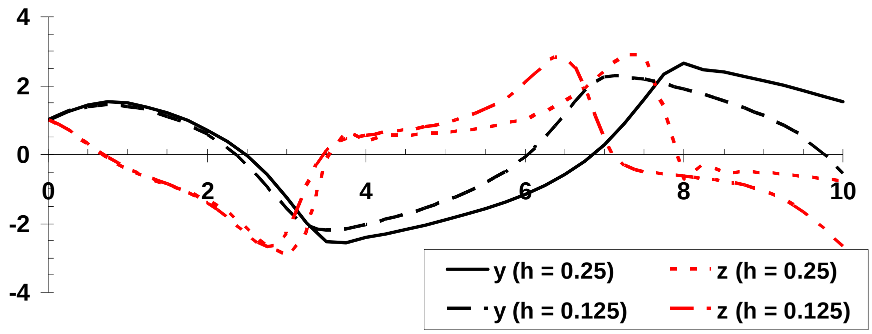
$$\frac{dz}{dt} = (1 - y^2)z - y$$

(a) Euler (h = 0.25). 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 = 0.25)	z (h = 0.25)	dy/dt	dz/dt
0	1	1	1	-1
0.25	1.25	0.75	0.75	-1.67188
0.5	1.4375	0.33203125	0.332031	-1.79158
0.75	1.52050781	-0.1158638	-0.11586	-1.3685
1	1.49154186	-0.457989	-0.45799	-0.93064

(b) Euler (h = 0.125). 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 = 0.125)	z (h = 0.125)	dy/dt	dz/dt
0	1	1	1	-1
0.125	1.125	0.875	0.875	-1.35742
0.25	1.234375	0.705322	0.705322	-1.60374
0.375	1.32254	0.504855	0.504855	-1.70073
0.5	1.385647	0.292263	0.292263	-1.65453



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

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

$$\frac{dz}{dt} = -4y$$

(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	1.0000	0.0000	0.0000	-4.0000	1.0000
0.1	1.0000	-0.4000	-0.4000	-4.0000	0.9801
0.2	0.9600	-0.8000	-0.8000	-3.8400	0.9211
0.3	0.8800	-1.1840	-1.1840	-3.5200	0.8253
0.4	0.7616	-1.5360	-1.5360	-3.0464	0.6967
0.5	0.6080	-1.8406	-1.8406	-2.4320	0.5403

(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.

$k_{1,1} = f_1(0,1,0) = z = 0$
$k_{1,2} = f_2(0,1,0) = -4y = -4(1) = -4$
$y(0.05) = 1 + 0(0.05) = 1$
$z(0.05) = 0 - 4(0.05) = -0.2$
$k_{2,1} = f_1(0.05,1,-0.2) = -0.2$
$k_{2,2} = f_2(0.05,1,-0.2) = -4(1) = -4$
$y(0.05) = 1 + (-0.2)(0.05) = 0.990$
$z(0.05) = 0 - 4(0.05) = -0.2$
$k_{3,1} = f_1(0.05, 0.990, -0.2) = -0.2$
$k_{3,2} = f_2(0.05, 0.990, -0.2) = -4(0.990) = -3.96$
$y(0.1) = 1 + (-0.2)(0.1) = 0.980$
$z(0.1) = 0 - 3.960(0.1) = -0.396$
$k_{4,1} = f_1(0.1, 0.980, -0.396) = -0.396$
$k_{4,2} = f_2(0.1, 0.980, -0.396) = -4(0.980) = -3.920$

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

$$\phi_1 = \frac{0 + 2(-0.2 - 0.2) - 0.396}{6} = -0.199$$
$$\phi_2 = \frac{-4 + 2(-4 - 3.960) - 3.920}{6} = -3.973$$

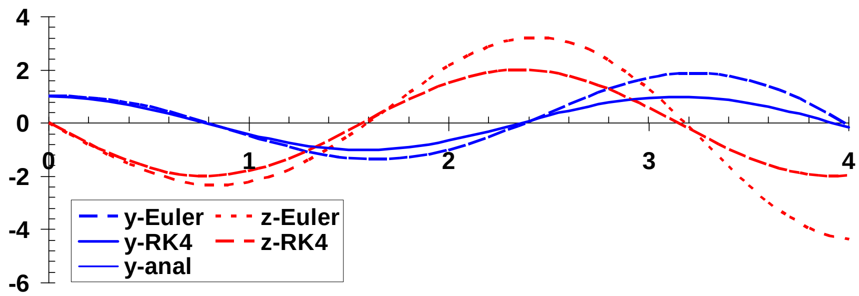
These slope estimates can then be used to make the prediction for the first step

$y(0.1) = 1 - 0.199(0.1) = 0.9801$
$z(0.1) = 0 - 3.973(0.1) = -0.3973$

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

t	***y*-RK4**	***z*-RK4**	***y*-anal**
0.0000	1.0000	0.0000	1.0000
0.1000	0.9801	-0.3973	0.9801
0.2000	0.9211	-0.7788	0.9211
0.3000	0.8253	-1.1293	0.8253
0.4000	0.6967	-1.4347	0.6967
0.5000	0.5403	-1.6829	0.5403

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)
```

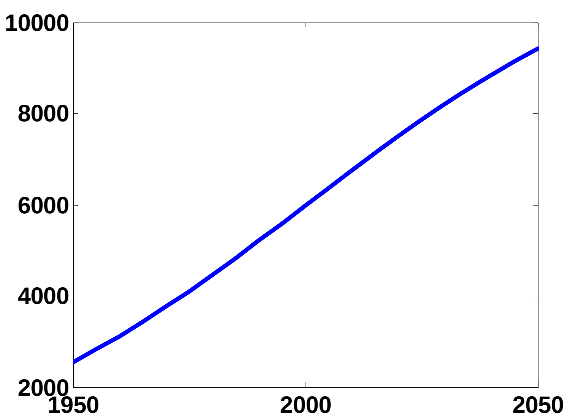
Here is the test of the solution of Prob. 22.5. First, an M-file holding the differential equation is written as

```
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],2560,5,0.1);
>> disp([t,p])
  1.0e+003 *
    1.9500    2.5600
    1.9550    2.8315
    1.9600    3.1223
    1.9650    3.4317
    1.9700    3.7587
    1.9750    4.1020
    1.9800    4.4596
    1.9850    4.8294
    1.9900    5.2085
    1.9950    5.5942
    2.0000    5.9833
    2.0050    6.3726
    2.0100    6.7587
    2.0150    7.1385
    2.0200    7.5090
    2.0250    7.8677
    2.0300    8.2121
    2.0350    8.5406
    2.0400    8.8516
    2.0450    9.1440
    2.0500    9.4173
```

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)
```

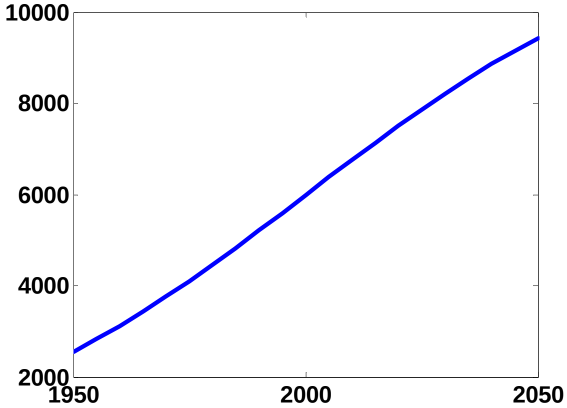
Here is the test of the solution of Prob. 22.5. First, an M-file holding the differential equation is written as

```
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 2000],2560,5);
>> disp([t,p])
  1.0e+003 *
    1.9500    2.5600
    1.9550    2.8314
    1.9600    3.1220
    1.9650    3.4314
    1.9700    3.7585
    1.9750    4.1020
    1.9800    4.4598
    1.9850    4.8298
    1.9900    5.2094
    1.9950    5.5955
    2.0000    5.9850
    2.0050    6.3747
    2.0100    6.7612
    2.0150    7.1413
    2.0200    7.5122
    2.0250    7.8711
    2.0300    8.2157
    2.0350    8.5441
    2.0400    8.8549
    2.0450    9.1470
    2.0500    9.4199
```

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)
```

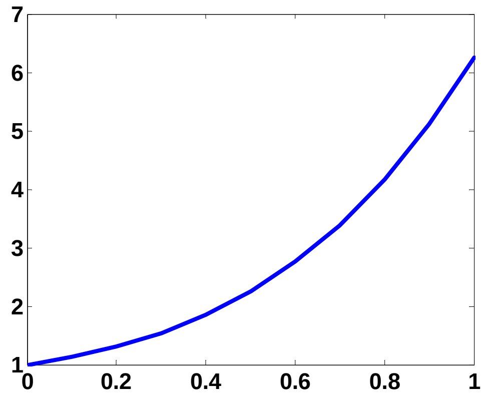
Here is the test of the solution of Prob. 22.2. First, an M-file holding the differential equation is written as

```
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    1.1236
    0.2000    1.2996
    0.3000    1.5376
    0.4000    1.8496
    0.5000    2.2500
    0.6000    2.7556
    0.7000    3.3856
    0.8000    4.1616
    0.9000    5.1076
    1.0000    6.2500
```

The following plot is generated



22.13 The following function is patterned on the code for the fourth-order RK method from Fig. 22.8:

```
function [t,y] = Eulersys(dydt,tspan,y0,h)
% [t,y] = Eulersys(dydt,tspan,y0,h):
%   uses Euler's method to integrate a system of ODEs
% input:
%   dydt = name of the M-file that evaluates the ODEs
%   tspan = [ti, tf] where ti and tf = initial and
%           final values of independent variable
%   y0 = initial values of dependent variables
%   h = step size
% output:
%   t = vector of independent variable
%   y = vector of solution for dependent variables

ti = tspan(1);
tf = tspan(2);
t = (ti:h:tf)';
n = length(t);
% if necessary, add an additional value of t
% so that range goes from t = ti to tf
if t(n)<tf
  t(n+1) = tf;
  n = n+1;
```

```
end
y(1,:) = y0;
for i = 1:n-1
  hh = t(i+1) - t(i);
  k1 = feval(dydt,t(i),y(i,:))';
  y(i+1,:) = y(i,:) + k1*hh;
end
plot(t,y(:,1),t,y(:,2),'--')
```

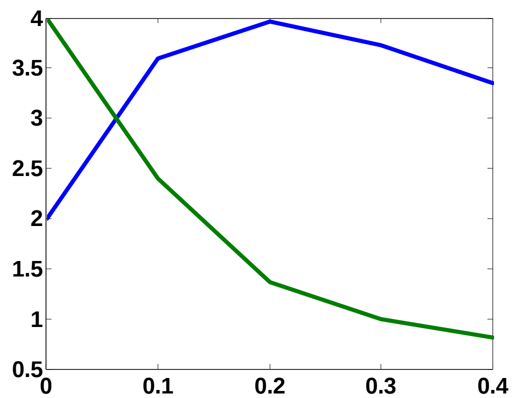
This code solves as many ODEs as are specified. Here is the test of the solution of Prob. 22.7. First, a single M-file holding the differential equations can be written as

```
function dy = dydtsys(t, y)
dy = [-2*y(1) + 5*y(2)*exp(-t);-y(1)*y(2)^2/2];
```

Then the M-file can be invoked as in

```
>> [t,y]=Eulersys(@dydtsys,[0 0.4],[2 4],0.1);
>> disp([t,y])

>> Prob2213Eulersys_Script
         0    2.0000    4.0000
    0.1000    3.6000    2.4000
    0.2000    3.9658    1.3632
    0.3000    3.7307    0.9947
    0.4000    3.3530    0.8101
```



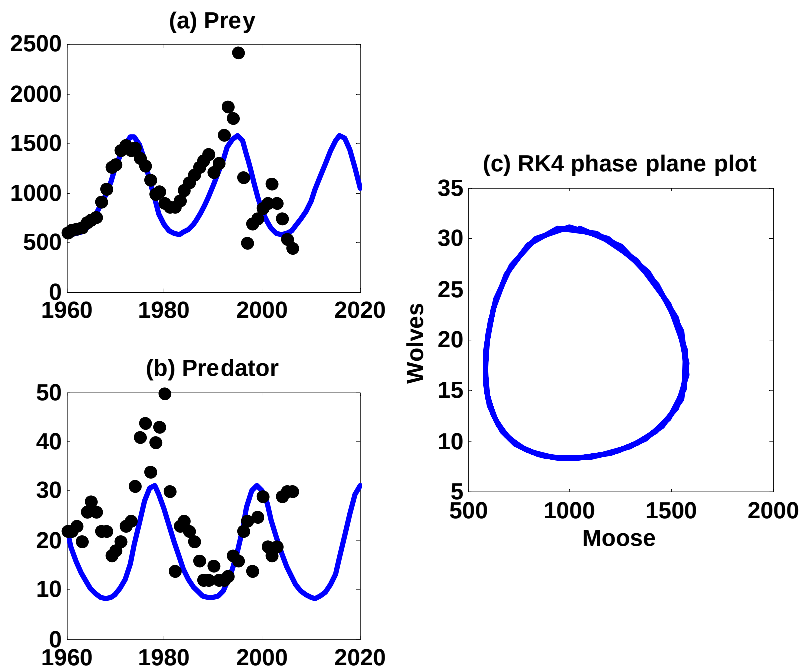
22.14 The following script uses the `rk4sys` function (Fig. 22.8) to solve the ODEs. The interp1 function is then used to determine the sum of the squares of the residuals.

```
clear,clc,clf
tdata=[1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973
1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989
1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005
2006];
Prey=[610 628 639 663 707 733 765 912 1042 1268 1295 1439 1493 1435 1467 1355
1282 1143 1001 1028 910 863 872 932 1038 1115 1192 1268 1335 1397 1216 1313
1590 1879 1770 2422 1163 500 699 750 850 900 1100 900 750 540 450];
Pred=[22 22 23 20 26 28 26 22 22 17 18 20 23 24 31 41 44 34 40 43 50 30 14 23
24 22 20 16 12 12 15 12 12 13 17 16 22 24 14 25 29 19 17 19 29 30 30];
h=0.0625;tspan=[1960:2020];y0=[610 22];
a=0.23;b=0.0133;c=0.4;d=0.0004;
[t y] = rk4sys(@predprey,tspan,y0,h,a,b,c,d);
% create plots
```

```
subplot(2,2,1);plot(t,y(:,1),tdata,Prey,'o')
title('(a) Prey')
subplot(2,2,3);plot(t,y(:,2),tdata,Pred,'o')
title('(b) Predator')
subplot(2,2,[2 4]);plot(y(:,1),y(:,2))
title('(c) RK4 phase plane plot')
xlabel('Moose'),ylabel('Wolves'),axis square
% determine sum of squares
SSRPred=0;SSRPrey=0;
for i=1:length(tdata)
  PreyRK4=interp1(t,y(:,1),tdata(i),'pchip');
  SSRPrey=SSRPrey+(Prey(i)-PreyRK4)^2;
end
for i=1:length(tdata)
  PredRK4=interp1(t,y(:,2),tdata(i),'pchip');
  SSRPred=SSRPred+(Pred(i)-PredRK4)^2;
end
SSRPrey,SSRPred
```

When the script is run, the result is

```
SSRPrey =
  4.5021e+006
SSRPred =
  4.8909e+003
```



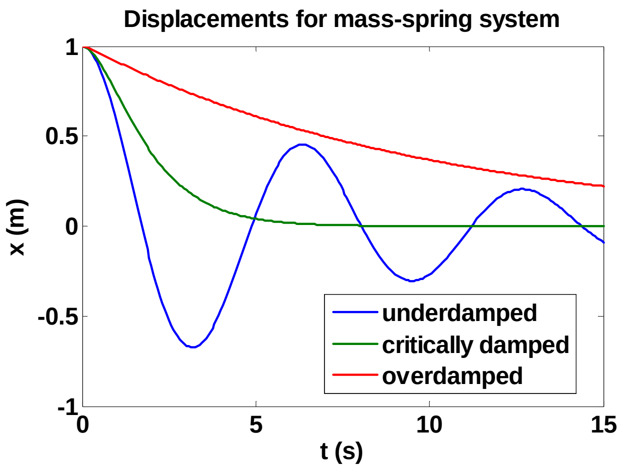
22.15 Function defining the derivatives:

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

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

```
clear,clc,clf
k=20;m=20;
tspan=[0:1/16:15];y0=[1 0];
[t1,y1]=rk4sys(@dydtTank,tspan,y0,1/16,m,k,5);
[t2,y2]=rk4sys(@dydtTank,tspan,y0,1/16,m,k,40);
[t3,y3]=rk4sys(@dydtTank,tspan,y0,1/16,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 and equation with 1 unknown,

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

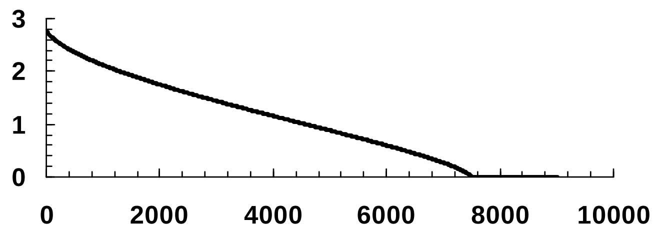
The area of the orifice can be computed as

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

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

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

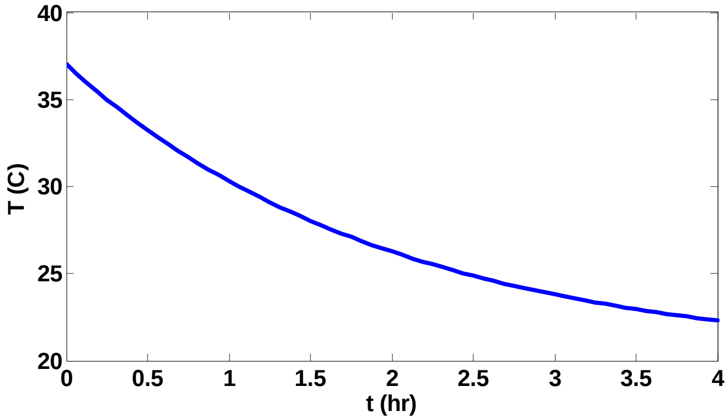
Thus, the person died 1.166 hrs prior to being discovered.

(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 Errata: On the first printing there were several mistakes in the mass balance equations. The correct equations should be:

$$\frac{dCA_1}{dt} = \frac{1}{\tau}(CA_0 - CA_1) - kCA_1$$

$$\frac{dCB_1}{dt} = -\frac{1}{\tau}CB_1 + kCA_1$$

$$\frac{dCA_2}{dt} = \frac{1}{\tau}(CA_1 - CA_2) - kCA_2$$

$$\frac{dCB_2}{dt} = \frac{1}{\tau}(CB_1 - CB_2) + kCA_2$$

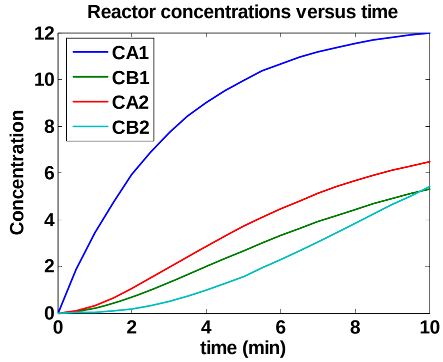
Function defining the derivatives:

```
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):

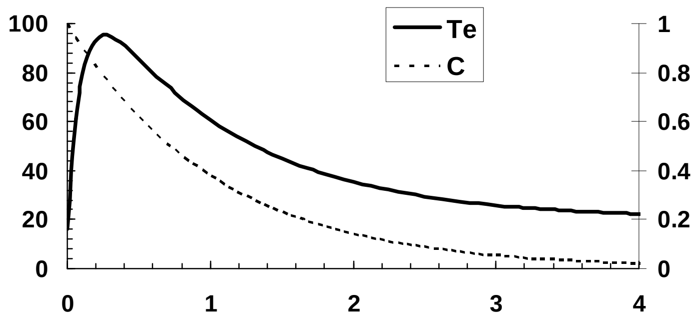
```
clear,clc,clf
CA0=20;k=0.12;tau=5;
tspan=[0:1/2:10];y0=[0 0 0 0];
[t,C]=rk4sys(@dCdtReactor,tspan,y0,1/16,tau,CA0,k);
plot(t,C(:,1),t,C(:,2),':',t,C(:,3),'--',t,C(:,4),'-.')
legend('CA1','CB1','CA2','CB2','location','best')
title('Reactor concentrations versus time')
xlabel('time (min)'),ylabel('Concentration')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

Output :



22.19 The classical 4th order RK method yields

t	*C*	*Te*
0	1	16
0.0625	0.941257	61.33579
0.125	0.885802	83.19298
0.1875	0.833553	92.53223
0.25	0.784367	95.27129
0.3125	0.738079	94.5932
0.375	0.694529	92.20458
0.4375	0.653556	89.01654
0.5	0.615011	85.51221
0.5625	0.578748	81.94466
0.625	0.544633	78.44362
0.6875	0.512538	75.07279
0.75	0.482341	71.86073
0.8125	0.453932	68.81748
0.875	0.427202	65.94338
0.9375	0.402052	63.23392
1	0.378387	60.68222
•		
•		
•		



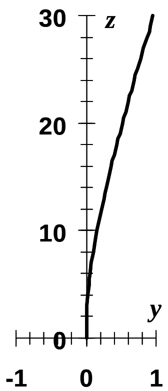
22.20 The second-order equation can be expressed as a pair of first-order equations,

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

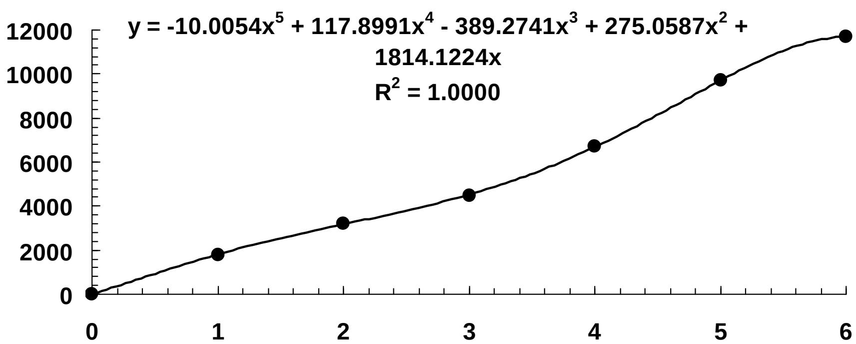
$$\frac{dw}{dz} = \frac{f}{2EI}(L-z)^2$$

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

z	y	w	dy/dz	dw/dz
0	0	0	0	0.004154
1	0	0.004154	0.004154	0.003882
2	0.004154	0.008035	0.008035	0.003618
3	0.012189	0.011654	0.011654	0.003365
4	0.023843	0.015018	0.015018	0.00312
5	0.038862	0.018138	0.018138	0.002885
•				
•				
•				
26	0.78	0.0435	0.0435	7.38E-05
27	0.8235	0.043574	0.043574	4.15E-05
28	0.867074	0.043615	0.043615	1.85E-05
29	0.910689	0.043634	0.043634	4.62E-06
30	0.954323	0.043638	0.043638	0



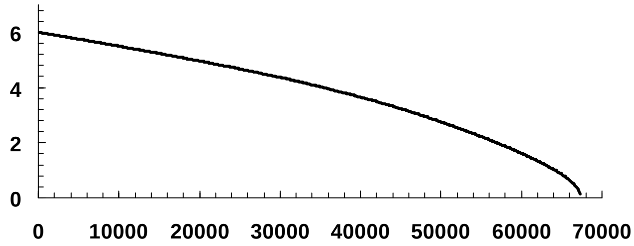
22.21 This problem can be approached in a number of ways. The simplest way is to fit the area-depth data with polynomial regression. A fifth-order polynomial with a zero intercept yields a perfect fit:



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.



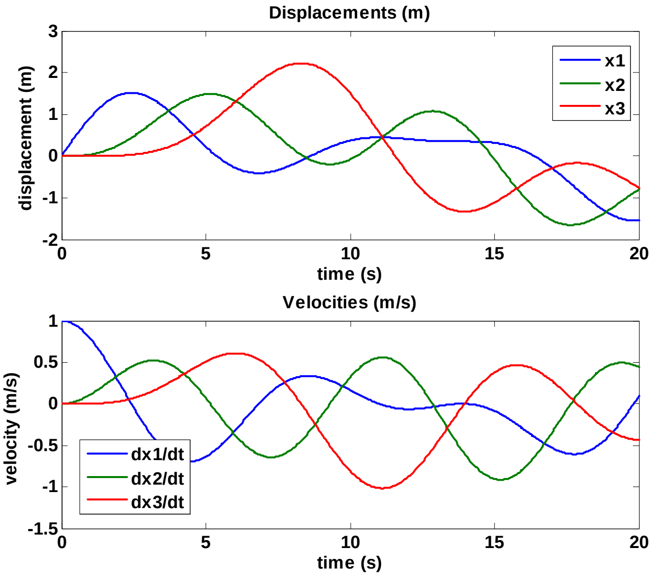
22.22 Function defining the derivatives:

```
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)')
```

Output :



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:

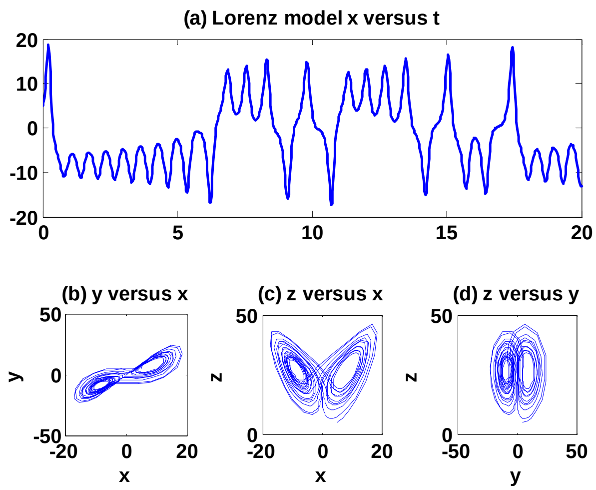
```
function yp=lorenz(t,y,sigma,b,r)
yp=[-sigma*y(1)+sigma*y(2);r*y(1)-y(2)-y(1)*y(3);-b*y(3)+y(1)*y(2)];
```

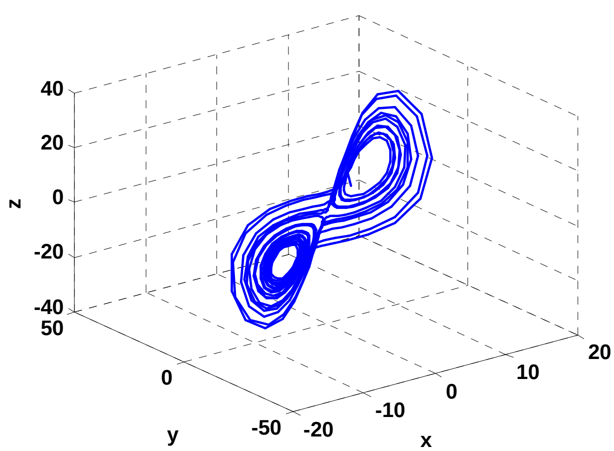
The following script solves the ODEs and generates the plots:

```
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
```