CHAPTER 26

26.1 (a) $h < 2/100{,}000 = 2\times10^{-5}$.

(b) The implicit Euler can be written for this problem as

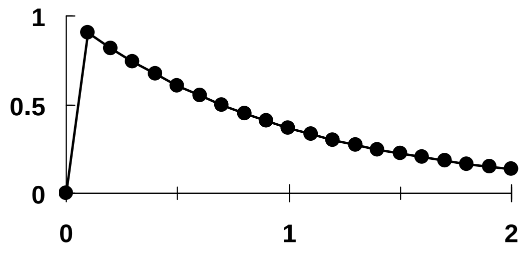
$$y_{i+1} = y_i + \left(-100{,}000 y_{i+1} + 99{,}999 e^{-t_{i+1}}\right) h$$

which can be solved for

$$y_{i+1} = \frac{y_i + 99{,}999 e^{-t_{i+1}} h}{1 + 100{,}000 h}$$

The results of applying this formula for the first few steps are shown below. A plot of the entire solution is also displayed

t	*y*
0	0
0.1	0.904738
0.2	0.818731
0.3	0.740819
0.4	0.67032
0.5	0.606531



26.2 The implicit Euler can be written for this problem as

$$y_{i+1} = y_i + \left(30(\sin t_{i+1} - y_{i+1}) + 3\cos t_{i+1}\right) h$$

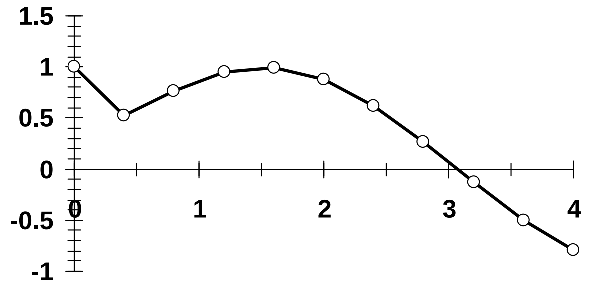
which can be solved for

$$y_{i+1} = \frac{y_i + 30\sin t_{i+1} h + 3\cos t_{i+1} h}{1 + 30h}$$

The results of applying this formula are tabulated and graphed below.

x	*y*
0	1
0.4	0.521407
0.8	0.766594
1.2	0.952761
1.6	0.993277
2	0.877344
2.4	0.622925

2.8	0.270163
3.2	-0.12525
3.6	-0.50089
4	-0.79745



26.3 (a) The explicit Euler can be written for this problem as

$$x_{1,i+1} = x_{1,i} + \left(999x_{1,i} + 1999x_{2,i}\right)h$$
$$x_{2,i+1} = x_{2,i} + \left(-1000x_{1,i} - 2000x_{2,i}\right)h$$

Because the step-size is much too large for the stability requirements, the solution is unstable,

t	x_1	x_2	dx_1/dt	dx_2/dt
0	1	1	2998	-3000
0.05	150.9	-149	-147102	147100
0.1	-7204.2	7206	7207803	-7207805
0.15	353186	-353184	-3.5E+08	3.53E+08
0.2	-1.7E+07	17305943	1.73E+10	-1.7E+10

(b) The implicit Euler can be written for this problem as

$$x_{1,i+1} = x_{1,i} + \left(999x_{1,i+1} + 1999x_{2,i+1}\right)h$$
$$x_{2,i+1} = x_{2,i} + \left(-1000x_{1,i+1} - 2000x_{2,i+1}\right)h$$

or collecting terms

$$(1-999h)x_{1,i+1} - 1999hx_{2,i+1} = x_{1,i}$$
$$1000hx_{1,i+1} + (1+2000h)x_{2,i+1} = x_{2,i}$$

or substituting h = 0.05 and expressing in matrix format

$$\begin{bmatrix} -48.95 & -99.95 \\ 50 & 101 \end{bmatrix} \begin{Bmatrix} x_{1,i+1} \\ x_{2,i+1} \end{Bmatrix} = \begin{Bmatrix} x_{1,i} \\ x_{2,i} \end{Bmatrix}$$

Thus, to solve for the first time step, we substitute the initial conditions for the right-hand side and solve the 2×2 system of equations. The best way to do this is with *LU* decomposition since we will have to solve the system repeatedly. For the present case, because it's easier to display, we will use the matrix inverse to obtain the solution. Thus, if the matrix is inverted, the solution for the first step amounts to the matrix multiplication,

$$\begin{Bmatrix} x_{1,i+1} \\ x_{2,i+1} \end{Bmatrix} = \begin{bmatrix} 1.886088 & 1.86648 \\ -0.93371 & -0.9141 \end{bmatrix} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix} = \begin{Bmatrix} 3.752568 \\ -1.84781 \end{Bmatrix}$$

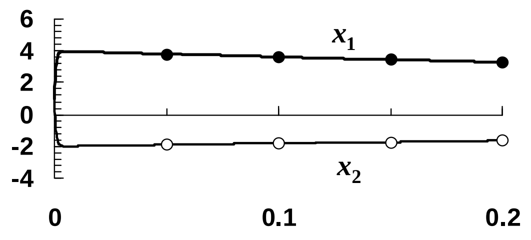
For the second step (from $x = 0.05$ to 0.1),

$$\begin{Bmatrix} x_{1,i+1} \\ x_{2,i+1} \end{Bmatrix} = \begin{bmatrix} 1.886088 & 1.86648 \\ -0.93371 & -0.9141 \end{bmatrix} \begin{Bmatrix} 3.752568 \\ -1.84781 \end{Bmatrix} = \begin{Bmatrix} 3.62878 \\ -1.81472 \end{Bmatrix}$$

The remaining steps can be implemented in a similar fashion to give

t	x_1	x_2
0	1	1
0.05	3.752568	-1.84781
0.1	3.62878	-1.81472
0.15	3.457057	-1.72938
0.2	3.292457	-1.64705

The results are plotted below, along with a solution with the explicit Euler using a step of 0.0005.



26.4 The analytical solution is

$$y = 12e^{-0.5t} - 2e^{-t}$$

Therefore, the exact results are $y(2.5) = 3.273888$ and $y(3) = 2.577988$.

<u>First step:</u>

Predictor:

$$y_1^0 = 5.222138 + [-0.5(4.143883) + e^{-2}]1 = 3.28553178$$

Corrector:

$$y_1^1 = 4.143883 + \frac{-0.5(4.143883) + e^{-2} - 0.5(3.28553178) + e^{-2.5}}{2} 0.5 = 3.2695612$$

The corrector can be iterated to yield

j	y_{i+1}^j	$\lvert\varepsilon_a\rvert$, %
1	3.2695612	
2	3.2715575	0.0610

The true error can be computed as

$$\varepsilon_t = \left|\frac{3.273888 - 3.2715575}{3.273888}\right| \times 100\% = 0.071\%$$

<u>Second step:</u>

Predictor:

$$y_2^0 = 4.143883 + [-0.5(3.2715575) + e^{-2.5}]1 = 2.5901892$$

Predictor Modifier:

$$y_2^0 = 2.5901892 + 4/5(3.2715575 - 3.28553178) = 2.57900983$$

Corrector:

$$y_2^1 = 3.2715575 + \frac{-0.5(3.2715575) + e^{-2.5} - 0.5(2.57900983) + e^{-3}}{2} 0.5 = 2.5732046$$

The corrector can be iterated to yield

j	y_{i+1}^j	$\lvert\varepsilon_a\rvert$, %
1	2.5732046	
2	2.5739303	0.0282

The true error can be computed as

$$\varepsilon_t = \left|\frac{2.577988 - 2.5739303}{2.577988}\right| \times 100\% = 0.157\%$$

26.5 The analytical solution is

$$y = 12e^{-0.5t} - 2e^{-t}$$

Therefore, the exact results are $y(2.5) = 3.273888$ and $y(3) = 2.577988$.

The Adams method can be implemented with the results

```
predictor =      3.270674927
Corrector Iteration
x         y            ea
2.5       3.274330476  11.1643%
2.5       3.273987768  1.0468%
2.5       3.274019897  0.0981%

predictor =      2.576436209
Corrector Iteration
x         y            ea
3         2.57830404   7.2444%
3         2.578128931  0.6792%
```

Therefore, the true percent relative errors can be computed as

$$\varepsilon_t = \left|\frac{3.273888 - 3.274019897}{3.273888}\right| \times 100\% = 0.0098\%$$

$$\varepsilon_t = \left|\frac{2.577988 - 2.578128931}{2.577988}\right| \times 100\% = 0.0055\%$$

26.6 (a) Non-self-starting Heun

First step:

Predictor:

$y_1^0 = 0.244898 + [-2(0.1875)/4]1 = 0.151148$

Corrector:

$$y_1^1 = 0.1875 + \frac{-2(0.1875)/4 - 2(0.151148)/4.5}{2} 0.5 = 0.1472683$$

The corrector can be iterated to yield

j	***y*$_{i+1}^j$**	**|ε_a|, %**
1	0.1472683	2.63
2	0.1476994	0.292

The true error can be computed as

$$\varepsilon_t = \left|\frac{0.148148 - 0.1476994}{0.148148}\right| \times 100\% = 0.3\%$$

Second step:

Predictor:

$y_2^0 = 0.1875 + [-2(0.1476994)/4.5]1 = 0.1218558$

Predictor Modifier:

$y_2^0 = 0.1218558 + 4/5(0.1476994-0.151148) = 0.1190969$

Corrector:

$$y_2^1 = 0.1476994 + \frac{-2(0.1476994)/4.5 - 2(0.1190969)/5}{2} 0.5 = 0.1193786$$

The corrector can be iterated to yield

j	***y*$_{i+1}^j$**	**|ε_a|, %**
1	0.1193786	0.236

The true error can be computed as

$$\varepsilon_t = \left|\frac{0.12-0.1193786}{0.12}\right|\times 100\% = 0.52\%$$

(b) Adams method

```
predictor =      0.152793519
Corrector Iteration
x         y                   ea           et
4.5       0.147605464         3.51E+00     0.366%
4.5       0.148037802         2.92E-01     0.074%
4.5       0.148001774         2.43E-02     0.099%
4.5       0.148004776         0.002028547  0.097%

predictor =      0.121661277
Corrector Iteration
x         y                   ea           et
5         0.119692476         1.64E+00     0.256%
5         0.119840136         1.23E-01     0.133%
5         0.119829061         9.24E-03     0.142%
```

26.7 The 4th-order RK method can be used to compute $y(0.25) = 0.7828723$. Then, the non-self-starting Heun method can be implemented as

First step:

Predictor:

$y_1^0 = 1 + f(0.25, 0.7828723)(0.5) = 0.6330286$

Corrector:

$$y_1^1 = 0.7828723 + \frac{f(0.25, 0.7828723) + f(0.5, 0.6330286)}{2} 0.25 = 0.6317830$$

The corrector can be iterated to yield

j	y_{i+1}^j	$\lvert\varepsilon_a\rvert$, %
1	0.6317830	0.197
2	0.6318998	0.018
3	0.6318888	0.0017
4	0.6318898	0.00016
5	0.6318897	0.000015

Second step:

Predictor:

$y_2^0 = 0.7828723 + f(0.5, 0.6318897)(0.5) = 0.5458282$

Predictor Modifier:

$y_2^0 = 0.5458282 + 4/5(0.6318897 - 0.6330286) = 0.5449171$

Corrector:

$$y_2^1 = 0.6318897 + \frac{f(0.5, 0.6318897) + f(0.75, 0.5449171)}{2} 0.25 = 0.5430563$$

The corrector can be iterated to yield

j	y_{i+1}^j	$\lvert\varepsilon_a\rvert$, %
1	0.5430563	0.343
2	0.5431581	0.0187
3	0.5431525	0.001
4	0.5431528	0.000056

26.8 The fourth-order RK method can be used to determine the following values:

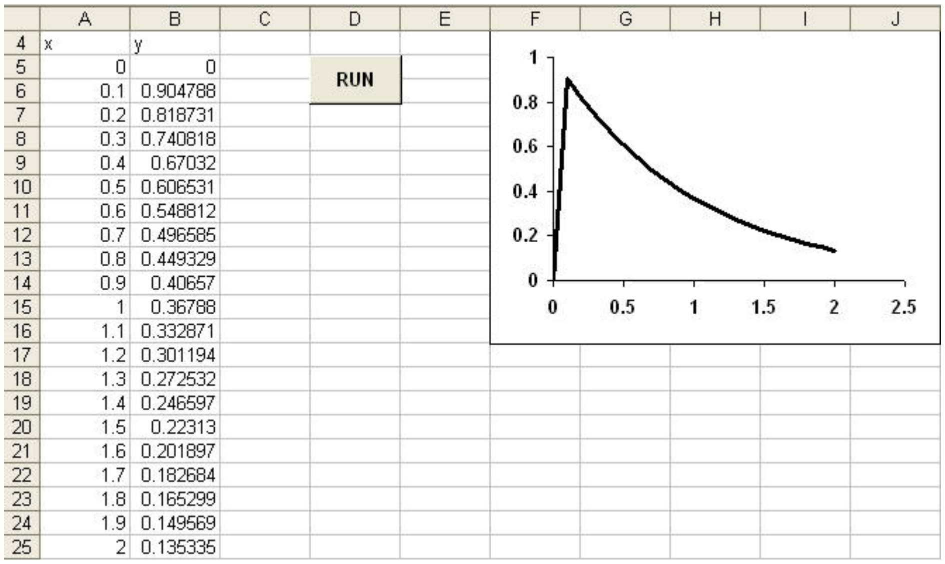
t	y
0	2
0.5	0.893333
1	0.50288
1.5	0.321905

The 4th-order Adams method can then be implemented with the results summarized below:

```
predictor =      0.476800394
Corrector Iteration
x                y                   ea
2                0.187936982         1.54E+02
2                0.224044908         1.61E+01
2                0.219531418         2.06E+00
2                0.220095604         0.256336947
2                0.220025081         0.032052389
2                0.220033896         0.004006388

predictor =      0.204189336
Corrector Iteration
x                y                   ea
2.5       0.156440735         3.05E+01
2.5       0.161556656         3.166642375
2.5       0.161008522         0.340438161
2.5       0.16106725          0.036462217
2.5       0.161060958         0.003906819
```

26.9

```
Option Explicit

Sub SimpImplTest()
Dim i As Integer, m As Integer
Dim xi As Double, yi As Double, xf As Double, dx As Double, xout As Double
Dim xp(200) As Double, yp(200) As Double
'Assign values
yi = 0
xi = 0
xf = 2
dx = 0.1
xout = 0.1
'Perform numerical Integration of ODE
Call ODESolver(xi, yi, xf, dx, xout, xp, yp, m)
'Display results
Sheets("Sheet1").Select
Range("a5:b205").ClearContents
Range("a5").Select
For i = 0 To m
  ActiveCell.Value = xp(i)
  ActiveCell.Offset(0, 1).Select
  ActiveCell.Value = yp(i)
  ActiveCell.Offset(1, -1).Select
Next i
Range("a5").Select
End Sub

Sub ODESolver(xi, yi, xf, dx, xout, xp, yp, m)
'Generate an array that holds the solution
Dim x As Double, y As Double, xend As Double
Dim h As Double
m = 0
xp(m) = xi
yp(m) = yi
x = xi
y = yi
Do       'Print loop
  xend = x + xout
  If (xend > xf) Then xend = xf  'Trim step if increment exceeds end
  h = dx
  Call Integrator(x, y, h, xend)
  m = m + 1
  xp(m) = x
  yp(m) = y
  If (x >= xf) Then Exit Do
Loop
End Sub

Sub Integrator(x, y, h, xend)
Dim ynew As Double
Do       'Calculation loop
  If (xend - x < h) Then h = xend - x  'Trim step if increment exceeds end
  Call SimpImpl(x, y, h, ynew)
  y = ynew
  If (x >= xend) Then Exit Do
Loop
End Sub

Sub SimpImpl(x, y, h, ynew)
'Implement implicit Euler's method
ynew = (y + h * FF(x + h)) / (1 + 200000 * h)
x = x + h
End Sub

Function FF(x)
'Define Forcing Function
FF = 200000 * Exp(-x) - Exp(-x)
End Function
```



	A	B
4	x	y
5	0	0
6	0.1	0.904788
7	0.2	0.818731
8	0.3	0.740818
9	0.4	0.67032
10	0.5	0.606531
11	0.6	0.548812
12	0.7	0.496585
13	0.8	0.449329
14	0.9	0.40657
15	1	0.36788
16	1.1	0.332871
17	1.2	0.301194
18	1.3	0.272532
19	1.4	0.246597
20	1.5	0.22313
21	1.6	0.201897
22	1.7	0.182684
23	1.8	0.165299
24	1.9	0.149569
25	2	0.135335

26.10 All linear systems are of the form

$$\frac{dy_1}{dt} = a_{11}y_1 + a_{12}y_2 + F_1$$

$$\frac{dy_2}{dt} = a_{21}y_1 + a_{22}y_2 + F_2$$

As shown in the book (p. 730), the implicit approach amounts to solving

$$\begin{bmatrix} 1-a_{11}h & -a_{12} \\ -a_{21} & 1-a_{22}h \end{bmatrix} \begin{Bmatrix} y_{1,i+1} \\ y_{2,i+1} \end{Bmatrix} = \begin{Bmatrix} y_{1,i} + F_1 h \\ y_{2,i} + F_2 h \end{Bmatrix}$$

Therefore, for Eq. 26.6: a_{11} = –5, a_{12} = 3, a_{21} = 100, a_{22} = –301, F_1 =, and F_2 = 0,

$$\begin{bmatrix} 1+5h & -3 \\ -100 & 1+301h \end{bmatrix} \begin{Bmatrix} y_{1,i+1} \\ y_{2,i+1} \end{Bmatrix} = \begin{Bmatrix} y_{1,i} \\ y_{2,i} \end{Bmatrix}$$

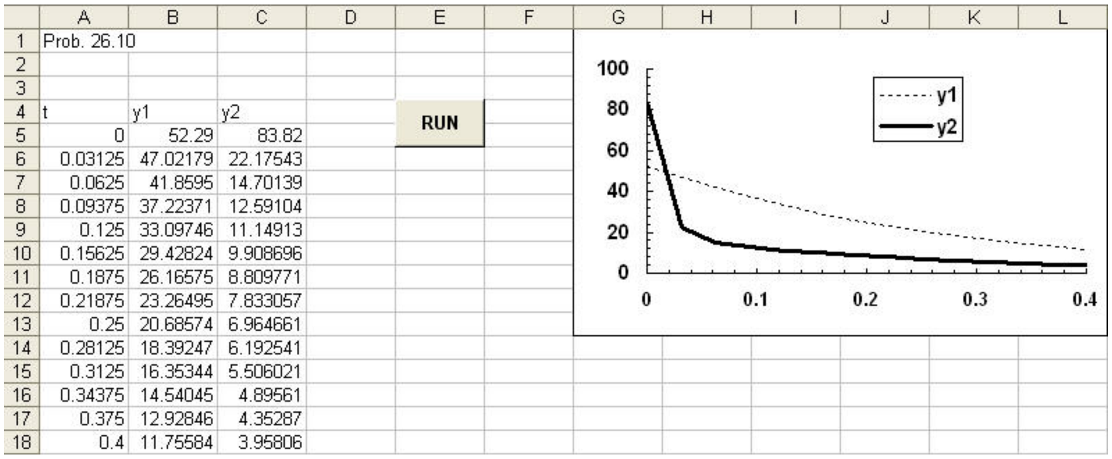
A VBA program written in these terms is

```
Option Explicit

Sub StiffSysTest()
Dim i As Integer, m As Integer, n As Integer, j As Integer
Dim xi As Single, yi(10) As Single, xf As Single, dx As Single, xout As Single
Dim xp(200) As Single, yp(200, 10) As Single

'Assign values
n = 2
xi = 0
xf = 0.4
yi(1) = 52.29
yi(2) = 83.82
dx = 0.05
xout = 0.05
```

```
'Perform numerical Integration of ODE
Call ODESolver(xi, yi(), xf, dx, xout, xp(), yp(), m, n)

'Display results
Sheets("Sheet1").Select
Range("a5:n205").ClearContents
Range("a5").Select
For i = 0 To m
  ActiveCell.Value = xp(i)
  For j = 1 To n
    ActiveCell.Offset(0, 1).Select
    ActiveCell.Value = yp(i, j)
  Next j
  ActiveCell.Offset(1, -n).Select
Next i
Range("a5").Select
End Sub

Sub ODESolver(xi, yi, xf, dx, xout, xp, yp, m, n)
'Generate an array that holds the solution
Dim i As Integer
Dim x As Single, y(10) As Single, xend As Single
Dim h As Single
m = 0
x = xi
'set initial conditions
For i = 1 To n
  y(i) = yi(i)
Next i
'save output values
xp(m) = x
For i = 1 To n
  yp(m, i) = y(i)
Next i
Do       'Print loop
  xend = x + xout
  If (xend > xf) Then xend = xf  'Trim step if increment exceeds end
  h = dx
  Call Integrator(x, y(), h, n, xend)
  m = m + 1
  'save output values
  xp(m) = x
  For i = 1 To n
    yp(m, i) = y(i)
  Next i
  If (x >= xf) Then Exit Do
Loop
End Sub

Sub Integrator(x, y, h, n, xend)
Dim j As Integer
Dim ynew(10) As Single
Do       'Calculation loop
  If (xend - x < h) Then h = xend - x  'Trim step if increment exceeds end
  Call StiffSys(x, y, h, n, ynew())
  For j = 1 To n
    y(j) = ynew(j)
  Next j
  If (x >= xend) Then Exit Do
Loop
End Sub
```

```
Sub StiffSys(x, y, h, n, ynew)
Dim j As Integer
Dim FF(2) As Single, b(2, 2) As Single, c(2) As Single, den As Single
Call Force(x, FF())
'MsgBox "pause"

b(1, 1) = 1 + 5 * h
b(1, 2) = -3 * h
b(2, 1) = -100 * h
b(2, 2) = 1 + 301 * h
For j = 1 To n
  c(j) = y(j) + FF(j) * h
Next j
den = b(1, 1) * b(2, 2) - b(1, 2) * b(2, 1)
ynew(1) = (c(1) * b(2, 2) - c(2) * b(1, 2)) / den
ynew(2) = (c(2) * b(1, 1) - c(1) * b(2, 1)) / den
x = x + h
End Sub

Sub Force(t, FF)
'Define Forcing Function
FF(0) = 0
FF(1) = 0
End Sub
```

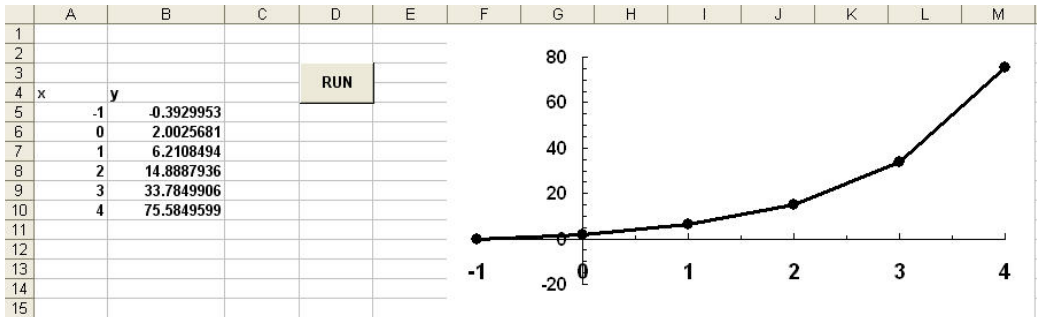
The result compares well with the analytical solution. If a smaller step size were used, the solution would improve



	A	B	C
1	Prob. 26.10		
2			
3			
4	t	y1	y2
5	0	52.29	83.82
6	0.03125	47.02179	22.17543
7	0.0625	41.8595	14.70139
8	0.09375	37.22371	12.59104
9	0.125	33.09746	11.14913
10	0.15625	29.42824	9.908696
11	0.1875	26.16575	8.809771
12	0.21875	23.26495	7.833057
13	0.25	20.68574	6.964661
14	0.28125	18.39247	6.192541
15	0.3125	16.35344	5.506021
16	0.34375	14.54045	4.89561
17	0.375	12.92846	4.35287
18	0.4	11.75584	3.95806

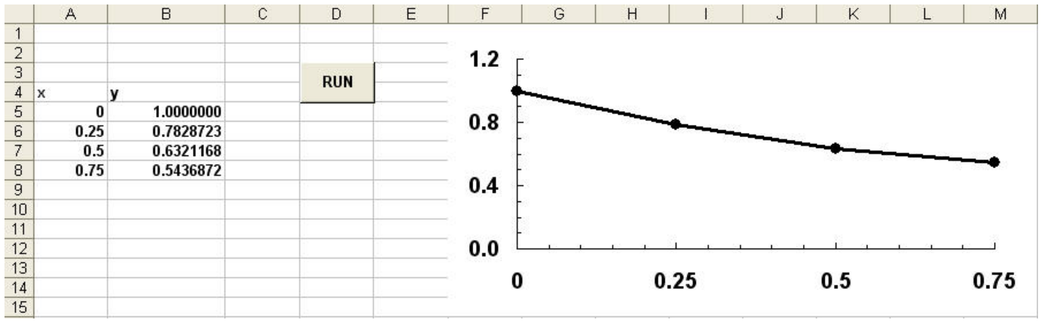
26.11
```
Option Explicit

Sub NonSelfStartHeun()
Dim n As Integer, m As Integer, i As Integer, iter As Integer
Dim xi As Double, xf As Double, yi As Double, h As Double
Dim x As Double, y As Double
Dim xp(1000) As Double, yp(1000) As Double
xi = -1
xf = 4
yi = -0.3929953
h = 1
n = (xf - xi) / h
x = xi
y = yi
```

```
m = 0
xp(m) = x
yp(m) = y
Call RK4(x, y, h)
m = m + 1
xp(m) = x
yp(m) = y
For i = 2 To n
  Call NSSHeun(xp(i - 2), yp(i - 2), xp(i - 1), yp(i - 1), x, y, h, iter)
  m = m + 1
  xp(m) = x
  yp(m) = y
Next i
Sheets("NSS Heun").Select
Range("a5:b1005").ClearContents
Range("a5").Select
For i = 0 To m
  ActiveCell.Value = xp(i)
  ActiveCell.Offset(0, 1).Select
  ActiveCell.Value = yp(i)
  ActiveCell.Offset(1, -1).Select
Next i
Range("a5").Select
End Sub

Sub RK4(x, y, h)

'Implement RK4 method
Dim k1 As Double, k2 As Double, k3 As Double, k4 As Double
Dim ym As Double, ye As Double, slope As Double
Call Derivs(x, y, k1)
ym = y + k1 * h / 2
Call Derivs(x + h / 2, ym, k2)
ym = y + k2 * h / 2
Call Derivs(x + h / 2, ym, k3)
ye = y + k3 * h
Call Derivs(x + h, ye, k4)
slope = (k1 + 2 * (k2 + k3) + k4) / 6
y = y + slope * h
x = x + h
End Sub

Sub NSSHeun(x0, y0, x1, y1, x, y, h, iter)
'Implement Non Self-Starting Heun
Dim i As Integer
Dim y2 As Double
Dim slope As Double, k1 As Double, k2 As Double
Dim ea As Double
Dim y2p As Double
Static y2old As Double, y2pold As Double
Call Derivs(x1, y1, k1)
y2 = y0 + k1 * 2 * h
y2p = y2
If iter > 0 Then
  y2 = y2 + 4 * (y2old - y2pold) / 5
End If
x = x + h
iter = 0
Do
  y2old = y2
  Call Derivs(x, y2, k2)
  slope = (k1 + k2) / 2
  y2 = y1 + slope * h
  iter = iter + 1
  ea = Abs((y2 - y2old) / y2) * 100
  If ea < 0.01 Then Exit Do
Loop
y = y2 - (y2 - y2p) / 5
y2old = y2
y2pold = y2p
End Sub

Sub Derivs(x, y, dydx)
'Define ODE
dydx = 4 * Exp(0.8 * x) - 0.5 * y
End Sub
```



x	y
-1	-0.3929953
0	2.0025681
1	6.2108494
2	14.8887936
3	33.7849906
4	75.5849599

26.12



x	y
0	1.0000000
0.25	0.7828723
0.5	0.6321168
0.75	0.5436872

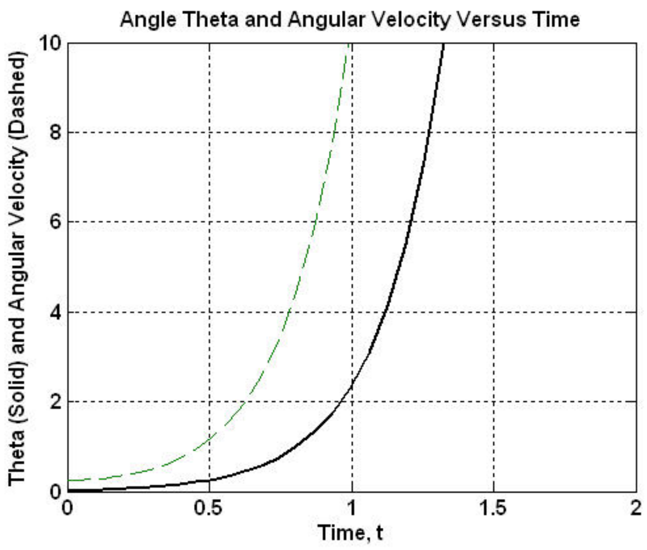
26.13 The second-order equation can be composed into a pair of first-order equations as

$$\frac{d\theta}{dt} = x \qquad\qquad \frac{dx}{dt} = \frac{g}{l}\theta$$

We can use MATLAB to solve this system of equations.

```
   tspan=[0,5]';
   x0=[0,0.25]';
   [t,x]=ode45('dxdt',tspan,x0);
   plot(t,x(:,1),t,x(:,2),'--')
   grid
   title('Angle Theta and Angular Velocity Versus Time')
   xlabel('Time, t')
   ylabel('Theta (Solid) and Angular Velocity (Dashed)')
   axis([0 2 0 10])
   zoom

   function dx=dxdt(t,x)
```

```
dx=[x(2);(9.81/0.5)*x(1)];
```



26.14 Analytic solution: Take Laplace transform

$$sX - x(0) = -700X - \frac{1000}{s+1}$$

$$sX + 700X = x(0) - \frac{1000}{s+1}$$

$$X = \frac{x(0)}{s+700} - \frac{1000}{(s+1)(s+700)}$$

Substituting the initial condition and expanding the last term with partial fractions gives,

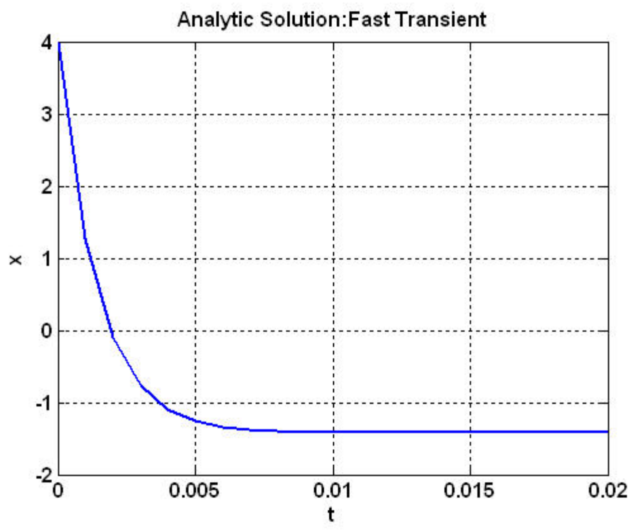
$$X = \frac{4}{s+700} - \frac{1.430615}{s+1} + \frac{1.430615}{s+700}$$

Taking inverse transforms yields

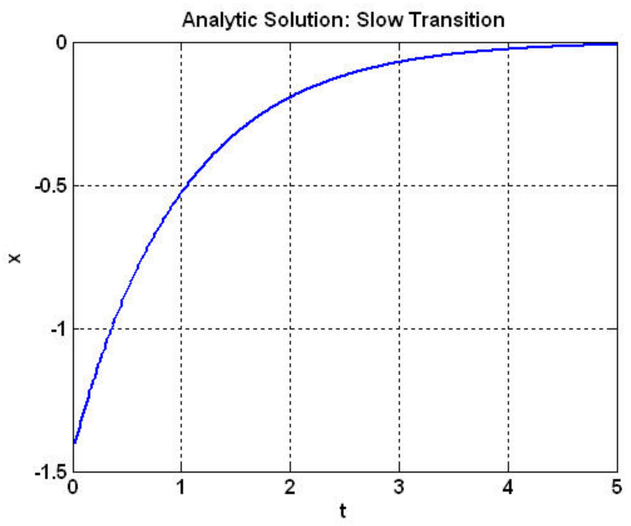
$$x = 5.430615e^{-700t} - 1.430615e^{-t}$$

MATLAB can be used to plot both the fast transient and the slow phases of the analytical solution.

```
t=[0:.001:.02];
x=5.430615*exp(-700*t)-1.430615*exp(-t);
plot(t,x)
grid
xlabel('t')
ylabel('x')
title('Analytic Solution:Fast Transient')
```



```
t=[0.02:.01:5];
x=6.004*exp(-500*t)-2.004*exp(-t);
plot(t,x)
grid
xlabel('t')
ylabel('x')
title('Analytic Solution: Slow Transition')
gtext('6.004e^-500t-2.004 e^-t')
```



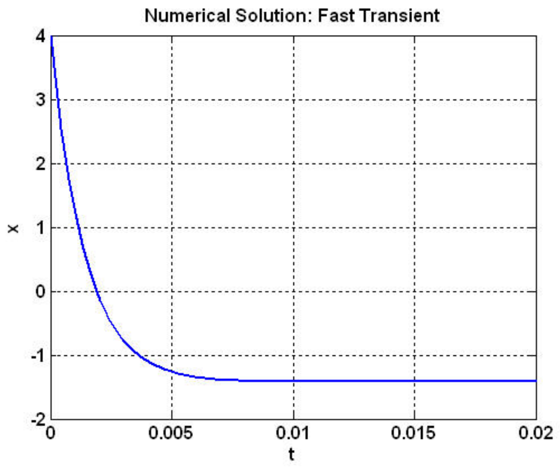
Numerical solution: First set up the function holding the differential equation:

```
function dx=dxdt(t,x)
dx=-700*x-1000*exp(-t);
```

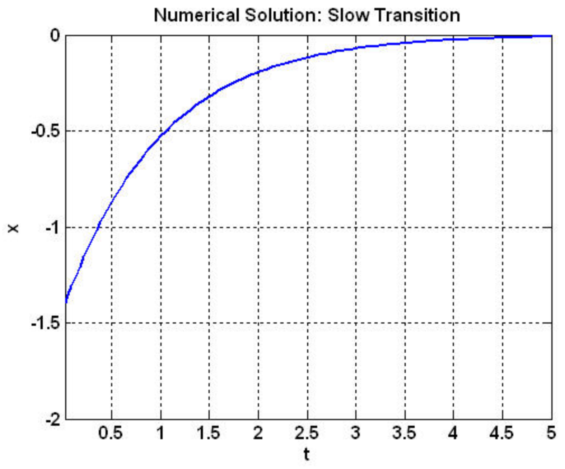
The following MATLAB code then generates the solutions and plots:

```
tspan=[0 5];
x0=[4];
[t,x]=ode23s(@dxdt,tspan,x0);
plot(t,x)
```

```
grid
xlabel('t')
ylabel('x')
title('Numerical Solution: Fast Transient')
axis([0 .02 -2 4])
```



```
tspan=[0 5];
x0=[4];
[t,x]=ode23s(@dxdt,tspan,x0);
plot(t,x)
grid
xlabel('t')
ylabel('x')
title('Numerical Solution: Slow Transition')
%axis([0.02 5 -2 0])
```



26.15 (a) Analytic solution:

$$y = \frac{1}{999}\left(1000e^{-x} - e^{-1000x}\right)$$

(b) The second-order differential equation can be expressed as the following pair of first-order ODEs,

$$\frac{dy}{dx} = w$$
$$\frac{dw}{dx} = -1000y - 1001w$$

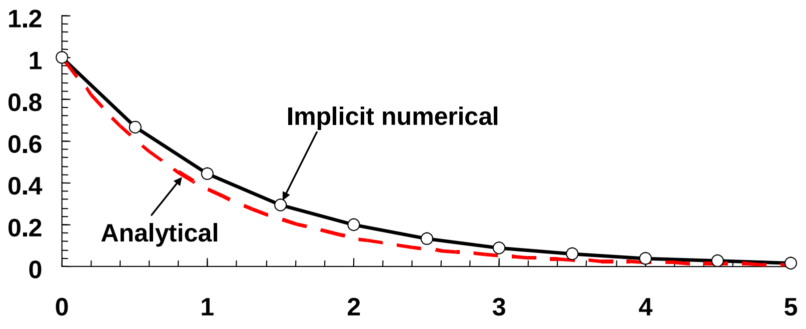
where $w = y'$. Using the same approach as described in Sec. 26.1, the following simultaneous equations need to be solved to advance each time step,

$$y_{i+1} - hw_{i+1} = y_i$$
$$1000hy_{i+1} + 1001hw_{i+1} = w_i$$

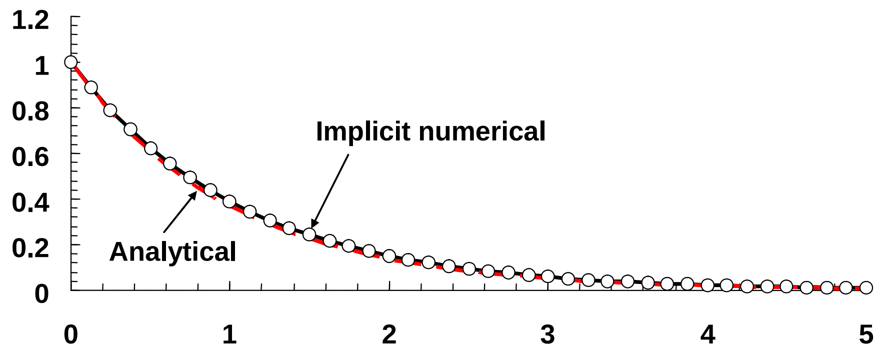
If these are implemented with a step size of 0.5, the following values are simulated

x	*y*	*w*
0	1	0
0.5	0.667332	-0.66534
1	0.444889	-0.44489
1.5	0.296593	-0.29659
2	0.197729	-0.19773
2.5	0.131819	-0.13182
3	0.087879	-0.08788
3.5	0.058586	-0.05859
4	0.039057	-0.03906
4.5	0.026038	-0.02604
5	0.017359	-0.01736

The results for *y* along with the analytical solution are displayed below:



Note that because we are using an implicit method the results are stable. However, also notice that the results are somewhat inaccurate. This is due to the large step size. If we use a smaller step size, the results will converge on the analytical solution. For example, if we use $h = 0.125$, the results are:

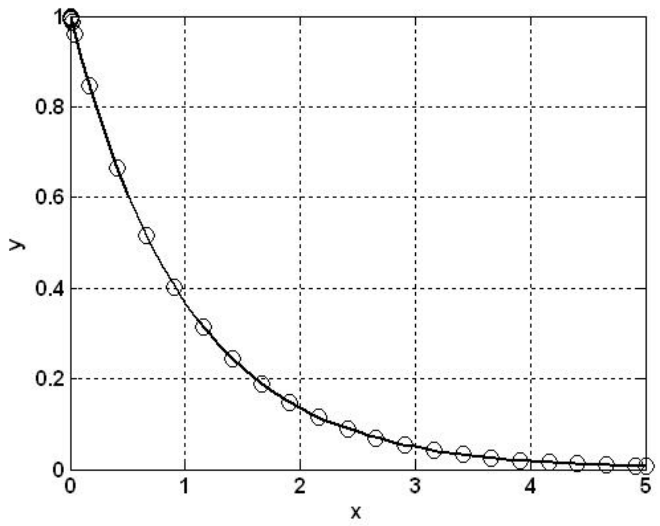


Finally, we can also solve this problem using one of the MATLAB routines expressly designed for stiff systems. To do this, we first develop a function to hold the pair of ODEs,

```
function dy = dydx(x, y)
dy = [y(2);-1000*y(1)-1001*y(2)];
```

Then the following session generates a plot of both the analytical and numerical solutions. As can be seen, the results are indistinguishable.

```
x=[0:.1:5];
y=1/999*(1000*exp(-x)-exp(-1000*x));
xspan=[0 5];
x0=[1 0];
[xx,yy]=ode23s(@dydx,xspan,x0);
plot(x,y,xx,yy(:,1),'o')
grid
xlabel('x')
ylabel('y')
```



26.16 (a) Analytic solution:

$$y = 1e^{-10t}$$

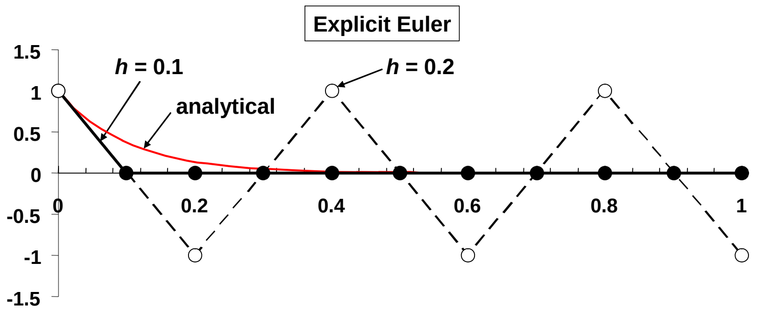
(b) Explicit Euler

$$y_{i+1} = y_i + (-10y_i)h$$

Here are the results for *h* = 0.2. Notice that the solution oscillates:

t	*y*	*dy*/*dt*
0	1	-10
0.2	-1	10
0.4	1	-10
0.6	-1	10
0.8	1	-10
1	-1	10

For *h* = 0.1, the solution plunges abruptly to 0 at *t* = 0.1 and then stays at zero thereafter. Both results are displayed along with the analytical solution below:



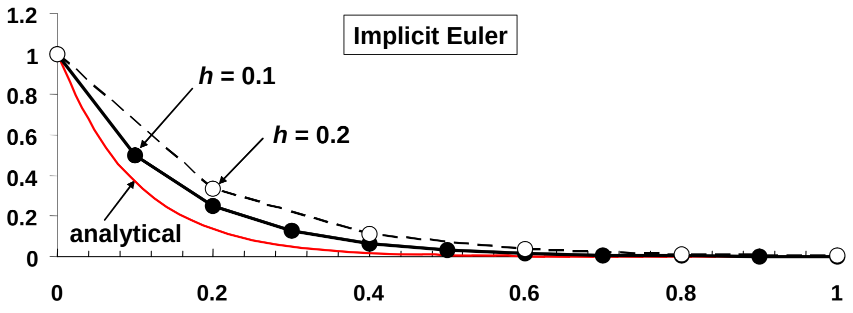
(c) Implicit Euler

$$y_{i+1} = \frac{y_i}{1+10h}$$

Here are the results for *h* = 0.2. Notice that although the solution is not very accurate, it is stable and declines monotonically in a similar fashion to the analytical solution.

t	*y*
0	1
0.2	0.333333333
0.4	0.111111111
0.6	0.037037037
0.8	0.012345679
1	0.004115226

For *h* = 0.1, the solution is also stable and tracks closer to the analytical solution. Both results are displayed along with the analytical solution below:


Implicit Euler
h = 0.1
h = 0.2
analytical