

# Systems of IVP's and Higher-Order Equations

## Systems of IVP's

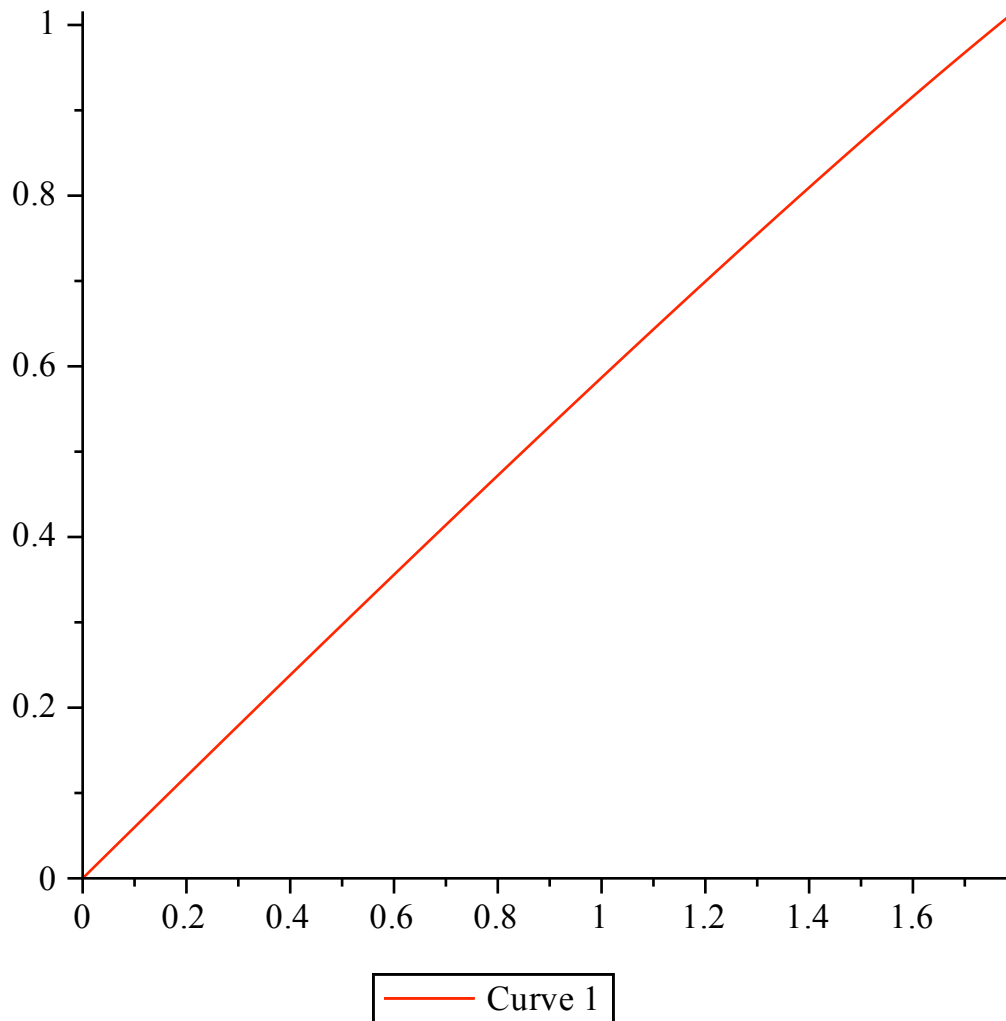
We first investigate a system of ODE's with initial conditions.

$$\begin{aligned}\frac{\partial}{\partial t} u_2 &= -4 u_1 + 3 u_2 + 6 \\ \frac{\partial}{\partial t} u_1 &= -2.4 u_1 + 1.6 u_2 + 3.6 \\ u_1(0) &= 0, u_2(0) = 0, t=0..0.5\end{aligned}$$

```
> restart;
> with(plots):
> deq:={D(u[1])(t)=-4*u[1](t)+3*u[2](t)+6,D(u[2])(t)=-2.4*u[1](t)
+1.6*u[2](t)+3.6};
    deq := {D(u1)(t) = -4 u1(t) + 3 u2(t) + 6, D(u2)(t) = -2.4 u1(t) + 1.6 u2(t) + 3.6}
> init:={u[1](0)=0,u[2](0)=0};
    init := {u1(0) = 0, u2(0) = 0}
```

Let's begin with the exact solution and express the results as functions.

```
> soln:=dsolve(deq union init,{u[1](t),u[2](t)});
    soln := {u1(t) =  $\frac{15}{8} e^{-\frac{2}{5}t} - \frac{27}{8} e^{-2t} + \frac{3}{2}$ , u2(t) =  $\frac{9}{4} e^{-\frac{2}{5}t} - \frac{9}{4} e^{-2t}$ }
> U[1]:=unapply(evalf(eval(u[1](t),soln)),t);
    U1 := t → 1.875000000 e-0.4000000000 t - 3.375000000 e-2. t + 1.500000000
> U[2]:=unapply(evalf(eval(u[2](t),soln)),t);
    U2 := t → 2.250000000 e-0.4000000000 t - 2.250000000 e-2. t
> p0:=plot([U[1](t),U[2](t),t=0..0.5],color=red):
display(p0);
```



Now let's apply Euler's method.

```
> eul:=dsolve(deq union init,{u[1](t),u[2](t)}, type=numeric,
method=classical[foreuler], start=0.0,stepsize=.1);
eul:=proc(x_classical) ... end proc
```

Let's check the output for **eul**.

```
> for i from 0 to 5 do
eul(i*.1);
od;
```

[ $t = 0., u_1(t) = 0., u_2(t) = 0.$ ]

[ $t = 0.1, u_1(t) = 0.600000000000000089, u_2(t) = 0.360000000000000042$ ]

[ $t = 0.2, u_1(t) = 1.068000000000000006, u_2(t) = 0.633600000000000052$ ]

[ $t = 0.3, u_1(t) = 1.430879999999999993, u_2(t) = 0.838656000000000068$ ]

[ $t = 0.4, u_1(t) = 1.710124800000000000, u_2(t) = 0.989429760000000047$ ]

[ $t = 0.5, u_1(t) = 1.922903808000000005, u_2(t) = 1.097308569600000001$ ]

The order for  $u_1$  and  $u_2$  is not always consistent, so we use eval in printing our tables.

```
> printf("  t      u[1]      eul[1]  |u[1]-eul[1]|  u[2]
eul[2]  |u[2]-eul[2]|\\n");
```

```

printf("\n");
for i from 0 to 5 do
printf("%4.1f %10.7f %10.7f %10.7f %10.7f %10.7f %10.7f\n",
.1*i,U[1](.1*i),eval(u[1](t),eul(.1*i)),abs(U[1](.1*i)-eval(u[1]
(t),eul(.1*i))),U[2](.1*i),eval(u[2](t),eul(.1*i)),abs(U[2](.1*i)
-eval(u[2](t),eul(.1*i))));
od;
t      u[1]      eul[1]      |u[1]-eul[1]|      u[2]      eul[2]      |u[2]
-eul[2]|
0.0    0.000000    0.000000    0.000000    0.000000    0.000000
0.000000
0.1    0.5382639  0.600000    0.0617361  0.3196320  0.360000
0.0403680
0.2    0.9685130  1.068000    0.0994870  0.5687917  0.633600
0.0648083
0.3    1.3107365  1.430880    0.1201435  0.7607448  0.8386560
0.0779112
0.4    1.5812844  1.7101248  0.1288404  0.9063334  0.9894298
0.0830964
0.5    1.7935270  1.9229038  0.1293768  1.0144155  1.0973086
0.0828931

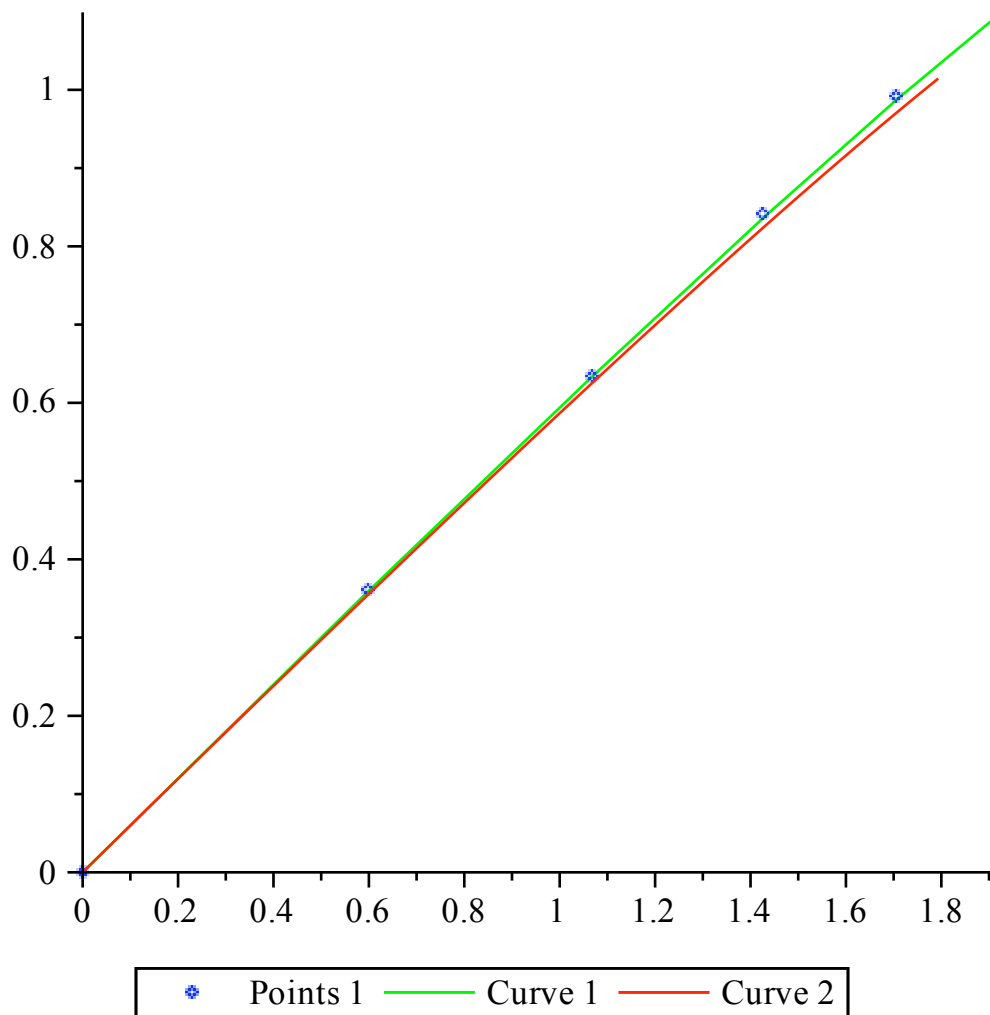
```

We add the graph of the numeric solution to that of the actual solution.

```

> p1:=odeplot(eul, [u[1](t),u[2](t)],0..(.5),color=green):
> p1:=pointplot([seq(eval([u[1](t),u[2](t)],eul(.1*i)),i=0..5)],
style=point,color=blue):
p2:=pointplot([seq(eval([u[1](t),u[2](t)],eul(.1*i)),i=0..5)],
style=line,color=green):
> display({p0,p1,p2});

```



Taylor method with absolute error less than or equal to  $10^{-6}$ .

```
> tay:=dsolve(deq union init,{u[1](t),u[2](t)}, type=numeric,
             method=taylorseries,abserr = Float(1,-6));
             tay := proc(x_taylorseries) ... end proc
```

```
> printf("  t      u[1]      tay[1]  |u[1]-tay[1]|  u[2]
tay[2]  |u[2]-tay[2]|\n");
printf("\n");
for i from 0 to 5 do
printf("%4.1f  %10.7f  %10.7f  %10.7f  %10.7f  %10.7f  %10.7f\n",
.1*i,U[1](.1*i),eval(u[1](t),tay(.1*i)),abs(U[1](.1*i)-eval(u[1]
(t),tay(.1*i))),U[2](.1*i),eval(u[2](t),tay(.1*i)),abs(U[2](.1*i)
-eval(u[2](t),tay(.1*i))));
od;
```

t	u[1]	tay[1]	u[1]-tay[1]	u[2]	tay[2]	u[2]-tay[2]
0.0	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
0.1	0.5382639	0.5382639	0.0000000	0.3196320	0.3196320	0.0000000
0.2	0.9685130	0.9685130	0.0000000	0.5687917	0.5687917	0.0000000
0.3	1.3107365	1.3107365	0.0000000	0.7607448	0.7607448	0.0000000

```

0.4 1.5812844 1.5812844 0.0000000 0.9063334 0.9063334
0.0000000
0.5 1.7935270 1.7935270 0.0000000 1.0144155 1.0144155
0.0000000

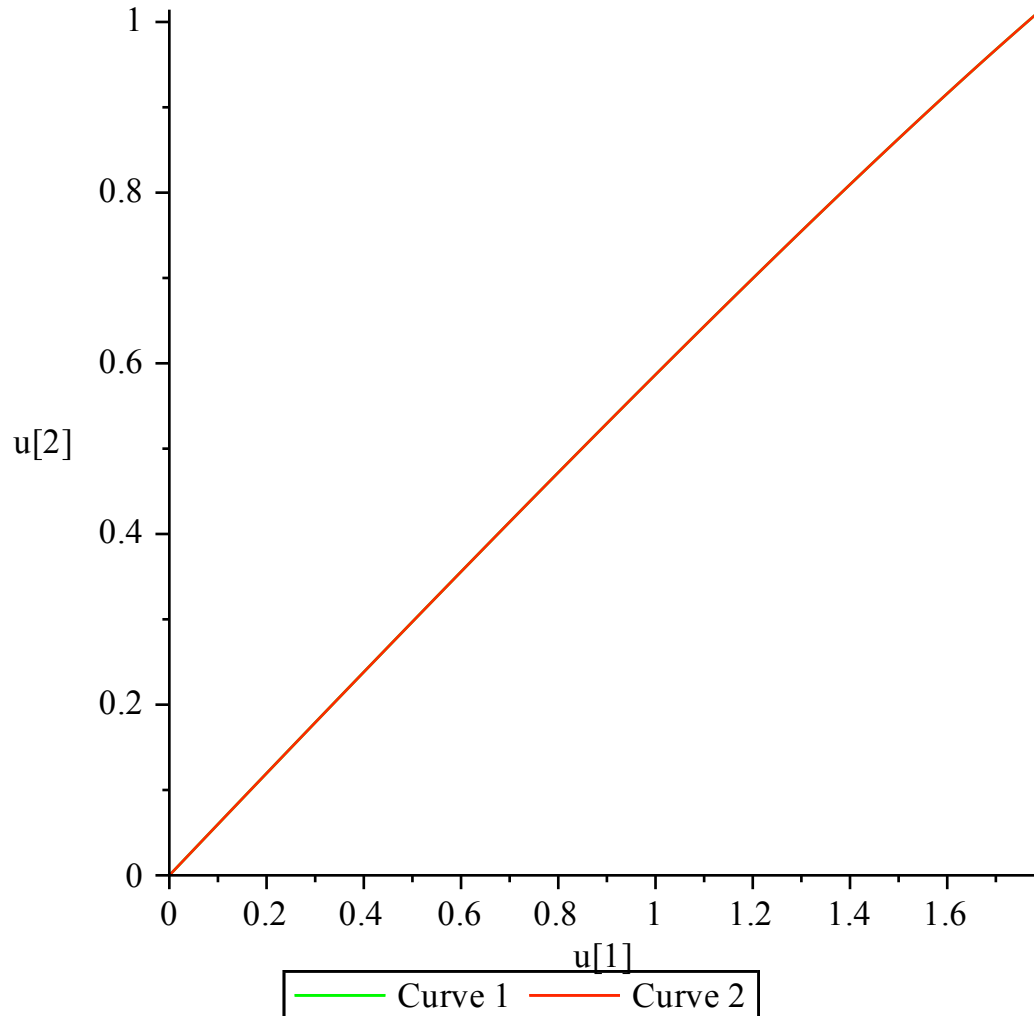
```

We plot the Taylor solution in green in `odeplot` using `with` with the actual solution in red.

```

> p3:=odeplot(tay, [u[1](t), u[2](t)], 0..(.5), color=green):
> display({p0, p3});

```



A pretty good match. Next we use the Runge-Kutta method of order 4.

```

> rk:=dsolve(deq union init, {u[1](t), u[2](t)}, type=numeric,
method=classical[rk4], start=0.0, stepsize=.1);
rk:=proc(x_classical) ... end proc

> printf(" t u[1] rk[1] |u[1]-rk[1]| u[2]
rk[2] |u[2]-rk[2]| \n");
printf("\n");
for i from 0 to 5 do
printf("%4.1f %10.7f %10.7f %10.7f %10.7f %10.7f %10.7f \n",
.1*i, U[1](.1*i), eval(u[1](t), rk(.1*i)), abs(U[1](.1*i)-eval(u[1]
(t), rk(.1*i))), U[2](.1*i), eval(u[2](t), rk(.1*i)), abs(U[2](.1*i)-
eval(u[2](t), rk(.1*i))));
od;
t u[1] rk[1] |u[1]-rk[1]| u[2] rk[2] |u[2]
-rk[2]|

```

```

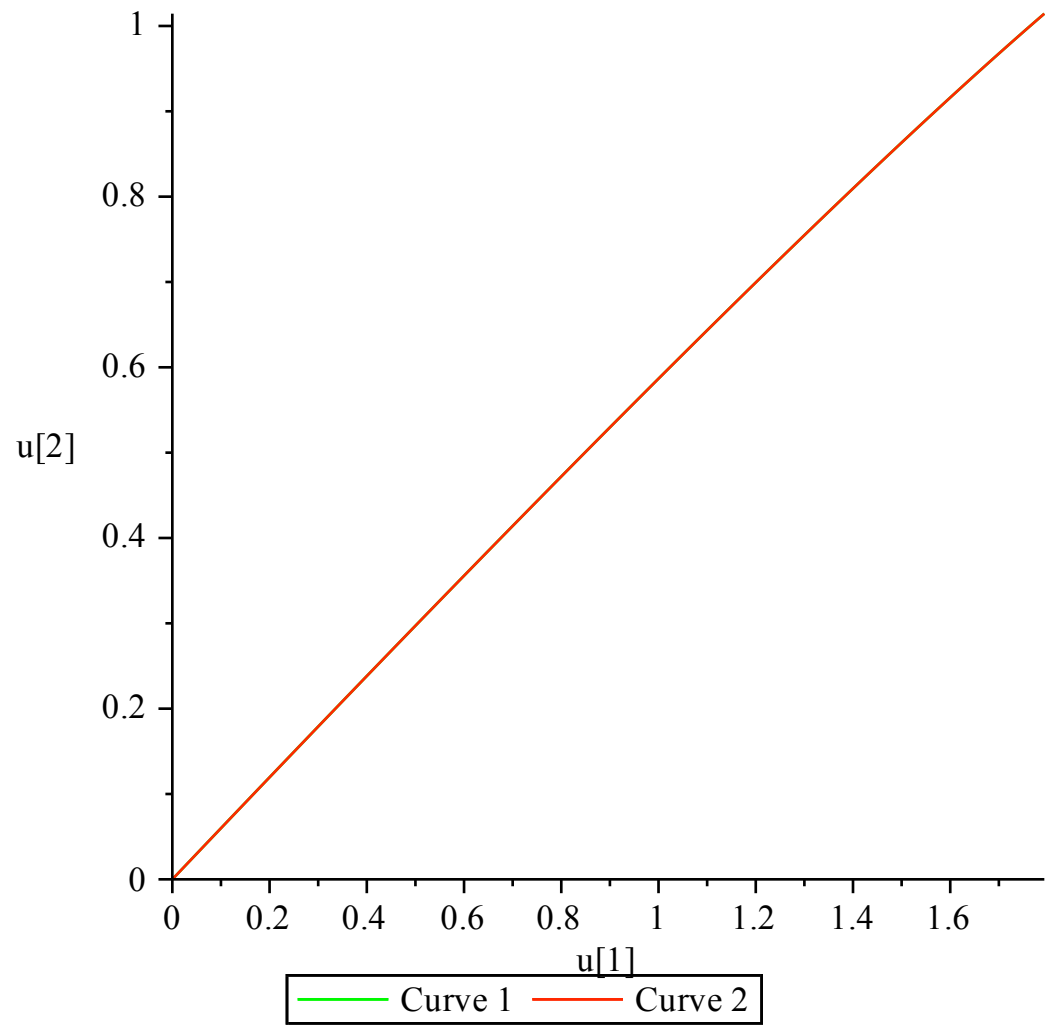
0.0 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000
0.1 0.5382639 0.5382552 0.0000087 0.3196320 0.3196262
0.0000058
0.2 0.9685130 0.9684987 0.0000143 0.5687917 0.5687822
0.0000095
0.3 1.3107365 1.3107190 0.0000175 0.7607448 0.7607331
0.0000117
0.4 1.5812844 1.5812652 0.0000191 0.9063334 0.9063206
0.0000127
0.5 1.7935270 1.7935075 0.0000196 1.0144155 1.0144024
0.0000130

```

```

> p4:=odeplot(rk,[u[1](t),u[2](t)],0..(.5),color=green):
> display({p0,p4});

```



Another good match.

## Higher-Order Equations

We consider the higher order equation

$$t^3 y''' - t^2 y'' + 3ty' - 4y = 5t^3 \ln(t) + 9t^3, t=1..2,$$

with initial conditions

$$y(1) = 0, y'(1) = 1, y''(1) = 3.$$

We use  $h = 0.1$ .

```
> restart;with(plots):
> deq:=t^3*(D@@3)(y)(t)-t^2*(D@@2)(y)(t)+3*t*D(y)(t)-4*y(t)=5*t^3*ln(t)+9*t^3;
      deq:=t^3 D^(3)(y)(t) - t^2 D^(2)(y)(t) + 3 t D(y)(t) - 4 y(t) = 5 t^3 ln(t) + 9 t^3
> init:=y(1)=0,D(y)(1)=1,(D@@2)(y)(1)=3;
      init:=y(1) = 0, D(y)(1) = 1, D^(2)(y)(1) = 3
```

We attempt to find the exact solution.

```
> soln:=dsolve({deq, init},y(t));
soln := y(t) = 1/4 * 1/t (t^3 (10 t ln(t) + 8 t - 4 cos(ln(t)) (t^1)^2 + I cos(ln(t)) ln(t)
- 3 cos(ln(t)) (t^1)^2 ln(t) - I cos(ln(t)) (t^1)^2 ln(t) - 4 cos(ln(t)) - 4 I sin(ln(t))
- 3 cos(ln(t)) ln(t) + 4 I sin(ln(t)) (t^1)^2 - 2 sin(ln(t)) (t^1)^2 + 3 I sin(ln(t)) (t^1)^2 ln(t)
- sin(ln(t)) (t^1)^2 ln(t) - 2 I cos(ln(t)) (t^1)^2 - 2 sin(ln(t)) + 2 I cos(ln(t))
- sin(ln(t)) ln(t) - 3 I sin(ln(t)) ln(t)) - t^2 + t cos(ln(t)) + t sin(ln(t))
```

Maple gives us a solution in terms of complex numbers. However, the solution is known using real numbers, so we enter it as a function.

```
> Y:=t->-t^2+t*cos(ln(t))+t*sin(ln(t))+t^3*ln(t);;
      Y:=t -> -t^2 + t cos(ln(t)) + t sin(ln(t)) + t^3 ln(t)
> p0:=plot(Y(t),t=1..2,color=red):
```

Let's use the Runge-Kutta method of order 4 as illustrative.

```
> rk:=dsolve({deq, init}, y(t), type=numeric,
      method=classical[rk4], start=1.0,stepsize=.1);
      rk:=proc(x_classical) ... end proc
```

Let's see what the basic output for rk is.

```
> for i from 0 to 10 do
  rk(1+i*.1);
od;
```

$$\left[ t = 1., y(t) = 0., \frac{d}{dt} y(t) = 1., \frac{d^2}{dt^2} y(t) = 3. \right]$$

$$\left[ t = 1.1, y(t) = 0.116547765077132395, \frac{d}{dt} y(t) = 1.34689724469854166, \frac{d^2}{dt^2} y(t) = 3.95601046915795163 \right]$$

$$\left[ t = 1.2, y(t) = 0.272737593417177349, \frac{d}{dt} y(t) = 1.79447640084850191, \frac{d^2}{dt^2} y(t) = 5.01051249411298372 \right]$$

$$\left[ t = 1.3, y(t) = 0.479101055922199870, \frac{d}{dt} y(t) = 2.35173979065363303, \frac{d^2}{dt^2} y(t) = 6.14739936557764466 \right]$$

$$\left[ t = 1.4, y(t) = 0.746997034090162493, \frac{d}{dt} y(t) = 3.02629852140462585, \frac{d^2}{dt^2} y(t) = 7.35468715506850490 \right]$$

$$\left[ t = 1.5, y(t) = 1.08849079479831268, \frac{d}{dt} y(t) = 3.82471572721752295, \frac{d^2}{dt^2} y(t) = 8.62323066151186168 \right]$$

$$\left[ t = 1.6, y(t) = 1.51626183931491254, \frac{d}{dt} y(t) = 4.75274601865115809, \frac{d^2}{dt^2} y(t) = 9.94589311929556530 \right]$$

$$\left[ t = 1.7, y(t) = 2.04353207184545216, \frac{d}{dt} y(t) = 5.81550687582345383, \frac{d^2}{dt^2} y(t) = 11.3169933791159245 \right]$$

$$\left[ t = 1.8, y(t) = 2.68400867195246784, \frac{d}{dt} y(t) = 7.01760416361247419, \frac{d^2}{dt^2} y(t) = 12.7319281671128923 \right]$$

$$\left[ t = 1.9, y(t) = 3.45183784122406845, \frac{d}{dt} y(t) = 8.36322594900899219, \frac{d^2}{dt^2} y(t) = 14.1869079328708363 \right]$$

$$\left[ t = 2.0, y(t) = 4.36156675051770382, \frac{d}{dt} y(t) = 9.85621392990919887, \frac{d^2}{dt^2} y(t) = 15.6787682487629496 \right]$$

We print a table for  $y$ .

```
> printf("    t          y          rk          |y-")
```

```

printf("\n");
for i from 0 to 10 do
printf("%4.1f    %12.8f    %12.8f    %12.8f\n",1+.1*i,Y(1+.1*
i),eval(y(t),rk(1+.1*i)),abs(Y(1+.1*i)-eval(y(t),rk(1+.1*i))));
od;

```

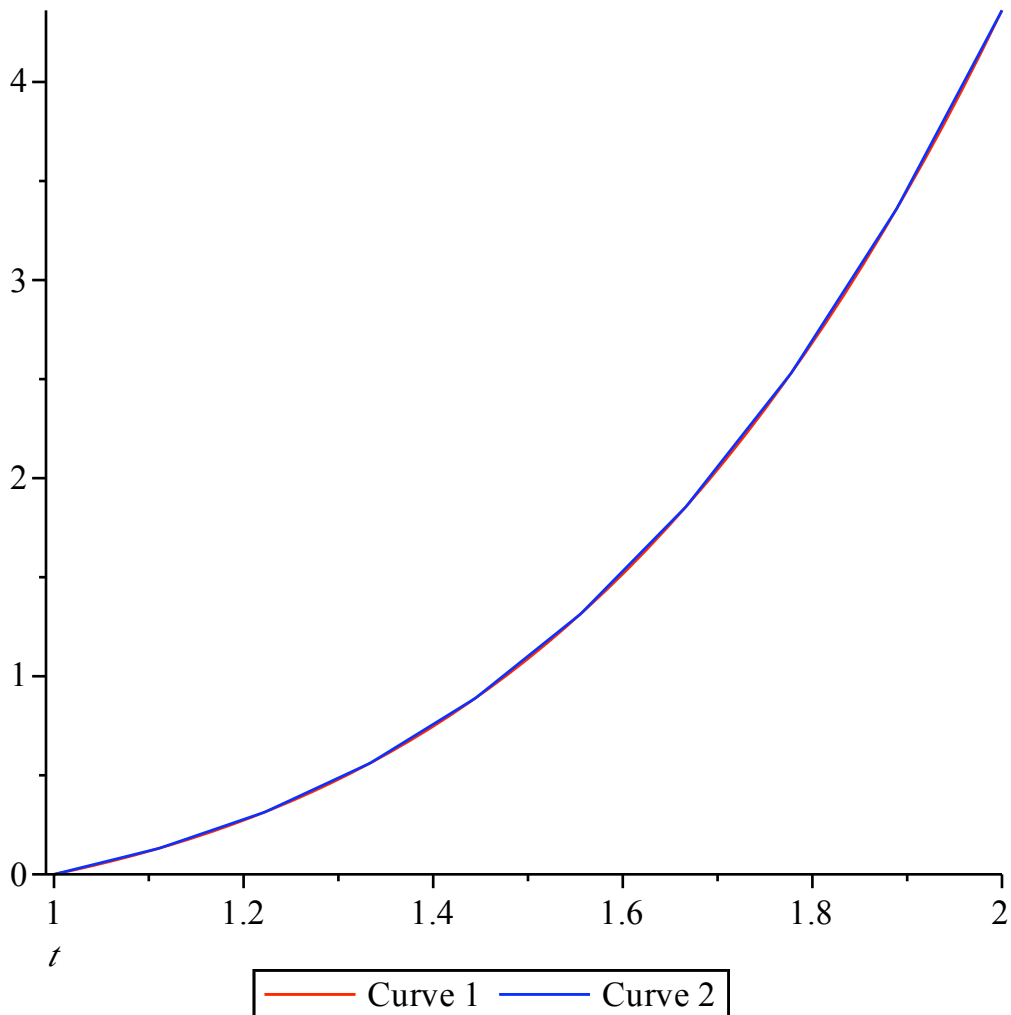
t	y	rk	y-rk
1.0	0.00000000	0.00000000	0.00000000
1.1	0.11654795	0.11654777	0.00000019
1.2	0.27273791	0.27273759	0.00000032
1.3	0.47910162	0.47910106	0.00000057
1.4	0.74699807	0.74699703	0.00000104
1.5	1.08849260	1.08849079	0.00000180
1.6	1.51626473	1.51626184	0.00000289
1.7	2.04353642	2.04353207	0.00000434
1.8	2.68401485	2.68400867	0.00000618
1.9	3.45184625	3.45183784	0.00000841
2.0	4.36157780	4.36156675	0.00001105

We look at the graphs of  $y(t)$  and the Runge-Kutta approximation.

```

> p1:=odeplot(rk,[t,y(t)],1..2, numpoints=10,color=blue):
> display(p0,p1);

```



We can also print a table for  $y'$ . For comparison, we first compute the derivative of the exact solution.

```

> Y1:=D(Y);

```

$$Y1 := t \rightarrow -2t + 2 \cos(\ln(t)) + 3t^2 \ln(t) + t^2$$

```

> printf("  t          y'          rk          |y'-rk|
rk|\n");
printf("\n");
for i from 0 to 10 do
printf("%4.1f    %12.8f    %12.8f    %12.8f\n", 1+.1*i, Y1
(1+.1*i), eval(diff(y(t), t), rk(1+.1*i)), abs(Y1(1+.1*i)-eval(diff(y
(t), t), rk(1+.1*i))));
od;

```

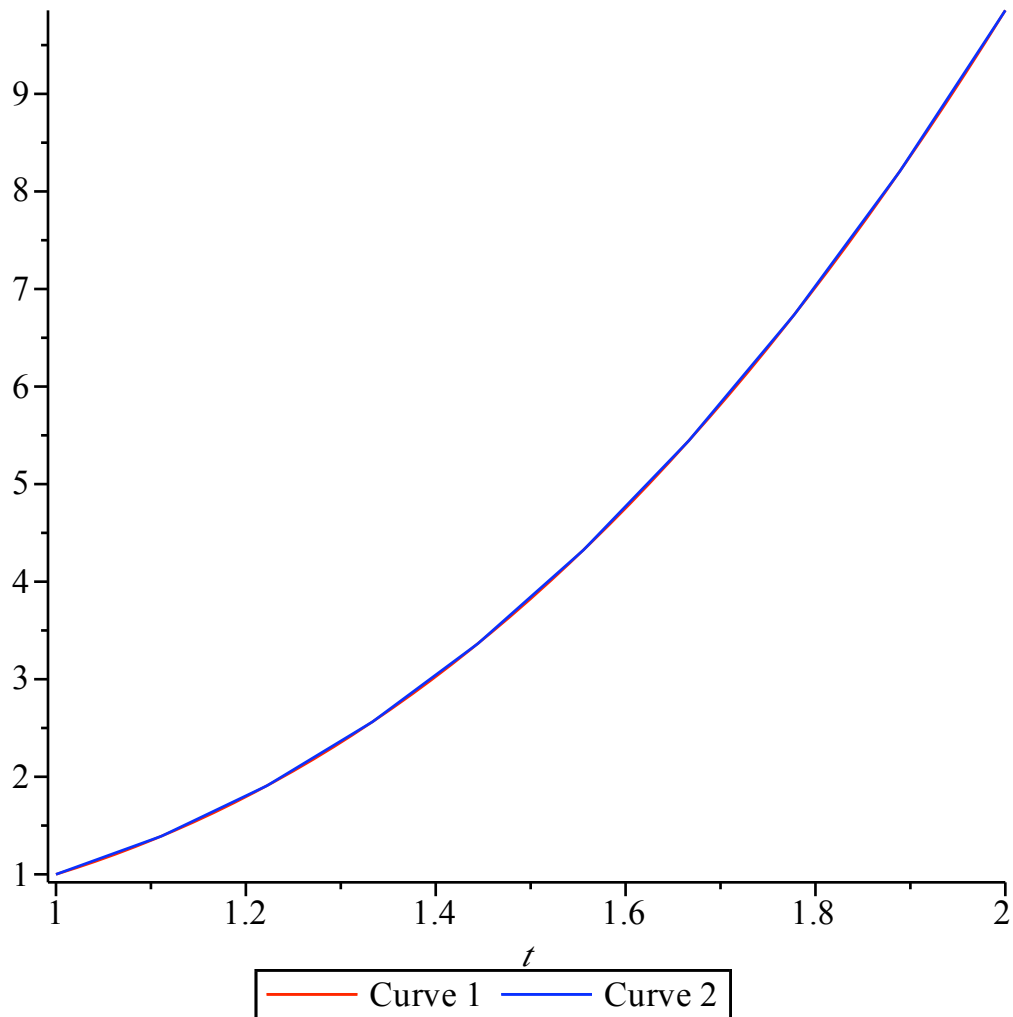
t	y'	rk	y'-rk
1.0	1.00000000	1.00000000	0.00000000
1.1	1.34689880	1.34689724	0.00000155
1.2	1.79447995	1.79447640	0.00000355
1.3	2.35174576	2.35173979	0.00000597
1.4	3.02630727	3.02629852	0.00000875
1.5	3.82472755	3.82471573	0.00001183
1.6	4.75276116	4.75274602	0.00001514
1.7	5.81552554	5.81550688	0.00001866
1.8	7.01762649	7.01760416	0.00002233
1.9	8.36325208	8.36322595	0.00002613
2.0	9.85624397	9.85621393	0.00003004

We look at the derivative plots.

```

> p2:=plot(Y1(t), t=1..2, color=red):
p3:=odeplot(rk, [t, diff(y(t), t)], 1..2, numpoints=10, color=blue):
> display(p2, p3);

```



Finally, we print a table for  $y''$ .

```
> Y2 := (D@@2)(Y);
```

$$Y2 := t \rightarrow -2 - \frac{2 \sin(\ln(t))}{t} + 6 t \ln(t) + 5 t$$

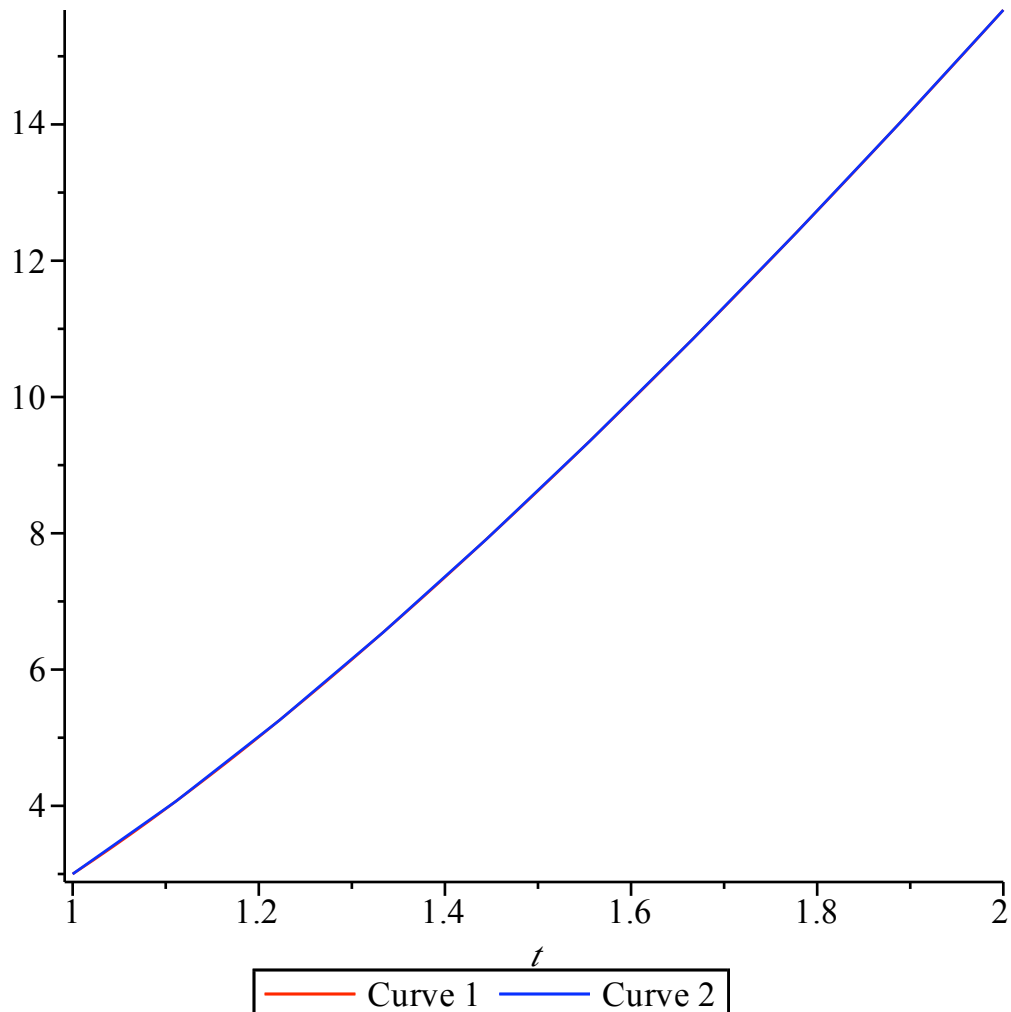
```
> printf("  t          y''          rk          |y''-rk|
rk|\n");
printf("\n");
for i from 0 to 10 do
printf("%4.1f    %12.8f    %12.8f    %12.8f\n", 1+.1*i, Y2
(1+.1*i), eval(diff(y(t), t$2), rk(1+.1*i)), abs(Y2(1+.1*i)-eval(diff
(y(t), t$2), rk(1+.1*i)))));
od;
```

t	y''	rk	y''-rk
1.0	3.00000000	3.00000000	0.00000000
1.1	3.95601820	3.95601047	0.00000773
1.2	5.01052665	5.01051249	0.00001415
1.3	6.14741875	6.14739937	0.00001938
1.4	7.35471078	7.35468716	0.00002362
1.5	8.62325771	8.62323066	0.00002704
1.6	9.94592294	9.94589312	0.00002982
1.7	11.31702545	11.31699338	0.00003207
1.8	12.73196208	12.73192817	0.00003391

1.9	14.18694335	14.18690793	0.00003542
2.0	15.67880489	15.67876825	0.00003664

We look at the graphs here also.

```
> p4:=plot(Y2(t),t=1..2,color=red):
> p5:=odeplot(rk,[t,diff(y(t),t$2)],1..2, numpoints=10,color=blue):
> display(p4,p5);
```



## Predator-Prey (page233 # 5)

The predator-prey system of equations

```
> deq:={D(x[1])(t)=3*x[1](t)-.002*x[1](t)*x[2](t),D(x[2])(t)=.0006*
x[1](t)*x[2](t)-.5*x[2](t)};
deq := {D(x1)(t) = 3 x1(t) - 0.002 x1(t) x2(t), D(x2)(t) = 0.0006 x1(t) x2(t) - 0.5 x2(t)}
> init:={x[1](0)=1000,x[2](0)=500};
init := {x1(0) = 1000, x2(0) = 500}
```

The numerical solution using a fourth order Runge-Kutta method.

```
> rk:=dsolve(deq union init,{x[1](t),x[2](t)}, type=numeric,
method=classical[rk4], start=0.0,stepsize=.1);
rk:=proc(x_classical) ... end proc
```

A table for the solution.

```

> printf("t (years)          prey          predators\n");
printf("\n");
for i from 0 to 40 do
printf("%4.1f          %10.0f          %10.0f\n", .1*i, eval(x[1](t), rk
(.1*i)), eval(x[2](t), rk(.1*i)));
od;

```

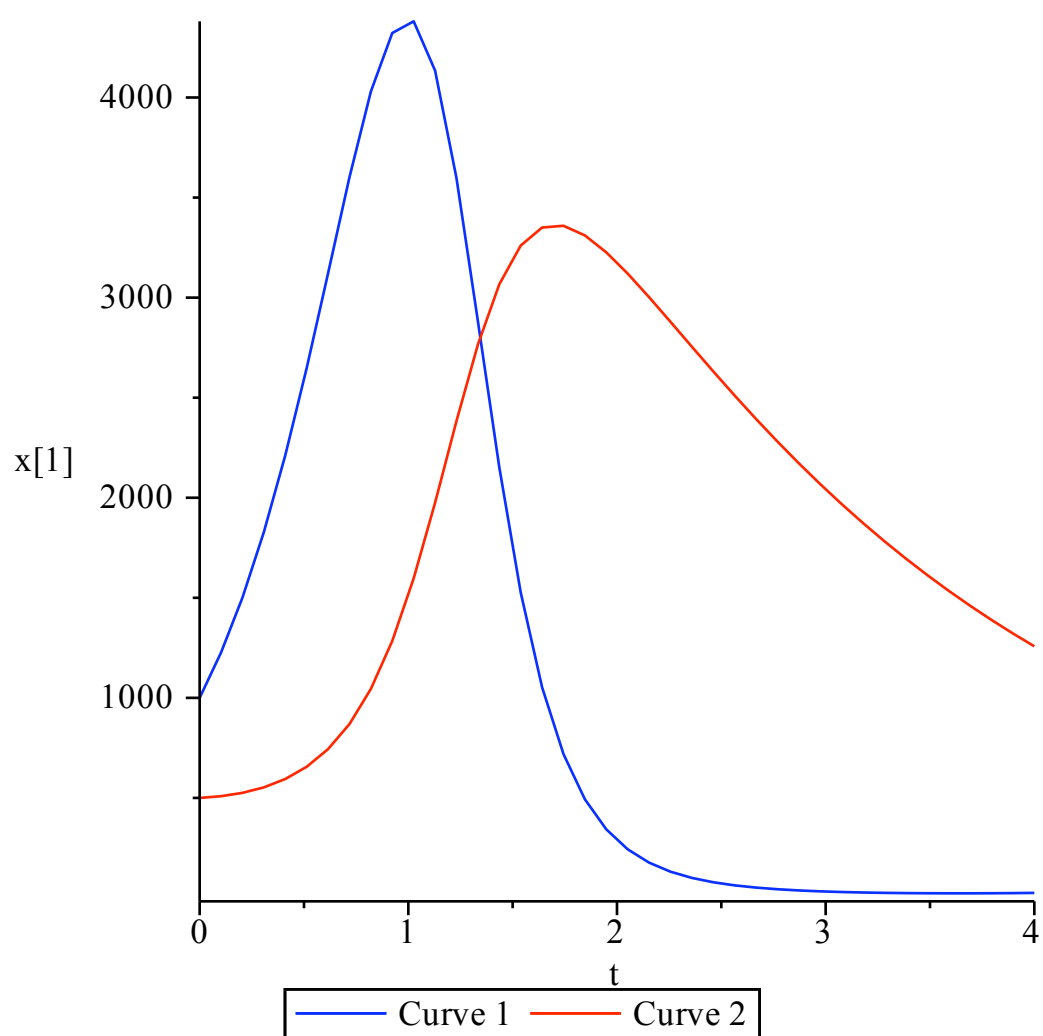
t (years)	prey	predators
0.0	1000	500
0.1	1221	508
0.2	1486	524
0.3	1802	550
0.4	2171	590
0.5	2591	647
0.6	3049	728
0.7	3519	844
0.8	3952	1005
0.9	4274	1224
1.0	4393	1512
1.1	4234	1866
1.2	3784	2261
1.3	3125	2648
1.4	2401	2973
1.5	1745	3200
1.6	1225	3325
1.7	846	3364
1.8	584	3338
1.9	407	3270
2.0	288	3175
2.1	208	3065
2.2	154	2947
2.3	117	2826
2.4	91	2705
2.5	72	2586
2.6	59	2469
2.7	49	2356
2.8	42	2247
2.9	36	2143
3.0	32	2042
3.1	29	1946
3.2	27	1855
3.3	25	1767
3.4	24	1683
3.5	24	1603
3.6	23	1527
3.7	23	1455
3.8	24	1386
3.9	24	1320
4.0	25	1258

We graph the solutions over the four year cycle. The predator graph is red and the prey graph is blue.

```

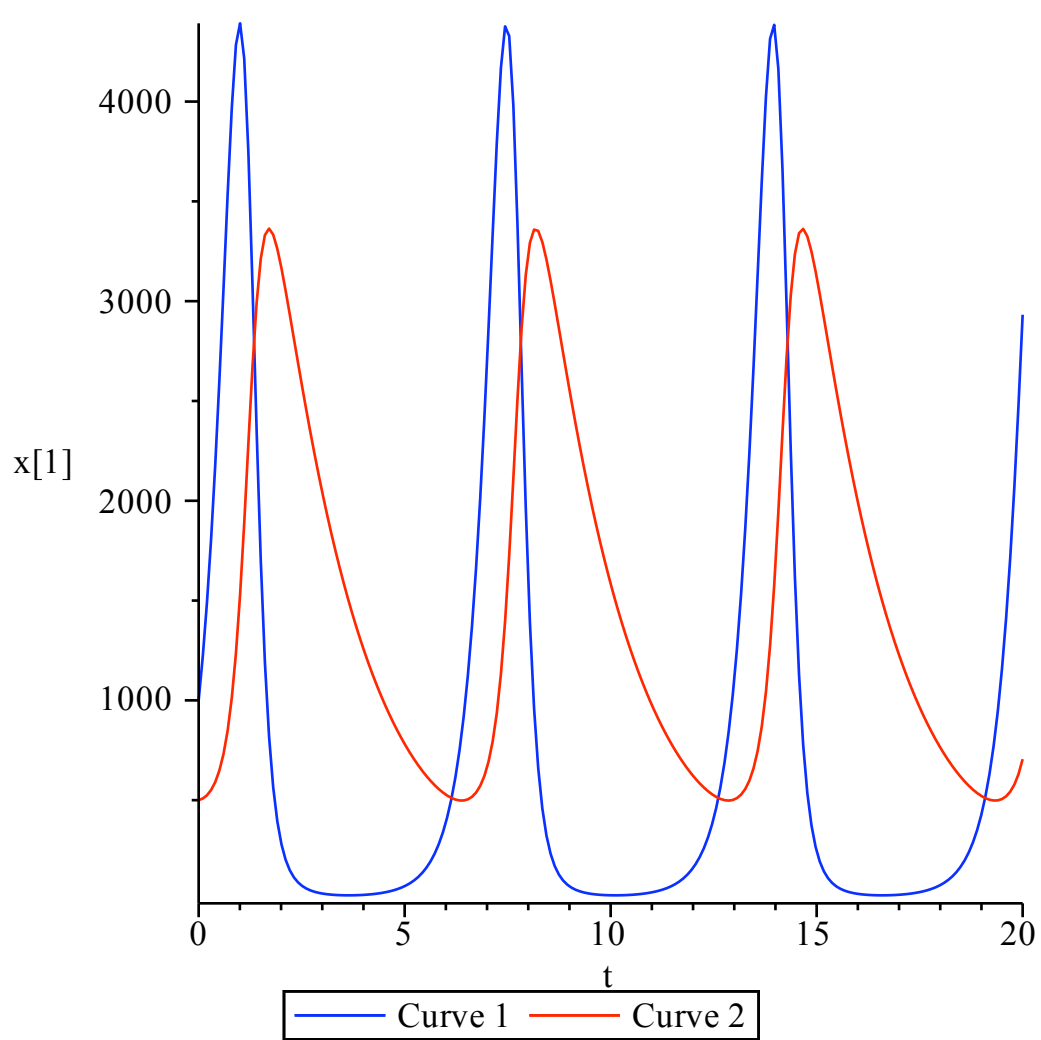
> p1:=odeplot(rk, [t, x[1](t)], 0..4, numpoints=40, color=blue):
p2:=odeplot(rk, [t, x[2](t)], 0..4, numpoints=40, color=red):
display(p1, p2);

```



Let's see what happens when we extend the time line to 20 years.

```
> p3:=odeplot(rk, [t,x[1](t)],0..20,numpoints=200,color=blue):  
p4:=odeplot(rk, [t,x[2](t)],0..20,numpoints=200,color=red):  
display(p3,p4);
```



We see that there is no stable solution to this problem. Rather, both populations cycle periodically.