

Numerical Analysis – Iterative Methods

Problem	Initial Value	Solution Interval	Solution
$y' = (t + y)^2$	$y(0) = -1$	$[0, \frac{\pi}{2}]$	$y(t) = -t + \tan(t - \frac{\pi}{4})$

Before proceeding to solve the queries, we will need to define functions that implement the I.V.P. (Initial Value Problem) corresponding to the problem above. We will define the functions **g.m** and **gtrue.m** which correspond to the aforementioned problem and its solution.

g.m

```
function yprime = g(t,y)
% g.m
% This function (problem 9) corresponds to
% my registration number (ge15039).
yprime = (t + y).^2;
end
```

gtrue.m

```
function ytrue = gtrue(t)
% gtrue.m
% Analytical solution of g.m (problem 9).
ytrue = -t + tan(t-pi/4);
end
```

Also, before proceeding to the questions, it is necessary to present all the methods that will be used:

rk4.m

```
function [tout, yout] = rk4(FunFcn,t0,tfinal,step,y0)
% Costant Stepsize 4 step order 4 rk4.m
% Initialization
ceta    = [ 1/2      1/2      1 ]';
alpha   = [ [ 1/2      0      0      0      ]
            [ 0       1/2      0      0      ]
            [ 0       0       1      0      ] ]';
beta    = [ 1/6      1/3      1/3      1/6      ]';
stages=4;
t = t0; y = y0(:); f = y*zeros(1,stages); tout = t; yout = y.';
```

```

% The main loop
while abs(t- tfinal)> 1e-6
    if t + step > tfinal, step = tfinal - t; end
    % Compute the slopes
    temp = feval(FunFcn,t,y);
    f(:,1) = temp(:);
    for j = 1:stages-1
        temp = feval(FunFcn, t+ceta(j)*step, y+step*f*alpha(:,j));
        f(:,j+1) = temp(:);
    end
    t = t + step;
    y = y + step*f*beta(:,1);
    tout = [tout; t];
    yout = [yout; y.'];
end;

```

ab5sem.m

```

function [tout, yout] = ab5sem(FunFcn,t0,tfinal,step,y0)
% ab5sem.m Costant Stepsize
% Adams Bashforth 5 step Explicit Method order 4
% calls rk4 for 4 steps
beta = [251 -1274 2616 -2774 1901 ]/720;
stages=5;
[tout,yout]=rk4(FunFcn,t0,t0+(stages-1)*step,step,y0);
tout=tout(1:stages);
yout=yout(1:stages);
t = tout(stages);
y = yout(stages).';
% The main loop
while abs(t- tfinal)> 1e-6
    if t + step > tfinal, step = tfinal - t; end
    f=feval(FunFcn,tout(length(tout)-
stages+1:length(tout)),yout(length(tout)-stages+1:length(tout)));
    y=y+step* beta*f;
    t = t + step;
    tout = [tout; t];
    yout = [yout; y.'];
end;

```

trap.m

```

function [tout, yout] = trap(FunFcn,t0,tfinal,step,y0)
% trap.m Implicit trapezium method for scalar systems
% It will not work with systems
% Initialization
tol=1e-9;
told=1e-6;
maxiter=30;
t = t0;
y = y0(:);
tout = t;

```

```

yout = y.';
% The main loop
while abs(t- tfinal)> 1e-6
    if t + step > tfinal, step = tfinal - t; end
    % Compute the slopes
    yp0=y;
    ypf=yp0;
    yp=inf;
    iter=0;
    while (abs(yp-ypf)>=tol)& (iter < maxiter)
        fyest=(feval(FunFcn,t+step,yp0+told)-
feval(FunFcn,t+step,yp0))/told;
        yp=yp0-(yp0-y-
step/2*(feval(FunFcn,t+step,yp0)+feval(FunFcn,t,y)))/(1-
step/2*fyest);
        ypf=yp0;
        yp0=yp;
        iter=iter+1;
    end
    if iter == maxiter
        disp('Maximum number of iteration succeeded');
        return;
    end
    t = t + step;
    y=yp;
    tout = [tout; t];
    yout = [yout; y.'];
end;

```

impeuler.m

```

function [tout, yout] = impeuler(FunFcn,t0,tfinal,step,y0)
% Costant Stepsize Improved Euler Method
% Initialization
t = t0;
y = y0(:);
tout = t;
yout = y.';
% The main loop
while abs(t- tfinal)> 1e-6
    if t + step >= tfinal, step = tfinal - t; end
    % Compute the slopes
    s1 = feval(FunFcn, t, y); s1 = s1(:);
    s2 = feval(FunFcn,t+step, y+step*s1); s2 = s2(:);
    t = t + step;
    y = y + step*(s1 + s2)/2;
    tout = [tout; t];
    yout = [yout; y.'];
end;

```

Question 1

Convert the program **rk4.m** to implement the 4th order Runge Kutta method with 5 stages, with coefficients:

0	0	0	0	0	0
1/3	1/3	0	0	0	0
1/3	1/6	1/6	0	0	0
1/2	1/8	0	3/8	0	0
1	1/2	0	-3/2	1	0
<hr/>					
	1/6	0	0	2/3	1/6

Name the program **nrk4.m**.

Answer 1

To construct the program **nrk4.m**, it is sufficient to replace the quantities in the ready-made program **rk4.m**:

c	A
	b^T

with those listed in the given coefficient table and increase the steps from 4 to 5, thus marginally changing the program **rk4.m**.

nrk4.m

```
function [tout, yout] = nrk4(FunFcn,t0,tfinal,step,y0)
% Costant step size 5 step order 4 nrk4.m
% Initialization
ceta = [ 1/3 1/3 1/2 1 ]';
alpha = [ [ 1/3 0 0 0 0 ]
          [ 1/6 1/6 0 0 0 ]
          [ 1/8 0 3/8 0 0 ]
          [ 1/2 0 -3/2 1 0 ] ]';
beta = [ 1/6 0 0 2/3 1/6 ]';
stages=5;
t = t0; y = y0(:); f = y*zeros(1,stages); tout = t; yout = y.';
% The main loop
while abs(t- tfinal)> 1e-6
    if t + step > tfinal, step = tfinal - t; end
    % Compute the slopes
    temp = feval(FunFcn,t,y);
    f(:,1) = temp(:);
    for j = 1:stages-1
        temp = feval(FunFcn, t+ceta(j)*step, y+step*f*alpha(:,j));
```

```

        f(:,j+1) = temp(:);
    end
    t = t + step;
    y = y + step*f*beta(:,1);
    tout = [tout; t];
    yout = [yout; y.'];
end;

```

The following script includes the solution of the problem with the **nrk4.m** method in the given interval and the initial conditions corresponding to them (according to the original table) for step $h=0.1$ was used.

q1.m

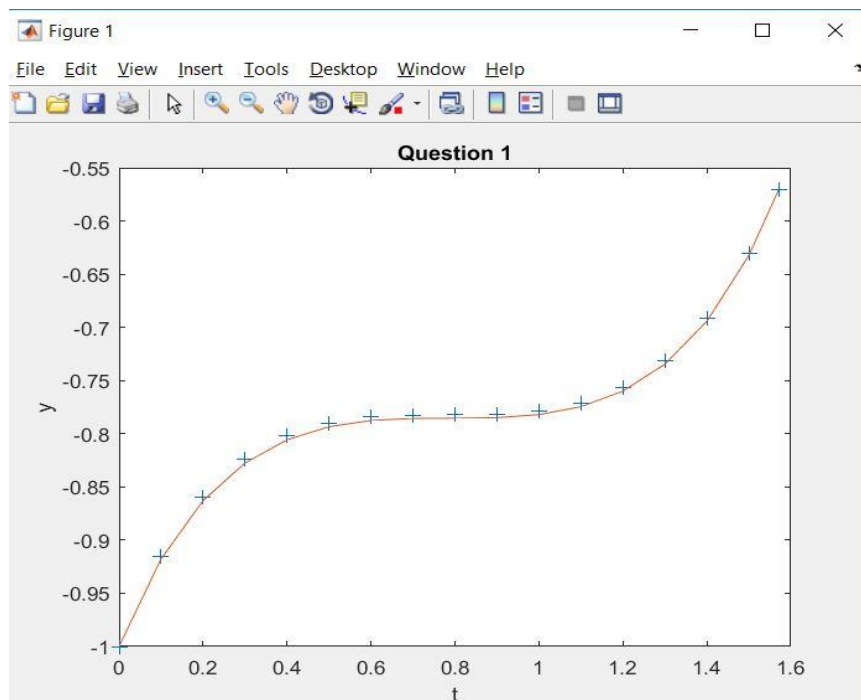
```

% Driver program q1.m (Question 1).
% Solve the problem g.m using the nrk4.m method
% use [0,pi/2] as an interval, for step size use h=0.1
% and as initial value use y(0)=-1.

% Initialise
t0      = 0;
tfinal  = pi/2;
y0      = -1;
step    = 0.1;
% Solve
[tout,yout]=nrk4('g',t0,tfinal,step,y0);
% Plot
plot(tout,yout,'+',tout,gtrue(tout),'-')
title('Question 1'),xlabel('t'),ylabel('y');

```

Running **q1.m** in the command window yields:



Question 2

Convert the **ab5sem.m** program to implement the 2-step multistep method:

$$y_{n+2} - y_n = h[(1/3)f_{n+2} + (4/3)f_{n+1} + (1/3)f_n],$$

where $f_n = f(x_n, y_n)$. The program should be named **new.m** (note that this is an indirect method and the Newton-Raphson method should be used).

Answer 2

The modification in **ab5sem.m** is limited to the part of the solution where the Newton Raphson method is used (because the method is indirect). In particular, the Newton Raphson method is applied as it was applied in the **trap.m** trapezium method but in this case the iterative formula is obtained:

$$y_{n+2}^k = y_{n+2}^{k-1} - \frac{F(y_{n+2}^{k-1})}{F'(y_{n+2}^{k-1})}$$

$$y_{n+2}^k = y_{n+2}^{k-1} - \frac{y_{n+2}^{k-1} - y_n - \frac{h}{3}(f(x_{n+2}, y_{n+2}^{k-1}) + 4f(x_{n+1}, y_{n+1}) + f(x_n, y_n))}{1 - \frac{h}{3}(\frac{\partial f(x_{n+2}, y_{n+2}^{k-1})}{\partial y})}$$

The conversion of **ab5sem.m** so that it implements the given two-step multi-step method was named **new.m**.

new.m

```
function [tout, yout] = new(FunFcn,t0,tfinal,step,y0)
% new.m Costant step size
% Adams Bashforth (modified) 2 step Explicit Method
% calls rk4 for 2 steps

% Initialisation:
% Notation: tolnew = tolerance of new method, tolnr = tolerance of
% Newton Raphson method, maxiter = maximum iterations, told =
% differentiation delta value.
```

```

tolnew = 1e-6;
tolnr = 1e-9;
maxiter = 30;
told = 1e-6;

stages = 2;

[tout,yout] = rk4(FunFcn,t0,t0+(stages-1)*step,step,y0);
tout = tout(1:stages);
yout = yout(1:stages);
% Assign t and y values to the final
% elements of tout and yout respectively.
t = tout(stages);
y = yout(stages).';

% The main loop
while abs(t - tfinal) > tolnew
    if t + step > tfinal, step = tfinal - t; end

    % Definition of variables related with the N.R. estimation.
    yp0 = y;
    ypf = yp0;
    yp = inf;
    iter = 0;

    % Newton Raphson estimation.
    while(abs(yp - ypf) >= tolnr)&&(iter < maxiter)
        fyest = (feval(FunFcn,t+step,yp0+told) -
feval(FunFcn,t+step,yp0))/told;
        yp = yp0 - ( yp0 - yout(end-1) - step/3 *
(feval(FunFcn,t+step,yp0) + 4 * feval(FunFcn,t,y) + feval(FunFcn,t-
step,yout(end-1))) ) / (1-step/3 * fyest);
        % Reassign values.
        ypf = yp0;
        yp0 = yp;
        iter = iter + 1;
    end;
    t = t + step;

    y = yp;
    tout = [tout; t];
    yout = [yout; y.'];
end;

```

The following script involves solving the initial problem with the **new.m** method in the given interval, initial conditions and for step $h=0.1$.

q2.m

```

% Driver program q2.m (Question 2).
% Solve the problem g.m using the new.m method
% use [0,pi/2] as an interval, for step size use h=0.1
% and as initial value use y(0)=-1.

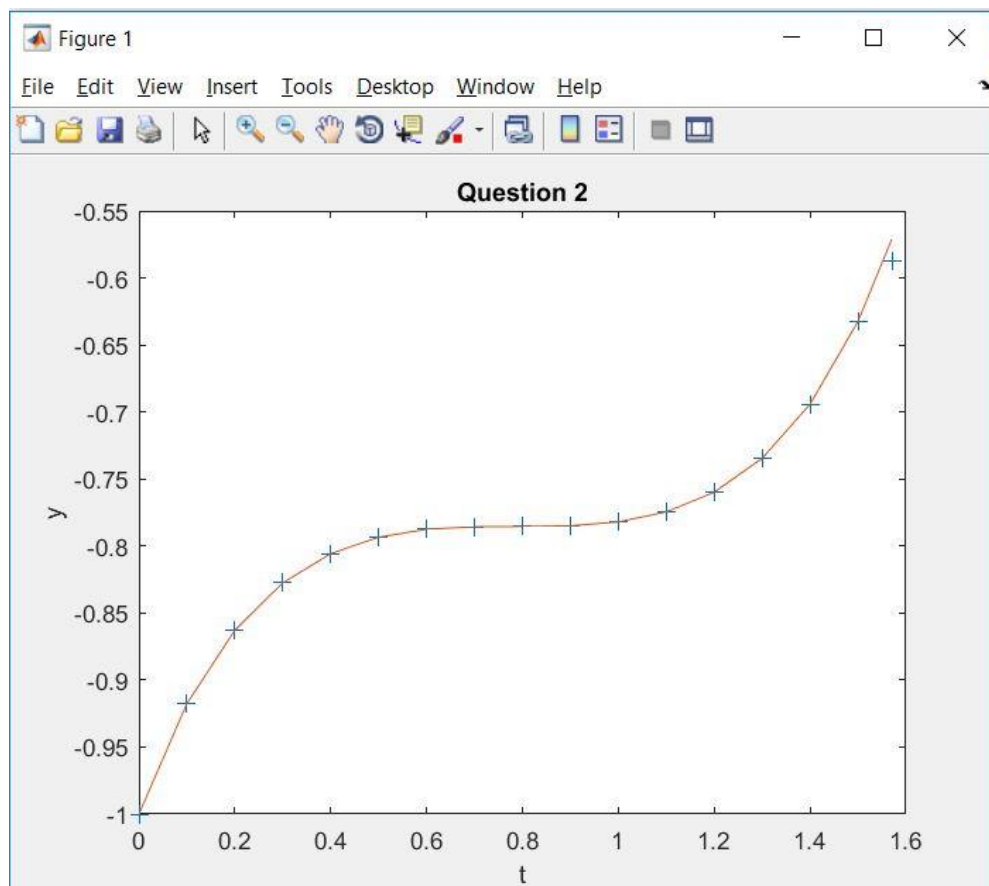
```

```

% Initialise
t0      = 0;
tfinal  = pi/2;
y0      = -1;
step    = 0.1;
% Solve
[tout,yout]=new('g',t0,tfinal,step,y0);
% Plot
plot(tout,yout,'+',tout,gtrue(tout),'-')
title('Question 2'),xlabel('t'),ylabel('y');

```

Running **q2.m** in the command window yields:



Question 3

Given the following initial value problem

$$u'(t) = v(t)$$

$$v'(t) = -0.2v(t) - \sin(u(t))$$

with initial values $u(0) = \pi/2$ and $v(0) = 0$.

A) Solve **nrk4.m** in the interval **[0,3]** with **h=0.1**, **h=0.01**, **h=0.00001** and comment on the results.

B) Run the same problem for **h=0.01** utilizing the improved Euler method. Comment on the results.

Answer 3

Before solving the individual questions, it is necessary to define a function that models the I.V.P. of Question 3. By setting a function of two variables as follows:

$$\mathbf{y}(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = \begin{pmatrix} u(t) \\ v(t) \end{pmatrix}$$

$$\mathbf{y}'(t) = \begin{pmatrix} u'(t) \\ v'(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ -0.2v(t) - \sin(u(t)) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & -0.2 \end{pmatrix} \mathbf{y}(t) + \begin{pmatrix} 0 & 0 \\ -1 & 0 \end{pmatrix} \sin(\mathbf{y}(t))$$

The situation is simplified greatly and the problem is modelled through the following function.

f.m

```
function yprime = f(t,y)
% f.m
yprime = [ 0 1 ; 0 -0.2 ] * y + [ 0 0 ; -1 0 ] * sin(y);
end
```

A)

The following script includes the solution of the I.V.P. of Question 3 using the **nrk4.m** method in the interval [0,3] and with initial values $u(0) = \pi/2$ and $v(0) = 0$, for step $h=0.1$, $h=0.01$ and $h=0.00001$ were used.

q3a.m

```
% Driver program q3a.m (Question 3 subquestion A).
% Solve the problem f.m using the nrk4.m method
% use [0,3] as an interval, for step size use h=0.1,
% h=0.01 and h=0.00001 and as initial values use
% u(0)=pi/2 & v(0)=0.

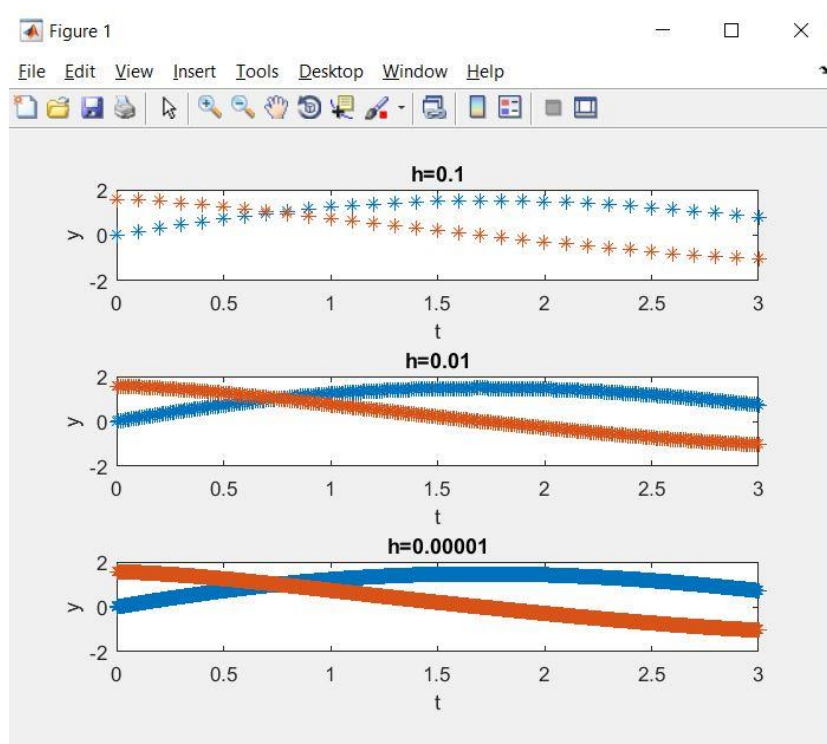
% Initialise
t0=0;
tfinal=3;
y0=[0;pi/2];
step1=0.1;
step2=0.01;
step3=0.00001;
% Solve
```

```

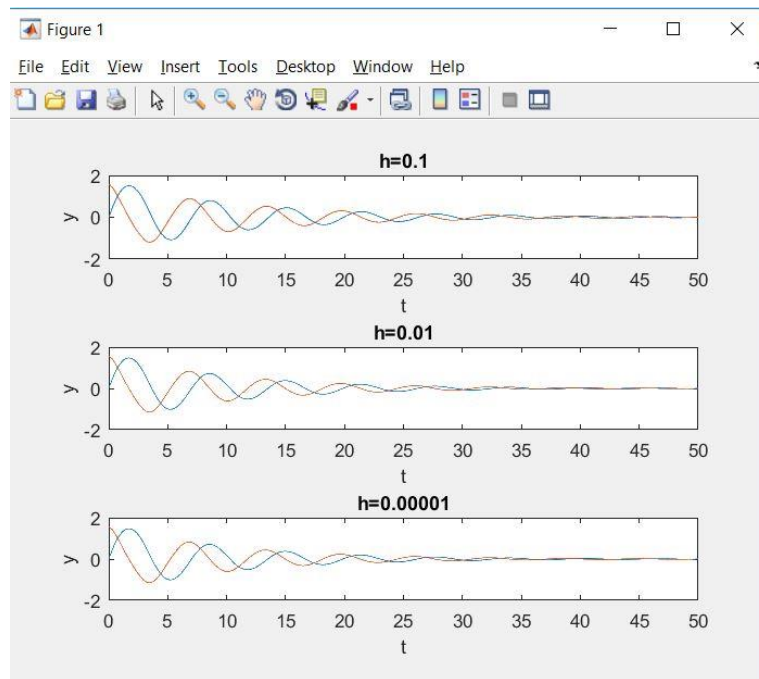
[tout1,yout1]=nrk4('f',t0,tfinal,step1,y0);
[tout2,yout2]=nrk4('f',t0,tfinal,step2,y0);
[tout3,yout3]=nrk4('f',t0,tfinal,step3,y0);
% Plot
subplot(3,1,1),plot(tout1,yout1,'*')
title('h=0.1'),xlabel('t'),ylabel('y');
subplot(3,1,2),plot(tout2,yout2,'*')
title('h=0.01'),xlabel('t'),ylabel('y');
subplot(3,1,3),plot(tout3,yout3,'*')
title('h=0.00001'),xlabel('t'),ylabel('y');

```

Running the command **q3a.m** through the command window yields:



Note that the graph depicts two plots which is expected, since $y(t)$ as defined in the function **f.m** is a function of two variables. Also, we see that the solutions of the I.V.P. are periodic functions with some phase difference and high damping as shown in the graph below which was plotted for a larger interval.



B)

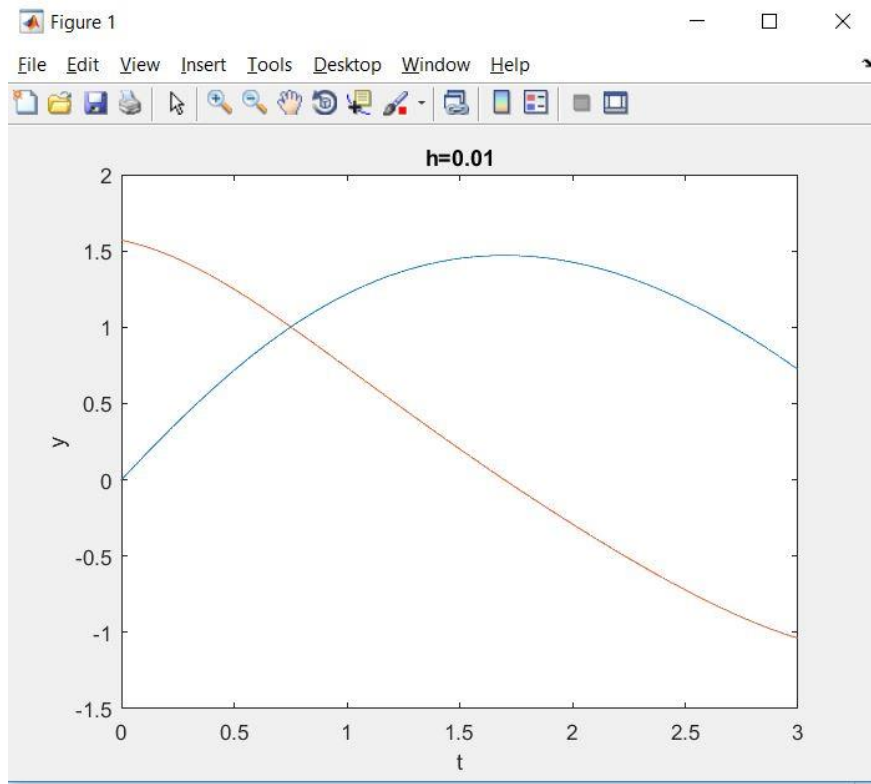
The following script solves the I.V.P. of Question 3 with the **impeuler.m** method in the interval $[0,3]$ and with initial values $u(0) = \pi/2$ and $v(0) = 0$, for step $h=0.01$.

q3b.m

```
% Driver program q3b.m (Question 3 subquestion B).
% Solve the problem f.m using the impeuler.m method
% use [0,3] as an interval, for step size use h=0.01
% and as initial values use u(0)=pi/2 & v(0)=0.

% Initialise
t0=0;
tfinal=3;
y0=[0;pi/2];
step=0.01;
% Solve
[tout,yout]=impeuler('f',t0,tfinal,step,y0);
% Plot
plot(tout,yout,'-')
title('h=0.01'),xlabel('t'),ylabel('y');
```

Running the command **q3b.m** in the command window yields:



We observe that the **impeuler.m** method is more efficient than **nrk4.m** as a direct method in which it is not necessary to solve a nonlinear system at each step.

Question 4

It is known that for a multistep method of order p the following relation for the total error, stands:

$$GE_{n+1} = \|y_{n+1} - y(x_{n+1})\| = O(h^p) \leq K \cdot h^p.$$

From this relation we see that if we double the step h , we expect the new total error to have a new quotient with the old one approximately equal to 2^{-p} .

Based on this observation we can empirically find the order of the method from the behaviour of the total error.

Solve the problem from the table below, write a MATLAB script that solves the initial value problem, with **new.m** for nine different steps ($h = 2^{-k}$), $k = 2, \dots, 10$).

The program should append to a vector the maximum absolute error for the different values of step h .

Then calculate and append to another vector the quotient of the two successive maximum errors.

According to the aforementioned description, what value should the terms of this last vector approximate, and why?

The program should use the **for** command and it should be possible with an alteration of a variable value to change the number of times the problem is solved by splitting the step size in the middle.

Answer 4

According to the theory for a multivariate method of order p we observe the following:

$$GE_{n+1} = \|y_{n+1} - y(x_{n+1})\| = O(h^p) \leq Kh^p$$

By dividing the step size by 2 we obtain a new relation for the global error:

$$GE'_{n+1} = O(h'^p) = O\left(\frac{h^p}{2^p}\right) \leq K' \frac{h^p}{2^p}$$

Further manipulation, of the aforementioned inequalities, yields:

$$\frac{GE'_{n+1}}{GE_{n+1}} = \frac{K'}{K} 2^{-p} = C 2^{-p}$$

The following script involves solving the I.V.P. using the **new.m** method in the interval $[0, \pi/2]$ and with initial value $y(0)=-1$, for step sizes $h = 2^{-k}$, $k = 2, \dots, 10$.

q4.m

```
% Driver program q4.m (Question 4).
```

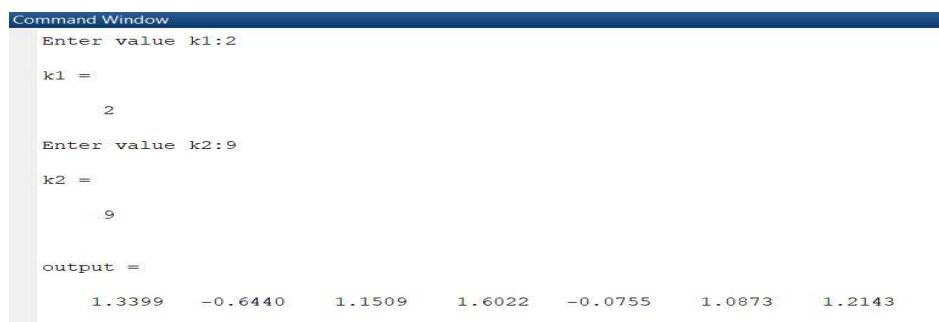
```

% Solve the problem g.m using the new.m method
% use [0,pi/2] as an interval, for step size define variable
% h accordingly and as initial value use y(0)=-1.

% Initialise
clc
t0      = 0;
tfinal  = pi/2;
y0      = -1;
array   = [];
ratios   = [];
% Input
k1 = input('Enter value k1:')
k2 = input('Enter value k2:')
for k=k1:k2
    % Set step size.
    h = 1/(2^k);
    [tout,yout] = new('g',t0,tfinal,h,y0);
    error = (abs(yout-gtrue(tout)));
    l      = length(error);
    temp   = 0;
    % Find index of maximum element in error array.
    for i=1:l
        if error(i)>temp
            temp=error(i);
            index=i;
        end
    end
    % Append maximum element to array.
    array = [array,error(index)];
end
% Calculate ratios of consecutive maximum elements of array.
for i=1:length(array)-1
    ratios(i) = array(i+1)/array(i);
end
% Return approximately the order of new.m method.
output = -log2(ratios)

```

Running the command **q4.m** in the command window and submitting the values 2, 9 for k1, k2 respectively, we obtain:



```

Command Window
Enter value k1:2
k1 =
     2
Enter value k2:9
k2 =
     9
output =
    1.3399   -0.6440    1.1509    1.6022   -0.0755    1.0873    1.2143

```

The vector returned by the **q4.m** script approximates the order of accuracy of the **new.m** method by computing the ratios of the successive maximum absolute errors of the method for each value of step h .