Numerical Computing II

Final Project

Samuel Vidovich

**Questions answered**

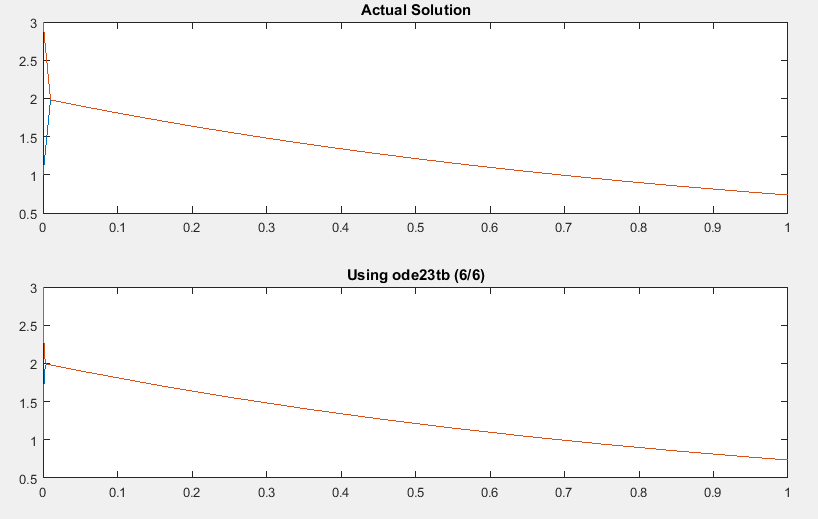
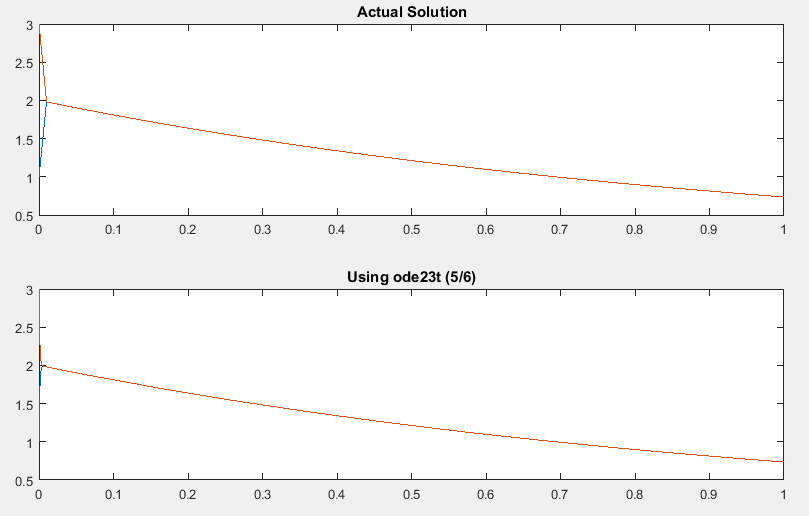
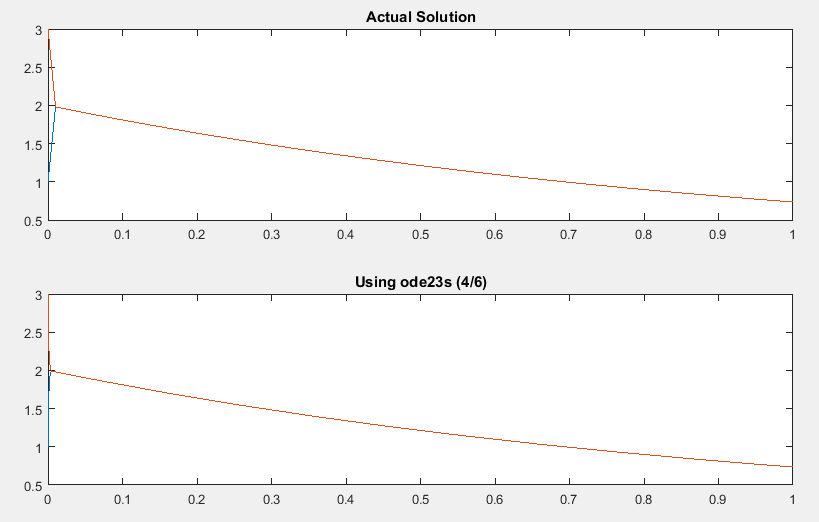
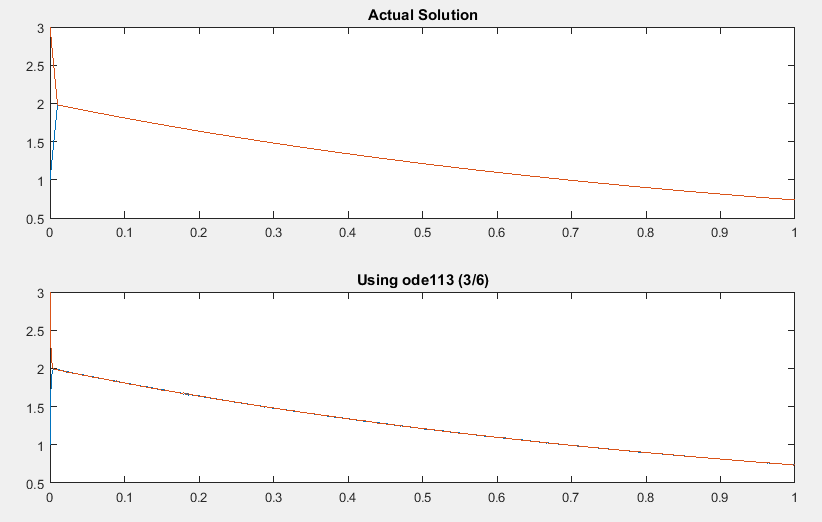
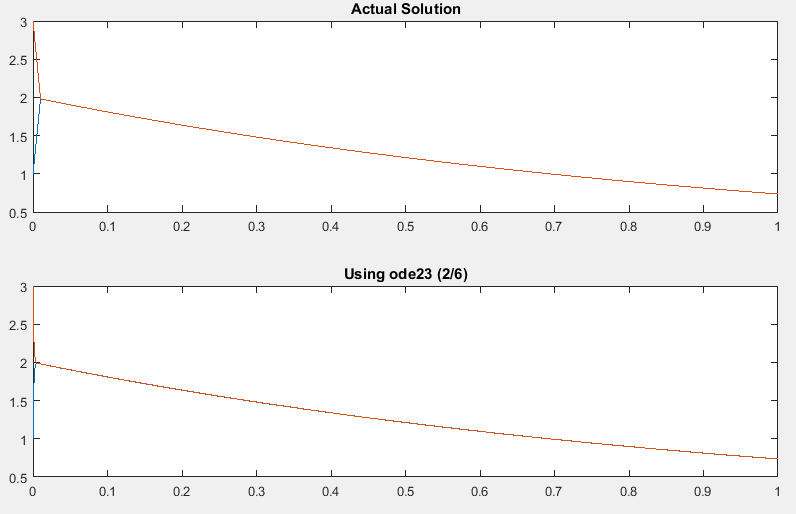
1. On sizes of h:

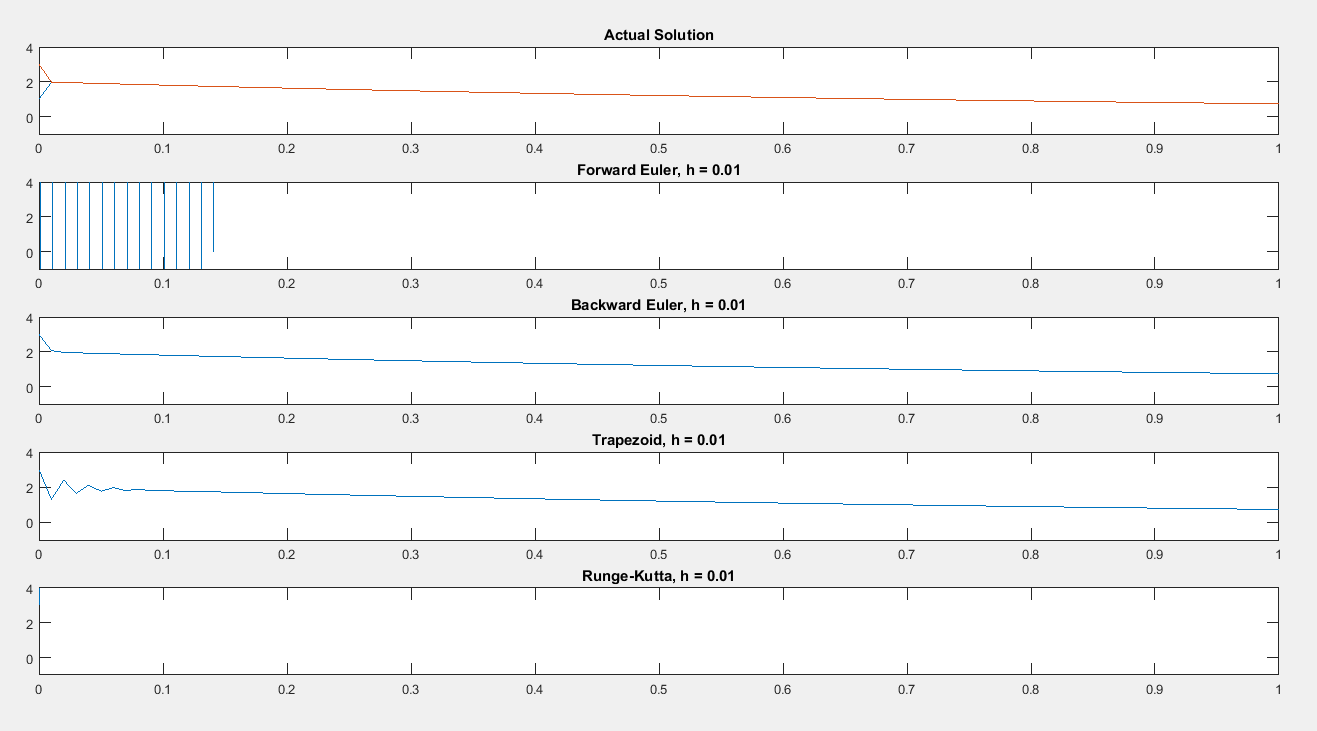
Runge Kutta requires a particularly small size of h to function at all. Forward Euler also needs a small h, or it oscillates. Backward Euler does fairly well even with a somewhat large h. Trapezoidal approximation admits a slight oscillation near the origin for *h = 10^-*2, but this disappears by the time *h = 10^-*4. For all of the algorithms, the size of h being small is very helpful.

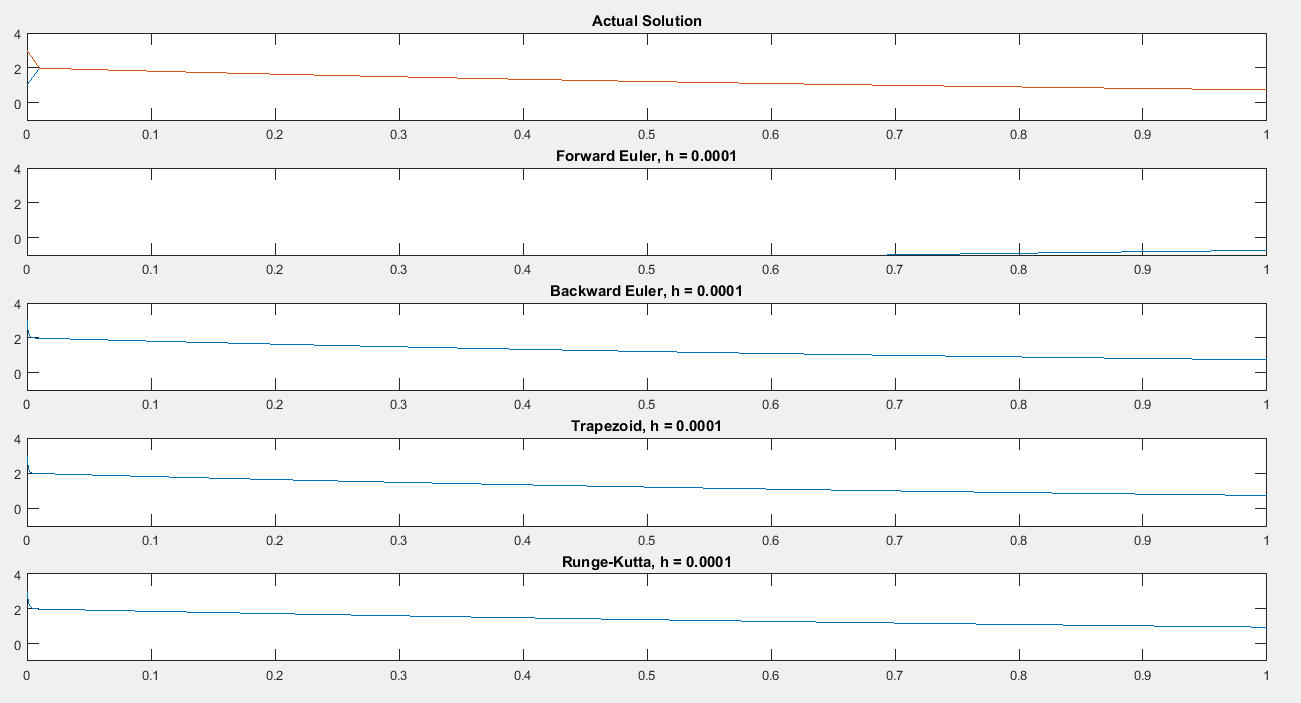
Except for forward Euler, which for a stiff problem like this is totally useless.

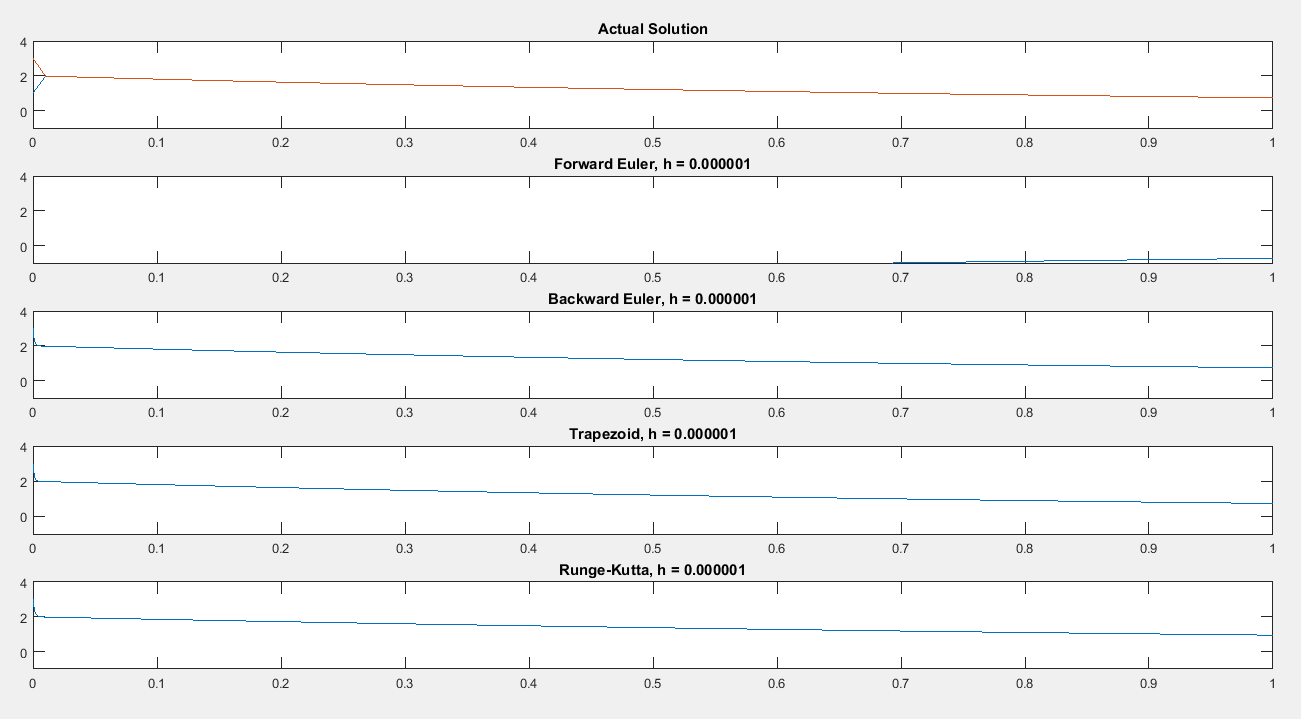
**Visuals**

Starting with the built-in Matlab solvers:



Next, the results from our programming:





The code:

% This will run both of the scripts which test the various ODE solvers.

customtesting

builtintesting

% sys.m

% This evaluates a 2d system of equations.

% The input is a 2-vector y.

% The output is a 2-vector f.

function f = sys(t,y)

f = zeros(2,1);

f(1,1) = -500.5\*y(1) + 499.5\*y(2);

f(2,1) = 499.5\*y(1) - 500.5\*y(2);

end

% customtesting.m

% This script tests some of the built in ODE solvers, and plots the results

% nicely!

y0 = [1; 3];

tspan = [0 1];

tini = linspace(0,1);

A = [-500.5 499.5; 499.5 -500.5];

hlo = 10^-2;

hmed = 10^-4;

hhi = 10^-6;

% The first subplot will always be the actual solution

figure(2)

subplot(5,1,1)

plot(tini,2\*exp(-tini) - exp(-1000\*tini));

axis([0 1 -1 4])

hold on;

plot(tini,2\*exp(-tini) + exp(-1000\*tini));

axis([0 1 -1 4])

title('Actual Solution')

% Begin by plotting with low accuracy

[t, y] = feuler(A, hlo, y0, tspan);

subplot(5,1,2)

plot(t,y(1,:));

axis([0 1 -1 4])

plot(t,y(2,:));

axis([0 1 -1 4])

title('Forward Euler, h = 0.01');

clearvars t y

[t, y] = beuler(A, hlo, y0, tspan);

subplot(5,1,3)

plot(t,y(1,:));

axis([0 1 -1 4])

plot(t,y(2,:));

axis([0 1 -1 4])

title('Backward Euler, h = 0.01');

clearvars t y

[t, y] = trapezoid(A, hlo, y0, tspan);

subplot(5,1,4)

plot(t,y(1,:));

axis([0 1 -1 4])

plot(t,y(2,:));

axis([0 1 -1 4])

title('Trapezoid, h = 0.01');

clearvars t y

[t, y] = runge(A, hlo, y0, tspan);

subplot(5,1,5)

plot(t,y(1,:));

axis([0 1 -1 4])

plot(t,y(2,:));

axis([0 1 -1 4])

title('Runge-Kutta, h = 0.01');

clearvars t y

pause

% Next, plot with medium accuracy

[t, y] = feuler(A, hmed, y0, tspan);

subplot(5,1,2,'replace')

plot(t,y(1,:));

axis([0 1 -1 4])

plot(t,y(2,:));

axis([0 1 -1 4])

title('Forward Euler, h = 0.0001');

clearvars t y

[t, y] = beuler(A, hmed, y0, tspan);

subplot(5,1,3,'replace')

plot(t,y(1,:));

axis([0 1 -1 4])

plot(t,y(2,:));

axis([0 1 -1 4])

title('Backward Euler, h = 0.0001');

clearvars t y

[t, y] = trapezoid(A, hmed, y0, tspan);

subplot(5,1,4,'replace')

plot(t,y(1,:));

axis([0 1 -1 4])

plot(t,y(2,:));

axis([0 1 -1 4])

title('Trapezoid, h = 0.0001');

clearvars t y

[t, y] = runge(A, hmed, y0, tspan);

subplot(5,1,5,'replace')

plot(t,y(1,:));

axis([0 1 -1 4])

plot(t,y(2,:));

axis([0 1 -1 4])

title('Runge-Kutta, h = 0.0001');

clearvars t y

pause

% Next, plot with high accuracy

[t, y] = feuler(A, hhi, y0, tspan);

subplot(5,1,2,'replace')

plot(t,y(1,:));

axis([0 1 -1 4])

plot(t,y(2,:));

axis([0 1 -1 4])

title('Forward Euler, h = 0.000001');

clearvars t y

[t, y] = beuler(A, hhi, y0, tspan);

subplot(5,1,3,'replace')

plot(t,y(1,:));

axis([0 1 -1 4])

plot(t,y(2,:));

axis([0 1 -1 4])

title('Backward Euler, h = 0.000001');

clearvars t y

[t, y] = trapezoid(A, hhi, y0, tspan);

subplot(5,1,4,'replace')

plot(t,y(1,:));

axis([0 1 -1 4])

plot(t,y(2,:));

axis([0 1 -1 4])

title('Trapezoid, h = 0.000001');

clearvars t y

[t, y] = runge(A, hhi, y0, tspan);

subplot(5,1,5,'replace')

plot(t,y(1,:));

axis([0 1 -1 4])

plot(t,y(2,:));

axis([0 1 -1 4])

title('Runge-Kutta, h = 0.000001');

clearvars t y

pause

close all;

% builtintesting.m

% This script tests some of the built in ODE solvers, and plots the results

% nicely!

y0 = [1 3];

tspan = [0 1];

tini = linspace(0,1);

figure(1)

subplot(2,1,1)

plot(tini,2\*exp(-tini) - exp(-1000\*tini));

hold on;

subplot(2,1,1)

plot(tini,2\*exp(-tini) + exp(-1000\*tini));

title('Actual Solution')

[t, y] = ode45('sys', tspan, y0);

subplot(2,1,2)

plot(t,y)

title('Using ode45 (1/6)')

clearvars t y;

pause

[t, y] = ode23('sys', tspan, y0);

subplot(2,1,2,'replace')

plot(t,y)

title('Using ode23 (2/6)')

clearvars t y;

pause

[t, y] = ode113('sys', tspan, y0);

subplot(2,1,2,'replace')

plot(t,y)

title('Using ode113 (3/6)')

clearvars t y;

pause

[t, y] = ode23s('sys', tspan, y0);

subplot(2,1,2,'replace')

plot(t,y)

title('Using ode23s (4/6)')

clearvars t y;

pause

[t, y] = ode23t('sys', tspan, y0);

subplot(2,1,2,'replace')

plot(t,y)

title('Using ode23t (5/6)')

clearvars t y;

pause

[t, y] = ode23tb('sys', tspan, y0);

subplot(2,1,2,'replace')

plot(t,y)

title('Using ode23tb (6/6)')

clearvars t y;

pause

close all

% runge.m

% This is a fourth-order Runge Kutta algorithm.

% Inputs are as follows:

% A: A 2d matrix from equation y' = Ay

% h: A scalar, step size

% y0: A 2-vector, initial condition

% t: A 2-vector, the first element of which is the beginning of the

% interval of interest, the second element of which is the end.

function [tint, y] = runge(A, h, y0, t)

tint = t(1):h:t(2);

n = length(tint);

y = zeros(2,n);

y(:,1) = y0;

for i = 1:1:(n-1)

k1 = A\*y(:,i);

k2 = A\*(y(:,i) + (h/2)\*k1);

k3 = A\*(y(:,i) + (h/2)\*k2);

k4 = A\*(y(:,i) + h\*k3);

y(:,i+1) = y(:,i) + (h/8)\*(k1 + 2\*k2 + 2\*k3 + k4);

end

end

% Trapezoid.m

% This is a trapezoid rule algorithm.

% Inputs are as follows:

% A: A 2d matrix from equation y' = Ay

% h: A scalar, step size

% y0: A 2-vector, initial condition

% t: A 2-vector, the first element of which is the beginning of the

% interval of interest, the second element of which is the end.

function [tint, y] = trapezoid(A, h, y0, t)

tint = t(1):h:t(2);

n = length(tint);

y = zeros(2,n);

y(:,1) = y0;

I = eye(2,2);

for i = 1:1:(n-1)

y(:,i+1) = (I - (h/2)\*A)\((I + (h/2)\*A)\*y(:,i));

end

end

% beuler.m

% This is a backward euler algorithm.

% Inputs are as follows:

% A: A 2d matrix from equation y' = Ay

% h: A scalar, step size

% y0: A 2-vector, initial condition

% t: A 2-vector, the first element of which is the beginning of the

% interval of interest, the second element of which is the end.

function [tint, y] = beuler(A, h, y0, t)

tint = t(1):h:t(2);

n = length(tint);

y = zeros(2,n);

y(:,1) = y0;

I = eye(2,2);

for i = 1:1:(n-1)

y(:,i+1) = (I - h\*A)\y(:,i);

end

end

% feuler.m

% This is a forward Euler algorithm.

% Inputs are as follows:

% A: A 2d matrix from equation y' = Ay

% h: A scalar, step size

% y0: A 2-vector, initial condition

% t: A 2-vector, the first element of which is the beginning of the

% interval of interest, the second element of which is the end.

function [tint, y] = feuler(A, h, y0, t)

tint = t(1):h:t(2);

n = length(tint);

y(:,1) = A\*y0;

for i = 1:1:(n-1)

y(:,i+1) = y(:,i) + h\*A\*y(:,i);

end

end