CHAPTER 20

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

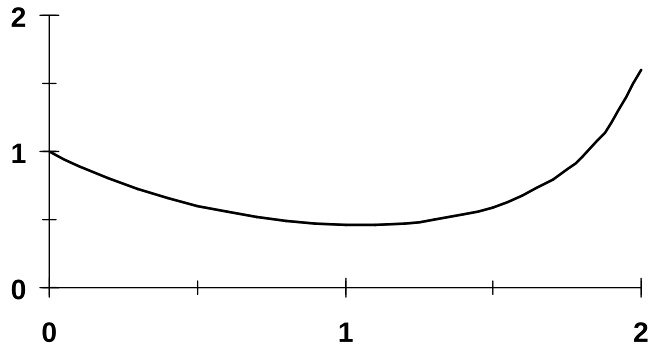
$$\int \frac{dy}{y} = \int t^2 - 1.1\, dt$$

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

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 $h = 0.5$

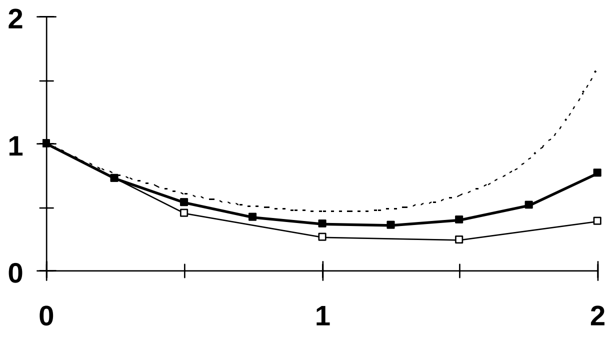
x	*y*	*dy/dx*
0	1	-1.1
0.5	0.45	-0.3825
1	0.25875	-0.02588
1.5	0.245813	0.282684
2	0.387155	1.122749

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



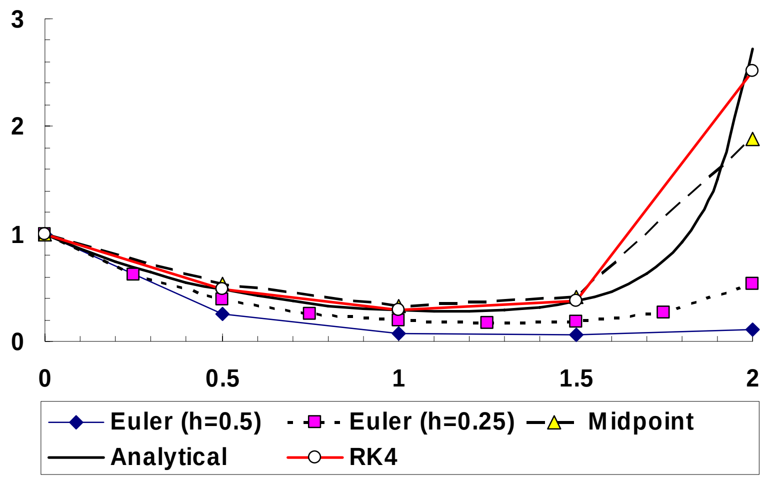
(c) Midpoint method (h = 0.5)

t	y	dy/dt	t_m	y_m	dy_m/dt
0	1	-1.5	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



20.2 **(a)** The analytical soilution 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
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:

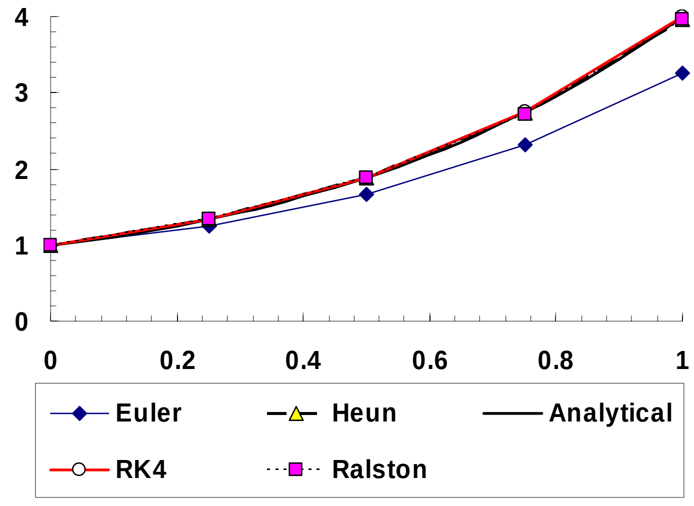
$$y(0.25) = 1 + \frac{1 + 2(1.49837)}{3} 0.25 = 1.33306$$

The remaining steps can be implemented and summarized as

x	*y*	k_1	*x* + 3/4*h*	*y* + (3/4)k_1*h*	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**											



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

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*$_1$	*t*$_{i+1}$	*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

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

The corrector can then be iterated to give

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

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

The iterations can be continued until the percent relative error falls below 0.1%. This occurs after 5 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

t	*y*
0	1.0000000
0.5	0.6499634
1	0.6399316
1.5	1.0339067
2	1.8705671
2.5	3.1725717
3	4.9525967

(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:

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

$$y(0.375) = 1 + (-1)(0.375) = 0.625$$

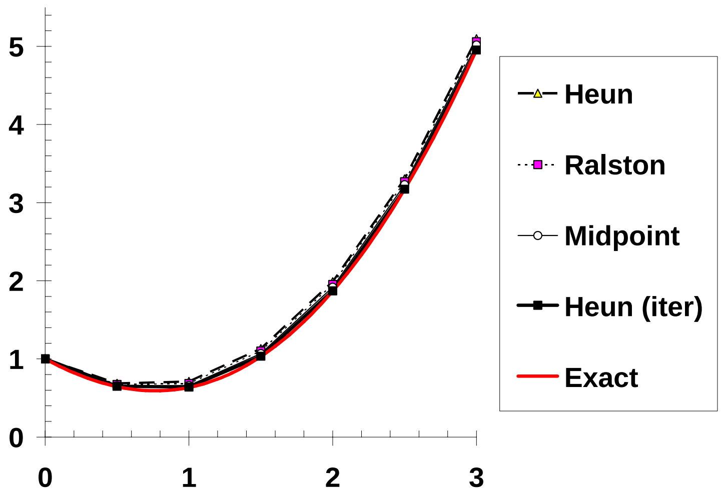
$$k_2 = -(0.625) + 0.375^2 = -0.6875$$

$$y(0.25) = 1 + \frac{-1 + 2(-6875)}{3} 0.5 = 0.65625$$

The remaining steps can be implemented and summarized as

t	*y*	k_1	*t* + 3/4*h*	*y* + (3/4)k_1h	k_2	*dy*/*dt*
0	**1**	-1.0000	0.375	0.625	-0.4844	-0.6563
0.5	**0.67188**	-0.4219	0.875	0.513672	0.2520	0.02734
1	**0.68555**	0.3145	1.375	0.803467	1.0872	0.82959
1.5	**1.10034**	1.1497	1.875	1.531464	1.9842	1.70599
2	**1.95334**	2.0467	2.375	2.720837	2.9198	2.62875
2.5	**3.26771**	2.9823	2.875	4.38607	3.8796	3.58047
3	**5.05794**	3.9421				

All the versions can be plotted along with the exact solution:



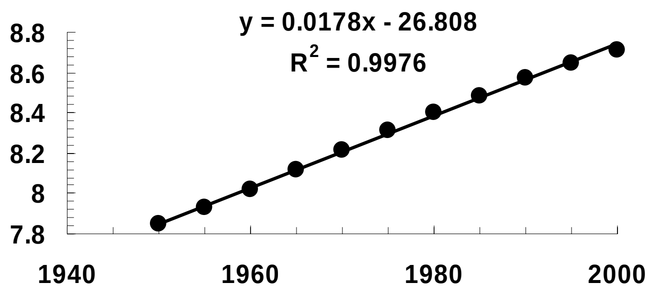
20.4 **(a)** The solution to the differential equation is

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

Taking the natural log of this equation gives

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

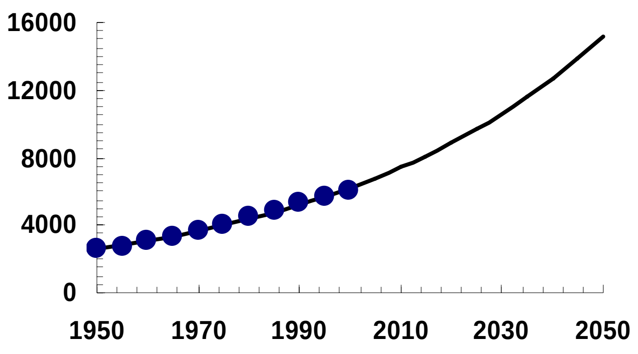
Therefore, a semi-log plot (ln *p* versus *t*) should yield a straight line with a slope of k_g. The plot, along with the linear regression best fit line is shown below. The estimate of the population growth rate is k_g = 0.0178/yr.



(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	2555.00	45.41	2668.53	47.43	2673.58	47.52	2792.60	49.64	47.49
1955	2792.46	49.63	2916.55	51.84	2922.06	51.94	3052.15	54.25	51.91
1960	3051.99	54.25	3187.61	56.66	3193.64	56.76	3335.81	59.29	56.73

1965	3335.64	59.29	3483.87	61.92	3490.45	62.04	3645.84	64.80	62.00
1970	3645.66	64.80	3807.66	67.68	3814.85	67.81	3984.69	70.82	67.77
1975	3984.48	70.82	4161.54	73.97	4169.41	74.11	4355.02	77.41	74.06
1980	4354.80	77.40	4548.31	80.84	4556.91	81.00	4759.78	84.60	80.95
1985	4759.54	84.60	4971.03	88.36	4980.43	88.52	5202.15	92.46	88.47
1990	5201.89	92.46	5433.04	96.57	5443.31	96.75	5685.64	101.06	96.69
1995	5685.35	101.05	5937.98	105.54	5949.21	105.74	6214.06	110.45	105.68
2000	6213.75	110.44	6489.86	115.35	6502.13	115.57	6791.60	120.72	115.50
2005	6791.25	120.71	7093.02	126.07	7106.43	126.31	7422.81	131.93	126.24
2010	7422.43	131.93	7752.25	137.79	7766.90	138.05	8112.68	144.20	137.97
2015	8112.27	144.19	8472.74	150.60	8488.76	150.88	8866.67	157.60	150.79
2020	8866.22	157.59	9260.20	164.59	9277.70	164.90	9690.74	172.25	164.80
2025	9690.24	172.24	10120.84	179.89	10139.97	180.23	10591.40	188.25	180.12
2030	10590.85	188.24	11061.47	196.61	11082.38	196.98	11575.76	205.75	196.86
2035	11575.17	205.74	12089.52	214.88	12112.37	215.29	12651.61	224.87	215.16
2040	12650.96	224.86	13213.11	234.85	13238.09	235.30	13827.45	245.77	235.16
2045	13826.74	245.76	14441.14	256.68	14468.44	257.17	15112.57	268.61	257.01
2050	15111.79								



20.5 **(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

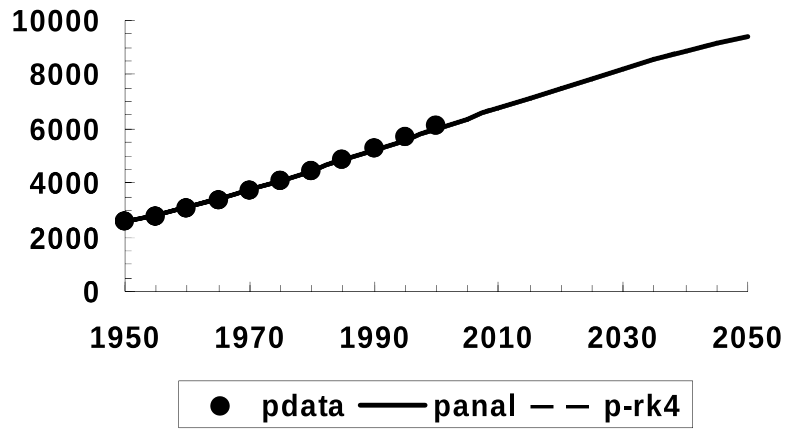
$$p = 2{,}555 \frac{12{,}000}{2{,}555 + (12{,}000 - 2{,}555)e^{-0.026(1955-1950)}} = 2{,}826.2$$

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.

20.6 The equations to be integrated are

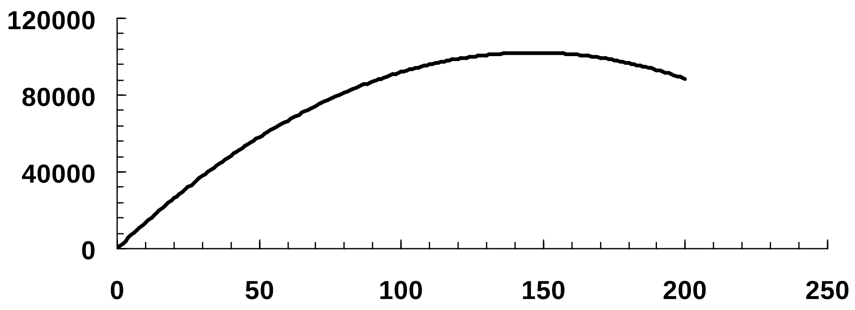
$$\frac{dv}{dt} = -9.81\frac{(6.37\times10^6)^2}{(6.37\times10^6 + x)^2} \qquad\qquad \frac{dx}{dt} = v$$

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	1400	0	-9.81	1400
1	1390.19	1400	-9.80569	1390.19
2	1380.384	2790.19	-9.80141	1380.384

3	1370.583	4170.574	-9.79717	1370.583
4	1360.786	5541.157	-9.79296	1360.786
5	1350.993	6901.943	-9.78878	1350.993

The entire solution for height can be plotted as



The maximum height occurs at about 146 s.

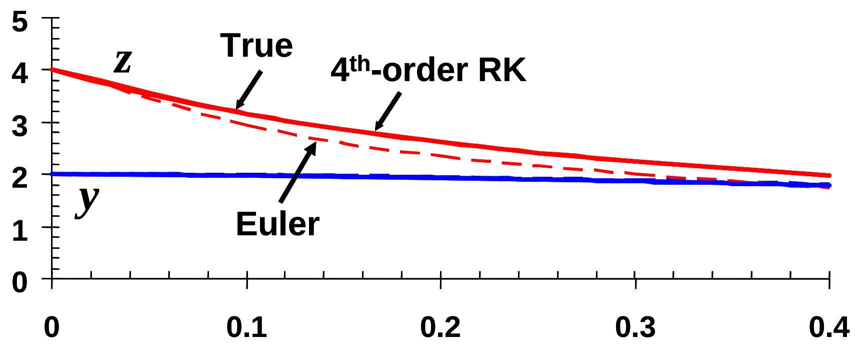
20.7 (*a*) Euler

x	*y*	*z*	*dy/dx*	*dz/dx*
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

x	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

Both methods are plotted on the same graph below. Notice how Euler's method (particularly for *z*) is not very accurate for this step size. The 4th-order RK is visually indistinguishable from the exact solution.



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

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

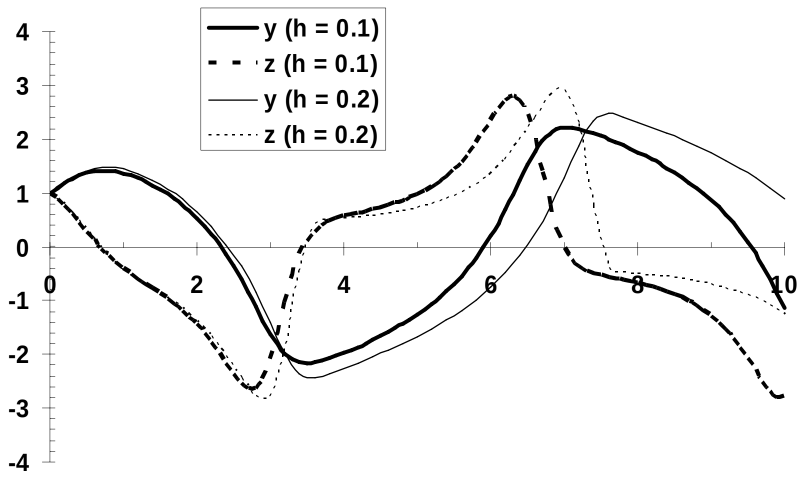
$$\frac{dz}{dt} = (1 - y^2)z - y$$

(a) Euler (h = 0.2). 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



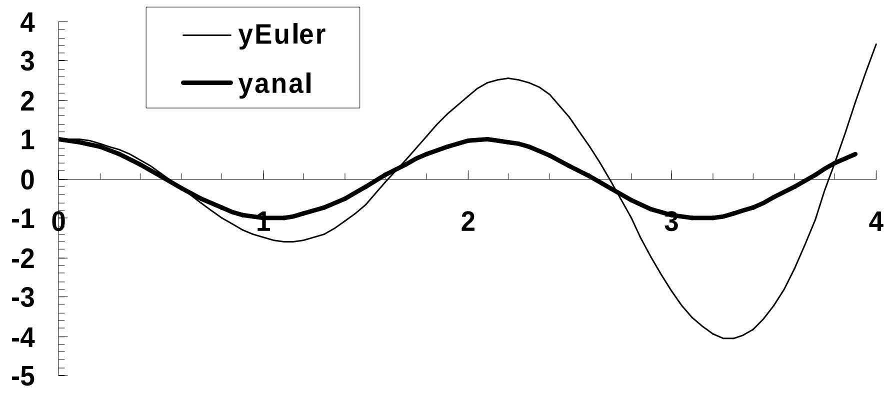
20.9 The second-order equation can be reexpressed as a system of two first-order ODEs,

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

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

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

t	*y*y_{Euler}	z_{Euler}	*dy/dt*	*dz/dt*	$y_{\text{analytical}}$
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

$$y(0.1) = 1 - 0.4466(0.1) = 0.9553$$

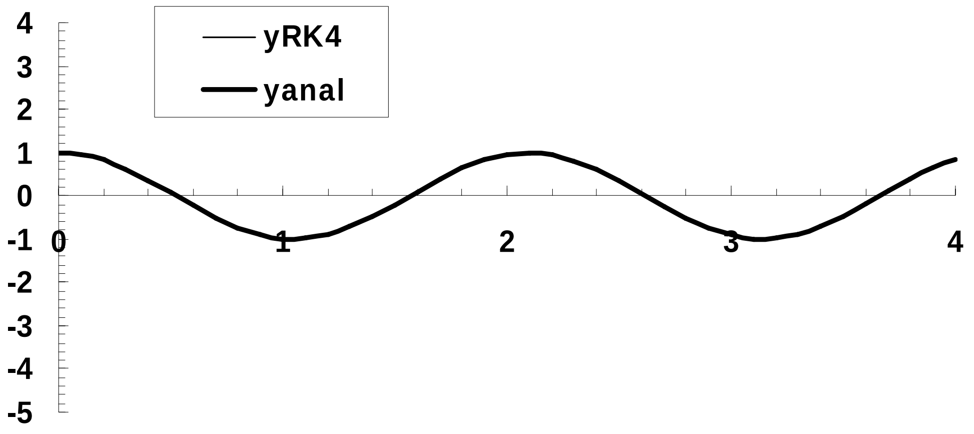
$$z(0.1) = 0 - 8.8650(0.1) = -0.8865$$

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

t	*y*	*z*	yanal

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.



20.10 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)
```

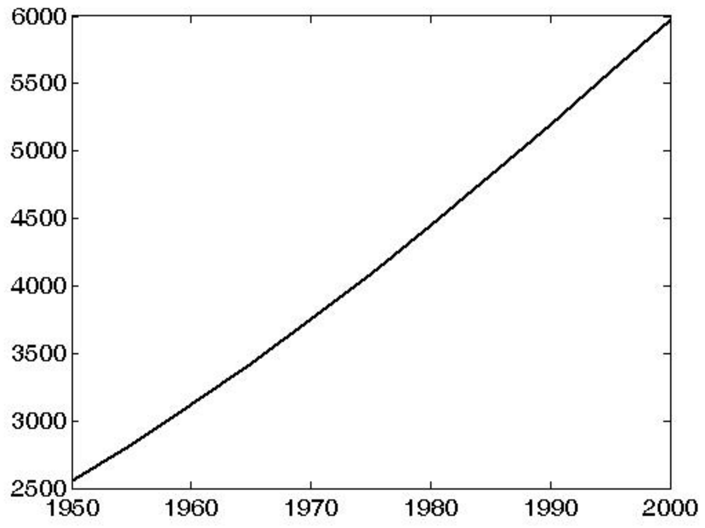
Here is the test of the solution of Prob. 20.5. First, an M-file holding the differential equation is written as

```
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 2000],2555,5,0.1);
>> disp([t,p])
  1.0e+003 *

    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
```

The following plot is generated



20.11 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)
```

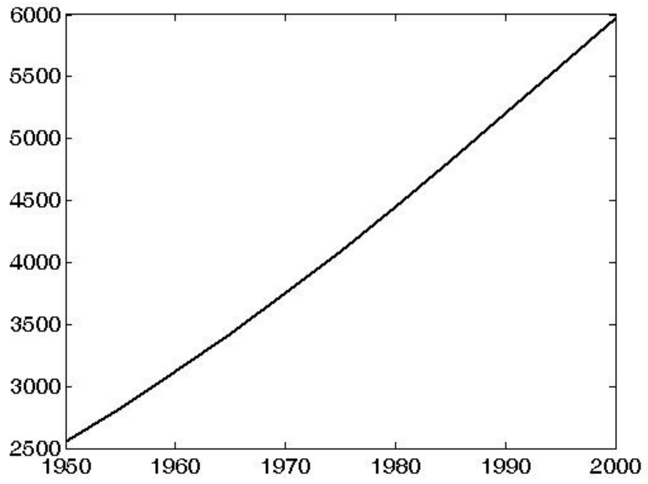
Here is the test of the solution of Prob. 20.5. First, an M-file holding the differential equation is written as

```
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],2555,5);
>> disp([t,p])
  1.0e+003 *
    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
```

The following plot is generated



20.12 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)
```

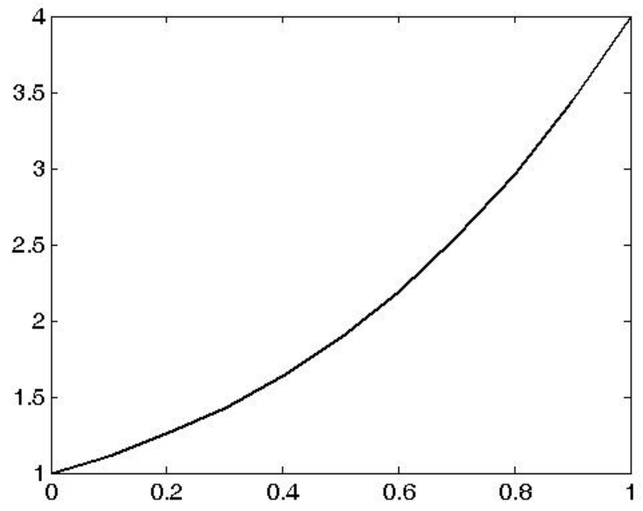
Here is the test of the solution of Prob. 20.2. First, an M-file holding the differential equation is written as

```
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



20.13

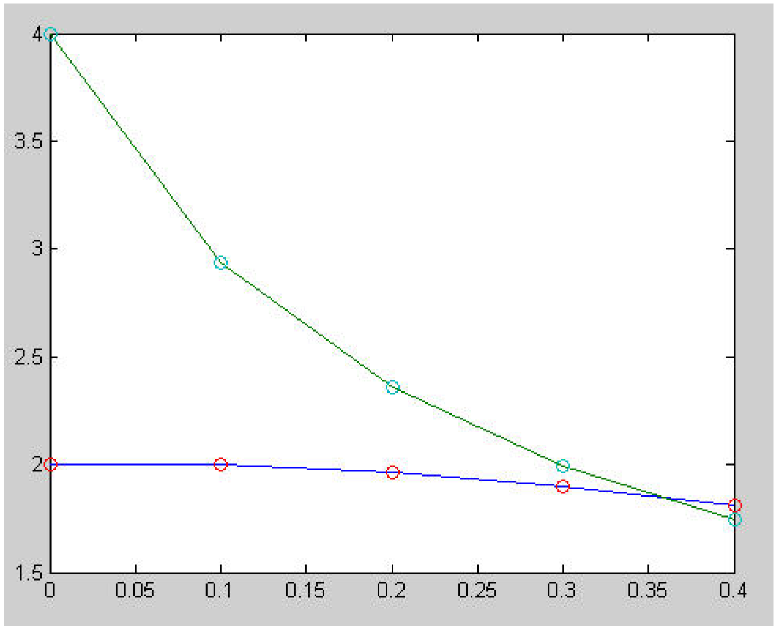
```
function [tp,yp] = eulsys(dydt,tspan,y0,h,varargin)
% eulsys: Euler method for a system of ODEs
%   [t,y] = eulsys(dydt,tspan,y0,h,p1,p2,...): integrates a
%                  system of ODEs with Euler's method
% input:
%   dydt = name of the M-file that evaluates the ODEs
%   tspan = [ti, tf]; initial 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
    k1 = dydt(tt,y(i,:),varargin{:})';
    y(i+1,:) = y(i,:) + k1*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
plot(tp',yp,tp',yp,'o')
```

Here is the test of the solution of Prob. 20.7. First, an M-file holding the differential equations is written as

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

Then the M-file can be invoked as in

```
>> [t,y]=eulsys(@dydtsys,[0 0.4],[2 4],0.1);
>> disp([t,y])
         0    2.0000    4.0000
    0.1000    2.0000    2.9333
    0.2000    1.9619    2.3597
    0.3000    1.8970    1.9956
    0.4000    1.8140    1.7437
```

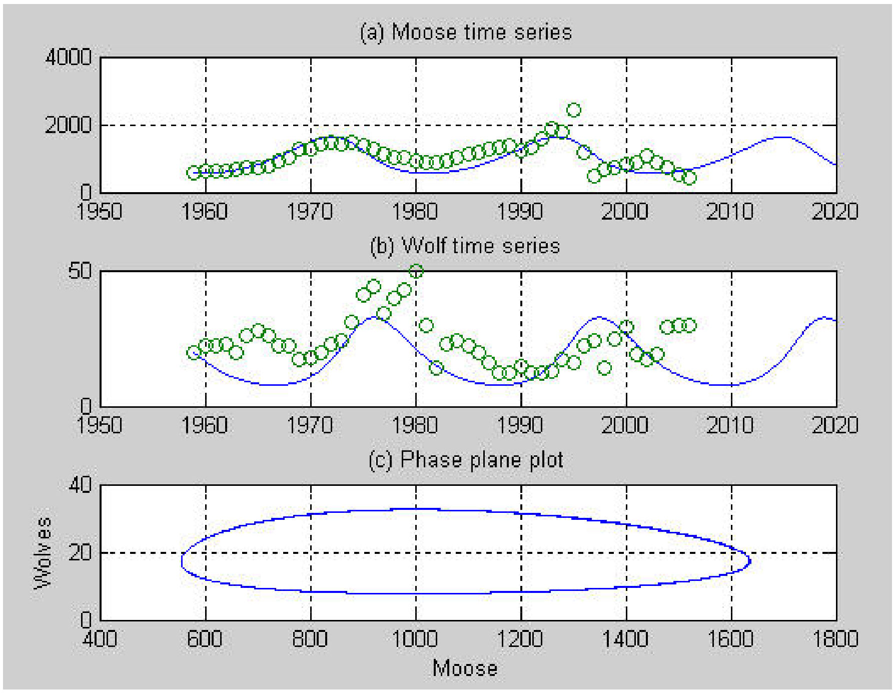


20.14 (a) The following script generates the solution:

```
h=0.0625;tspan=[1959 2020];y0=[563 20];
a=0.23;b=0.0133;c=0.4;d=0.0004;
td=[1959 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];
Md=[563 610 628 639 663 707 733 765 912 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];
Wd=[20 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];
[t y] = rk4sys(@predprey,tspan,y0,h,a,b,c,d);
subplot(3,1,1);plot(t,y(:,1),td,Md,'o')
title('(a) Moose time series'),grid
subplot(3,1,2);plot(t,y(:,2),td,Wd,'o')
title('(b) Wolf time series'),grid
subplot(3,1,3);plot(y(:,1),y(:,2))
title('(c) Phase plane plot'),grid,xlabel('Moose'),ylabel('Wolves')
ssrM=0;ssrW=0;
for i = 1:length(td)
  for j = 1:length(t)
    if td(i)==t(j)
      ssrM=ssrM+(Md(i)-y(j,1))^2;
      ssrW=ssrW+(Wd(i)-y(j,2))^2;
    end
  end
end
fprintf('SSR(Moose) = %8.4g\n',ssrM)
fprintf('SSR(Wolves) = %8.4g\n',ssrW)
```

When the script is run, the following results are generated:

```
SSR(Moose) = 3.866e+006
SSR(Wolves) =     5872
```



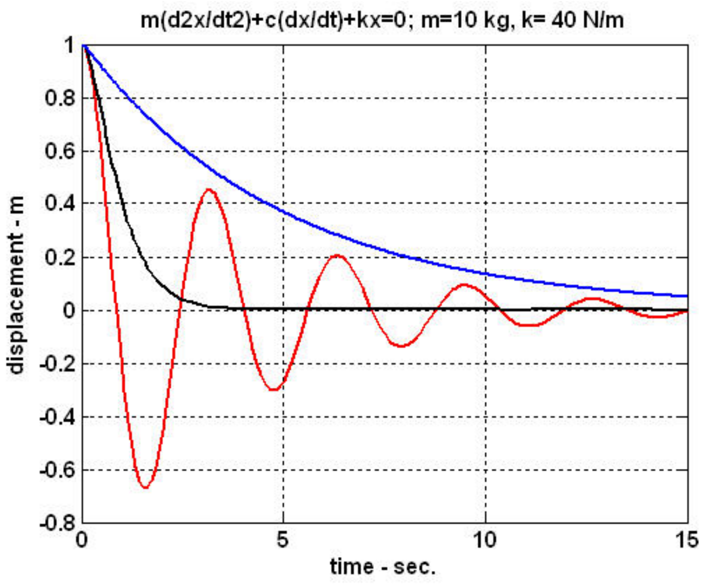
20.15 Main Program:

```
%Damped spring mass system
%mass:  m=10 kg
%damping: c=5,40,200 N/(m/s)
%spring: k=40 N/m
% MATLAB 5 version
%Independent Variable t, tspan=[tstart tstop]
%initial conditions [x(1)=velocity, x(2)=displacement];

tspan=[0 15];  ic=[0 1];

global cm km
m=10; c(1)=5; c(2)=40; c(3)=200; k=40;
km=k/m;

for n=1:3
  cm=c(n)/m;
  [t,x]=ode45('kc',tspan,ic);
  plot(t,x(:,2)); grid;
  xlabel('time - sec.'); ylabel('displacement - m');
  title('m(d2x/dt2)+c(dx/dt)+kx=0; m=10 kg, k= 40 N/m')
  hold on
end
```

Function ‘kc’:

```
%Damped spring mass system - m d2x/dt2 + c dx/dt + k x =0
%mass:  m=10 kg
%damping: c=5,40,200 N/(m/s)
%spring: k=40 N/m
```

```
%x(1)=velocity, x(2)=displacement

function dx=kc(t,x);
global cm km
dx=[-cm*x(1)-km*x(2); x(1)];
```



20.16 The volume of the tank can be computed as

$$\frac{dV}{dt} = -CA\sqrt{2gH} \qquad (1)$$

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

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

Equation (2) can be differentiated to give

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

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

$$\frac{dH}{dt} = -\frac{CA\sqrt{2gH}}{2\pi r H - \pi H^2} \qquad (4)$$

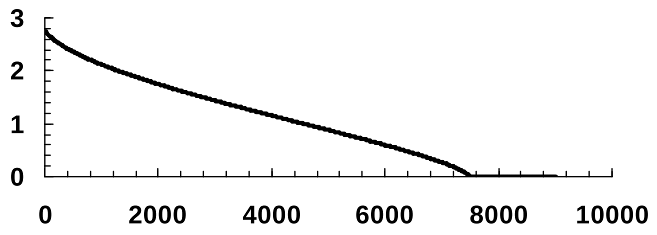
The area of the orifice can be computed as

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

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

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

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.

20.17 (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,

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

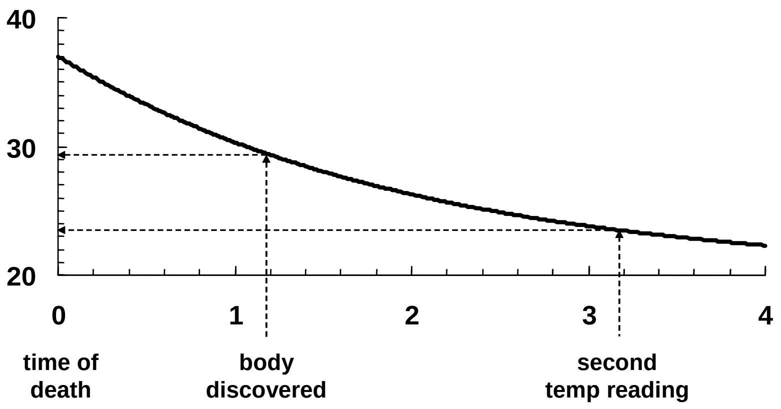
Substituting the values yields

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

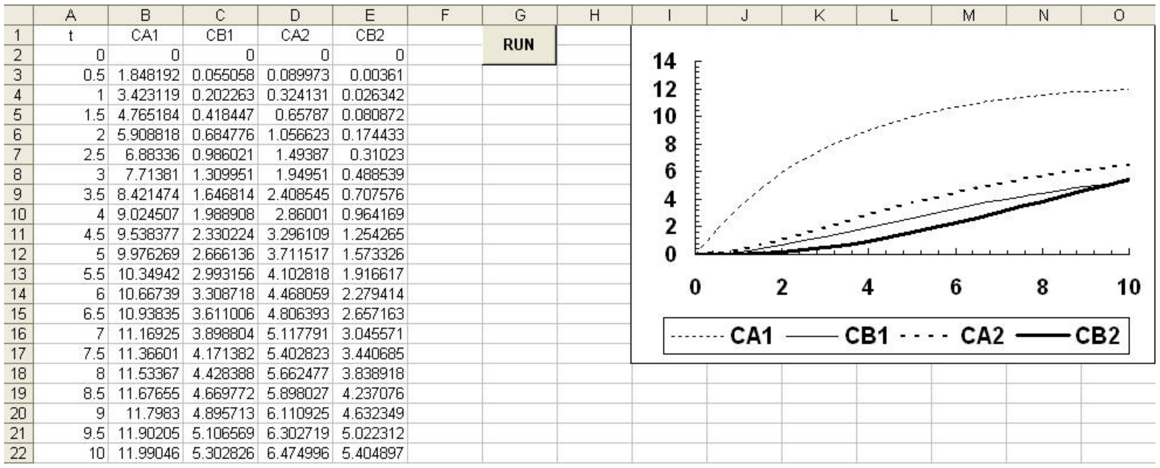
The time of death can then be computed as

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

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



20.18 The classical 4th order RK method yields

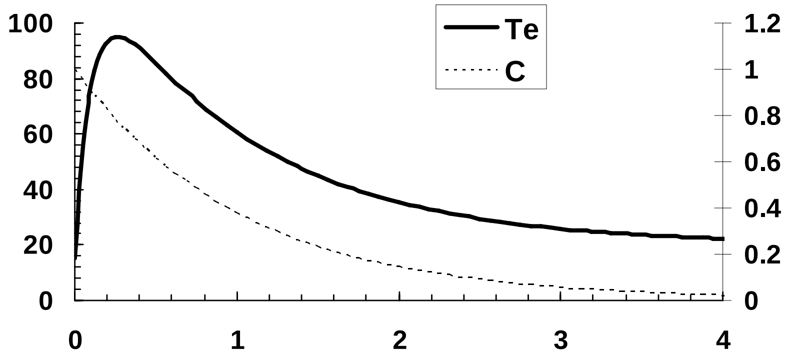


	A	B	C	D	E
1	t	CA1	CB1	CA2	CB2
2	0	0	0	0	0
3	0.5	1.848192	0.055058	0.089973	0.00361
4	1	3.423119	0.202263	0.324131	0.026342
5	1.5	4.765184	0.418447	0.65787	0.080872
6	2	5.908818	0.684776	1.056623	0.174433
7	2.5	6.88336	0.986021	1.49387	0.31023
8	3	7.71381	1.309951	1.94951	0.488539
9	3.5	8.421474	1.646814	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

20.19 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

•
•
•

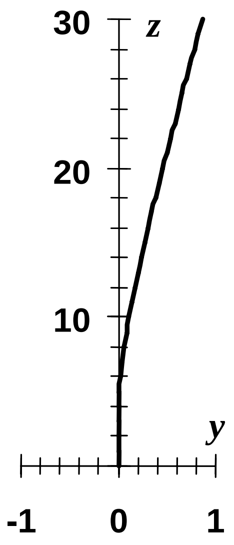


20.20 The second-order equation can be expressed as a pair of first-order equations,

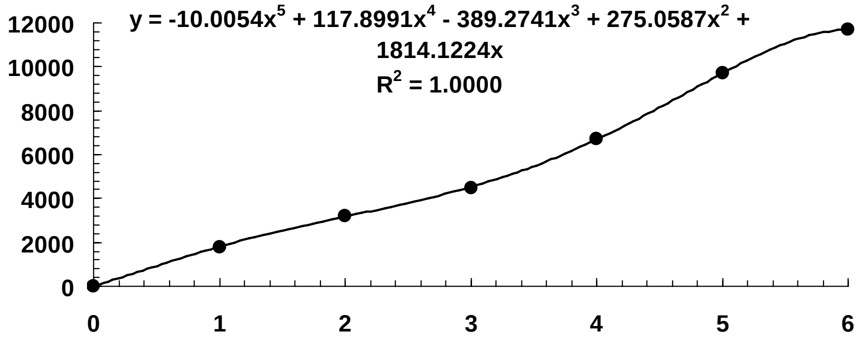
$$\frac{dy}{dz} = w \qquad \frac{dw}{dz} = \frac{200ze^{-2z/30}}{(5+z)2EI}(L-z)^2$$

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



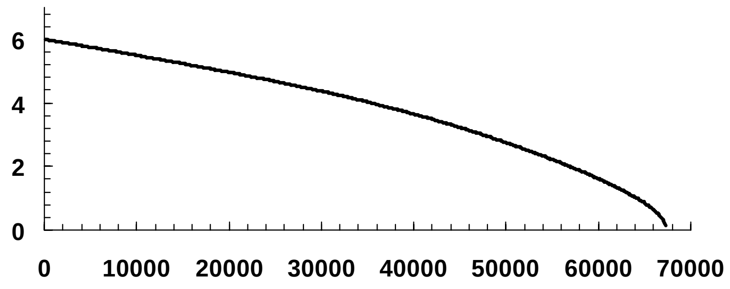
20.21 This problem can be approached in a number of ways. The simplest way is to fit the area-depth data with polynomial regression. A fifth-order polynomial with a zero intercept yields a perfect fit:



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.



20.22 (a) and **(b)** The second-order equations can be expressed as the following system of first-order ODEs,

$$\frac{dx_1}{dt} = x_4 \qquad \frac{dx_2}{dt} = x_5 \qquad \frac{dx_3}{dt} = x_6$$

$$\frac{dx_4}{dt} = -\frac{k_1}{m_1}x_1 + \frac{k_2}{m_1}(x_2 - x_1)$$

$$\frac{dx_5}{dt} = \frac{k_2}{m_2}(x_1 - x_2) + \frac{k_3}{m_2}(x_3 - x_2)$$

$$\frac{dx_6}{dt} = \frac{k_3}{m_3}(x_2 - x_3)$$

We can develop an M-file to compute them,

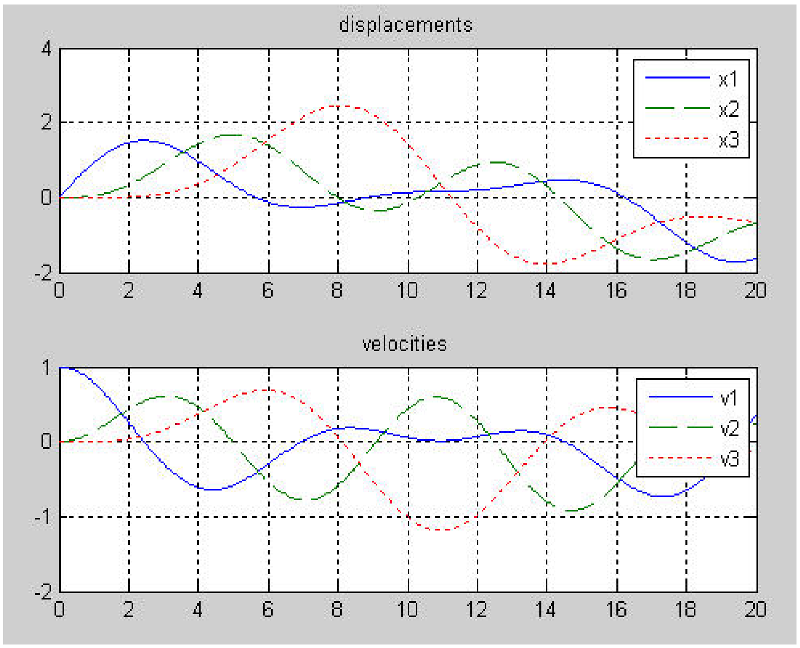
```
function dx = dxdtProb2022(t,x)
k1=3000;k2=2400;k3=1800;
m1=12000;m2=10000;m3=8000;
dx = [x(4);x(5);x(6);-k1/m1*x(1)+k2/m1*(x(2)-x(1));k1/m2*(x(1)-
x(2))+k3/m2*(x(3)-x(2));k3/m3*(x(2)-x(3))];
```

The following statement employs the `rk4sys` function (Fig. 20.8) to generate the solution.

```
>> [t,x]=rk4sys(@dxdtProb2022,[0 20],[0 0 0 1 0 0],.0625);
```

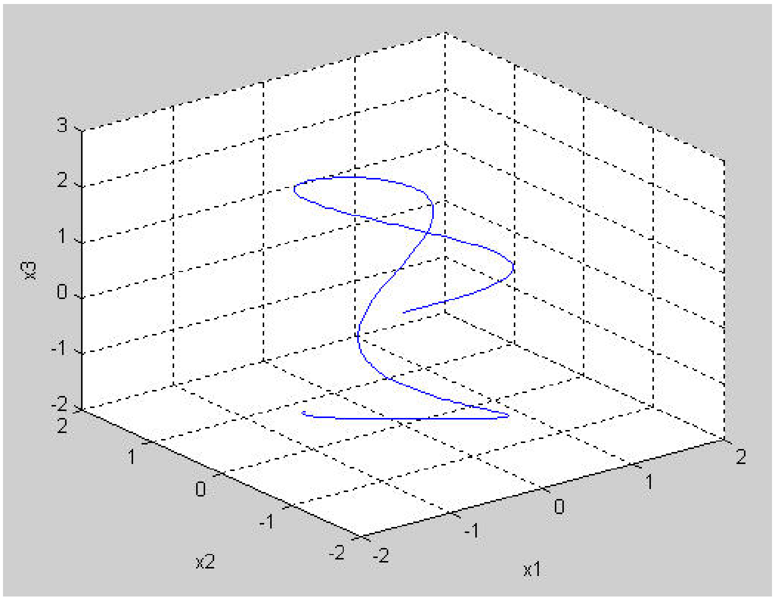
Plots of displacement and velocity can then be generated,

```
>> subplot(2,1,1);plot(t,x(:,1),t,x(:,2),'--',t,x(:,3),':')
>> grid;title('displacements');legend('x1','x2','x3')
>> subplot(2,1,2);plot(t,x(:,4),t,x(:,5),'--',t,x(:,6),':')
>> grid;title('velocities');legend('v1','v2','v3')
```



(c) The three-dimensional phase-plane plot of the displacements can be generated as

```
plot3(x(:,1),x(:,2),x(:,3))
xlabel('x1');ylabel('x2');zlabel('x3');grid
```



20.23 An M-file can be developed to hold the Lorenz model,

```
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)];
```

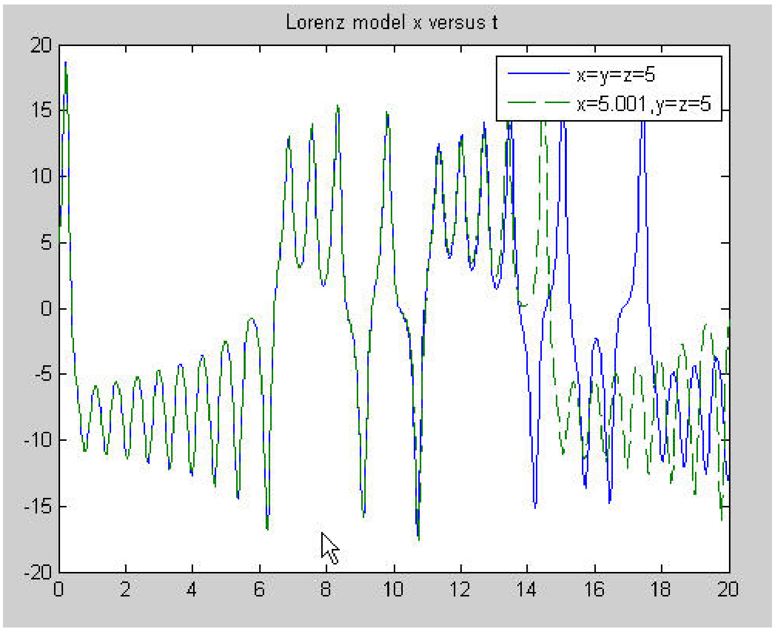
Here is an M-file to implement the midpoint method for a system of ODEs,

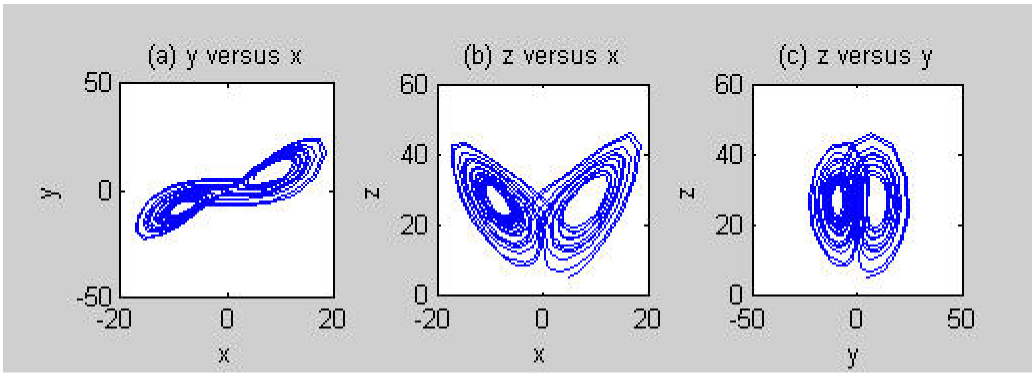
```
function [tp,yp] = midpointsys(dydt,tspan,y0,h,varargin)
% midpointsys: midpoint method for a system of ODEs
%   [t,y] = midpointsys(dydt,tspan,y0,h,p1,p2,...): integrates a
%                  system of ODEs with fourth-order RK method
% input:
%   dydt = name of the M-file that evaluates the ODEs
%   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
    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
```

Here is a script to integrate the equations and generate the plots,

```
h=0.03125;tspan=[0 20];y0=[5 5 5];
sigma=10;b=8/3;r=28;
```

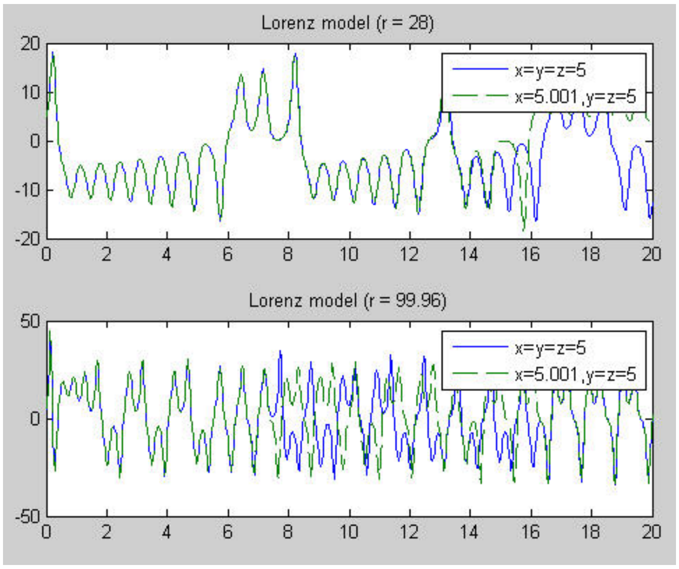
```
[t y] = midpointsys(@lorenz,tspan,y0,h,sigma,b,r);
y0=[5.001 5 5];
[tp yp] = midpointsys(@lorenz,tspan,y0,h,sigma,b,r);
plot(t,y(:,1),tp,yp(:,1),'--')
title('Lorenz model x versus t');legend('x=y=z=5','x=5.001,y=z=5')
pause
subplot(1,3,1);plot(y(:,1),y(:,2))
xlabel('x');ylabel('y')
axis square;title('(a) y versus x')
subplot(1,3,2);plot(y(:,1),y(:,3))
xlabel('x');ylabel('z')
axis square;title('(b) z versus x')
subplot(1,3,3);plot(y(:,2),y(:,3))
xlabel('y');ylabel('z')
axis square;title('(c) z versus y')
pause
subplot(1,1,1)
plot3(y(:,1),y(:,2),y(:,2))
xlabel('x');ylabel('y');zlabel('z');grid
```

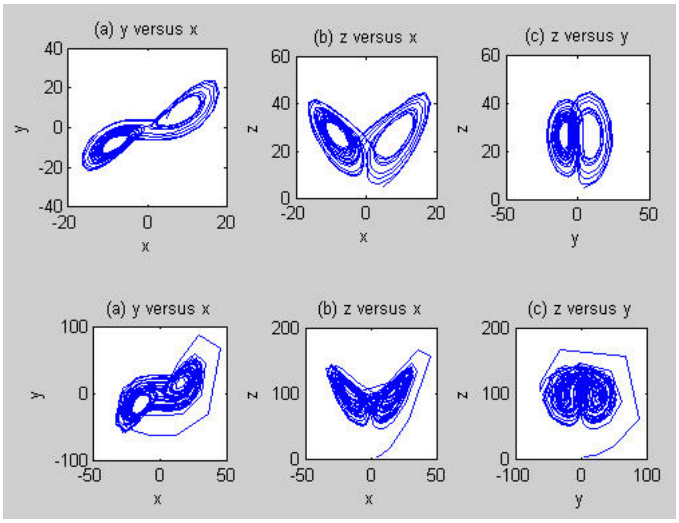


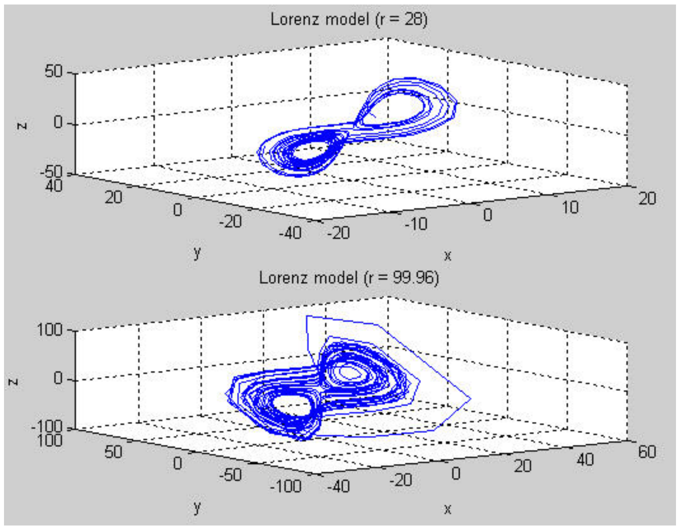


20.24 The following script generates the solutions and the plots

```
%Time-series plots
h=0.03125;tspan=[0 20];
sigma=10;b=8/3;r=28;
y0=[5 5 5];
[t1 y1] = rk4sys(@lorenz,tspan,y0,h,sigma,b,r);
y0=[5.001 5 5];
[tp1 yp1] = rk4sys(@lorenz,tspan,y0,h,sigma,b,r);
subplot(2,1,1);plot(t1,y1(:,1),tp1,yp1(:,1),'--')
title('Lorenz model (r = 28)');legend('x=y=z=5','x=5.001,y=z=5')
sigma=10;b=8/3;r=99.96;
y0=[5 5 5];
[t2 y2] = rk4sys(@lorenz,tspan,y0,h,sigma,b,r);
y0=[5.001 5 5];
[tp2 yp2] = rk4sys(@lorenz,tspan,y0,h,sigma,b,r);
subplot(2,1,2);plot(t2,y2(:,1),tp2,yp2(:,1),'--')
title('Lorenz model (r = 99.96)');legend('x=y=z=5','x=5.001,y=z=5')
%Phase plane plots
pause
subplot(2,3,1);plot(y1(:,1),y1(:,2))
xlabel('x');ylabel('y')
axis square;title('(a) y versus x')
subplot(2,3,2);plot(y1(:,1),y1(:,3))
xlabel('x');ylabel('z')
axis square;title('(b) z versus x')
subplot(2,3,3);plot(y1(:,2),y1(:,3))
xlabel('y');ylabel('z')
axis square;title('(c) z versus y')
subplot(2,3,4);plot(y2(:,1),y2(:,2))
xlabel('x');ylabel('y')
axis square;title('(a) y versus x')
subplot(2,3,5);plot(y2(:,1),y2(:,3))
xlabel('x');ylabel('z')
axis square;title('(b) z versus x')
subplot(2,3,6);plot(y2(:,2),y2(:,3))
xlabel('y');ylabel('z')
axis square;title('(c) z versus y')
%3D phase plane plots
pause
subplot(2,1,1);plot3(y1(:,1),y1(:,2),y1(:,2))
title('Lorenz model (r = 28)')
xlabel('x');ylabel('y');zlabel('z');grid
subplot(2,1,2);plot3(y2(:,1),y2(:,2),y2(:,2))
title('Lorenz model (r = 99.96)')
xlabel('x');ylabel('y');zlabel('z');grid
```


Lorenz model (r = 28)
x=y=z=5
x=5.001,y=z=5
Lorenz model (r = 99.96)
x=y=z=5
x=5.001,y=z=5


(a) y versus x
(b) z versus x
(c) z versus y
(a) y versus x
(b) z versus x
(c) z versus y


Lorenz model (r = 28)
Lorenz model (r = 99.96)