2021

Test 2

AE_5830 DR. HOSDER

MATT PAHAYO

Q1;

A forward difference approximation to the first derivative of a function $f(x)$ at x_i is given by the formula

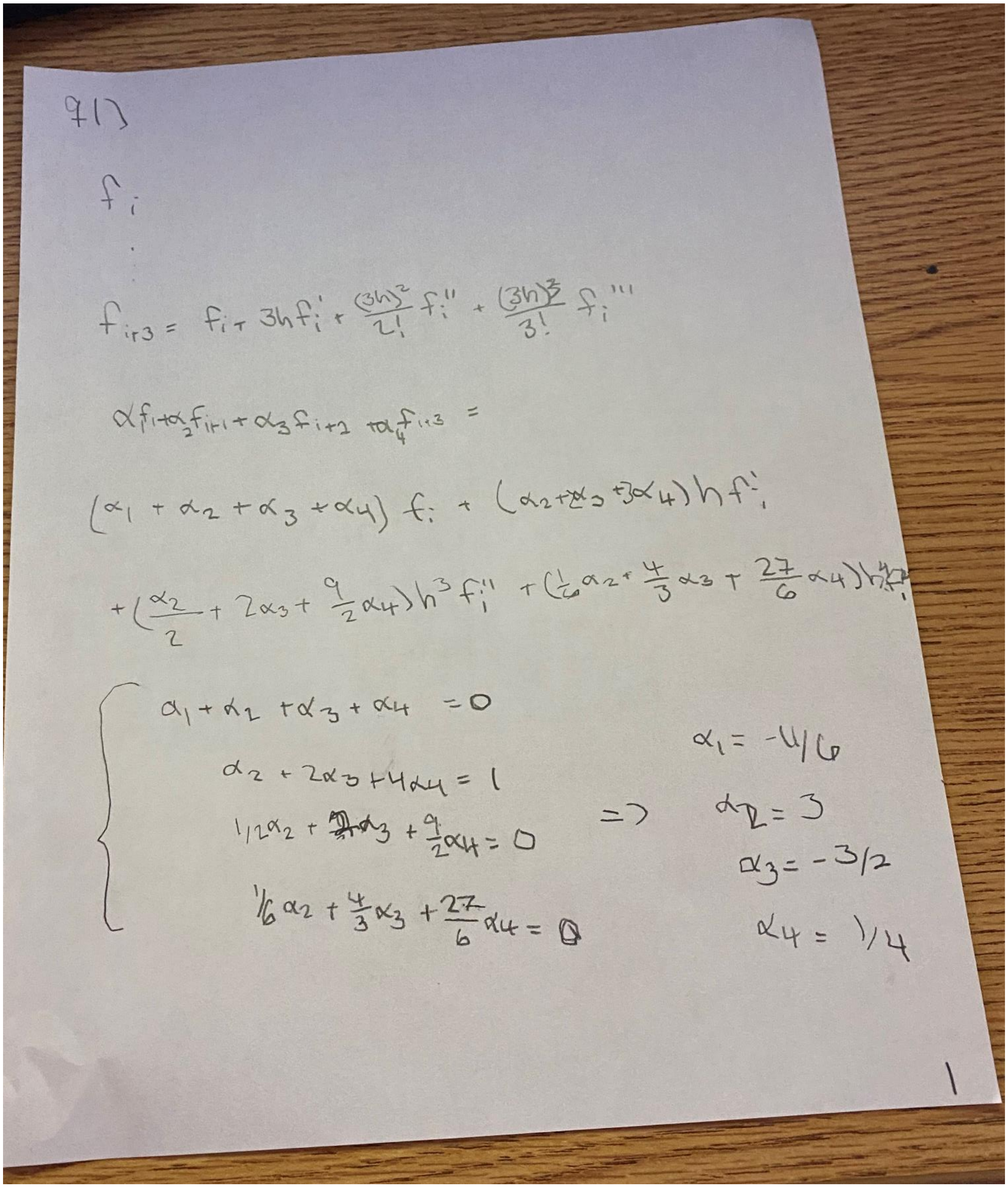
$$\left(\frac{df}{dx}\right)_i = \frac{1}{6\Delta x}\left[a_1 f(x_i) + a_2 f(x_i + \Delta x) + a_3 f(x_i + 2\Delta x) + a_4 f(x_i + 3\Delta x)\right]$$

where Δx is the mesh spacing.
(a) Determine coefficients a_1, a_2, a_3, and a_4 so that the finite difference approximation is 3^{rd} order accurate. You can use Taylor's series expansions, the method of undetermined coefficients or Lagrange polynomial approximation for your derivation, however you should show each step clearly in your calculations.
(b) Use the approximation derived in part (a) to determine the first derivative of the function

$$f(x) = x^2 Sin(x) - 2x$$

at $x_i = -3.0$ for $\Delta x = 0.1$, 0.01, and 0.001. Also calculate the error for each Δx value by using the exact derivative of the function at the given point. **Comment** on the results you have obtained.

(a) fwd_fprimeO3.m



9.1)

f_i

$\vdots$

$$f_{i+3} = f_i + 3hf_i' + \frac{(3h)^2}{2!} f_i'' + \frac{(3h)^3}{3!} f_i'''$$

$$\alpha_1 f_i + \alpha_2 f_{i+1} + \alpha_3 f_{i+2} + \alpha_4 f_{i+3} =$$

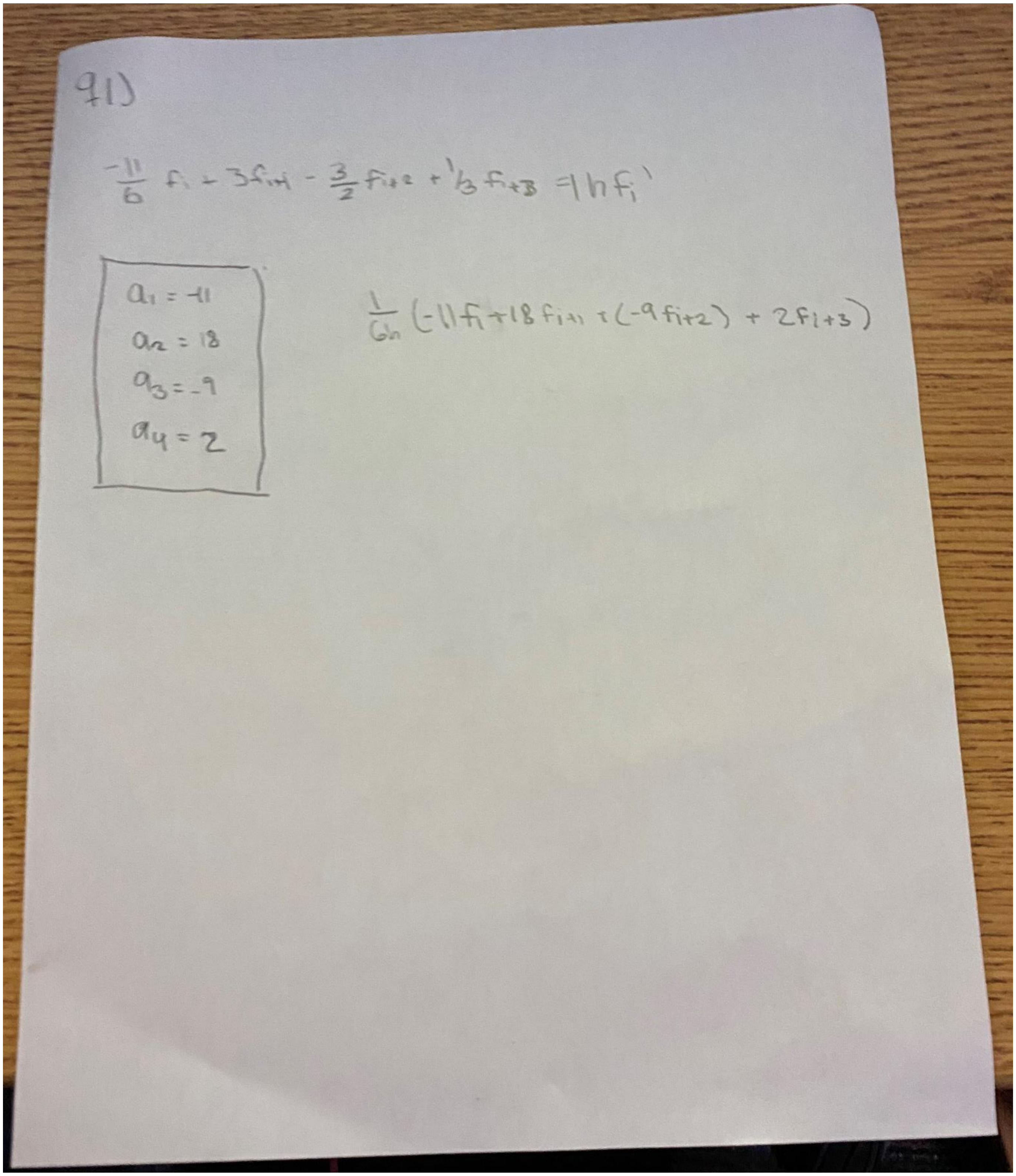
$$(\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4) f_i + (\alpha_2 + 2\alpha_3 + 3\alpha_4) h f_i'$$

$$+ \left(\frac{\alpha_2}{2} + 2\alpha_3 + \frac{9}{2}\alpha_4\right) h^3 f_i'' + \left(\frac{1}{6}\alpha_2 + \frac{4}{3}\alpha_3 + \frac{27}{6}\alpha_4\right) h^4 f_i$$

$$\begin{cases} \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 0 \\ \alpha_2 + 2\alpha_3 + 4\alpha_4 = 1 \\ 1/2\,\alpha_2 + \cancel{2}\alpha_3 + \frac{9}{2}\alpha_4 = 0 \\ 1/6\,\alpha_2 + \frac{4}{3}\alpha_3 + \frac{27}{6}\alpha_4 = 0 \end{cases} \Rightarrow \begin{aligned} \alpha_1 &= -4/6 \\ \alpha_2 &= 3 \\ \alpha_3 &= -3/2 \\ \alpha_4 &= 1/4 \end{aligned}$$

1

Fig, 1 Work for Question 1, page 1



q1)

$$-\frac{11}{6} f_i + 3f_{i+1} - \frac{3}{2} f_{i+2} + \frac{1}{3} f_{i+3} = 1hf_i'$$

$$\boxed{\begin{array}{l} a_1 = -11 \\ a_2 = 18 \\ a_3 = -9 \\ a_4 = 2 \end{array}}$$

$$\frac{1}{6h}\left(-11f_i + 18f_{i+1} + (-9f_{i+2}) + 2f_{i+3}\right)$$

Fig, 1 Work for Question 1, page 2

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \\ 0 & 1/2 & 2 & 9/2 \\ 0 & 1/6 & 4/3 & 27/6 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

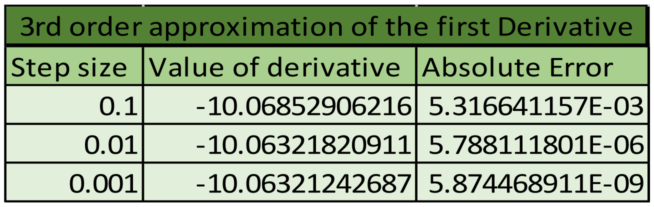
solve with favorite method =>

$$\boxed{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} -11/6 \\ 3 \\ -3/2 \\ 1/3 \end{bmatrix}}$$

=>

$$\boxed{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} -11 \\ 18 \\ -9 \\ 2 \end{bmatrix}}$$

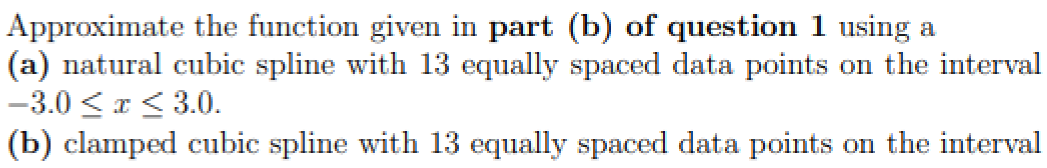
Table 1 question 1 B



3rd order approximation of the first Derivative		
Step size	Value of derivative	Absolute Error
0.1	-10.06852906216	5.316641157E-03
0.01	-10.06321820911	5.788111801E-06
0.001	-10.06321242687	5.874468911E-09

As shown in Table 1 the absolute error significantly decreases when the step size is decreased. (3 orders of magnitude per 1 order of magnitude change in step size).

Q2:



Approximate the function given in **part (b) of question 1** using a
(a) natural cubic spline with 13 equally spaced data points on the interval $-3.0 \le x \le 3.0$.
(b) clamped cubic spline with 13 equally spaced data points on the interval

$-3.0 \leq x \leq 3.0$. For the boundary condition at $x = -3.0$ approximate the first derivative with the finite difference formula derived in question 1 with $\Delta x = 0.5$. For the boundary condition at $x = 3.0$, use the exact value of the derivative.

For each part (part (a) and part (b)): plot the function and the approximation to the function on the same figure. On another figure plot the error distribution. Give the error values at $x = -2.8$, -1.4, 0.0, 1.4, and 2.8 for each case. Using the plots and the error values, **comment** on the performance of the approximations obtained in part (a) and part (b).

Spline.m and gauss.m

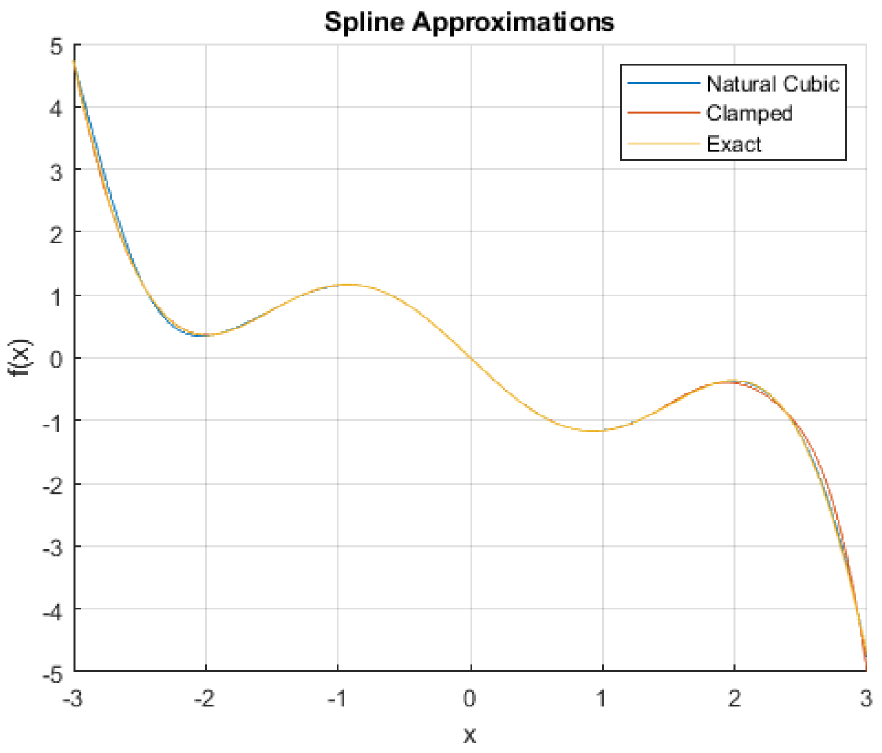
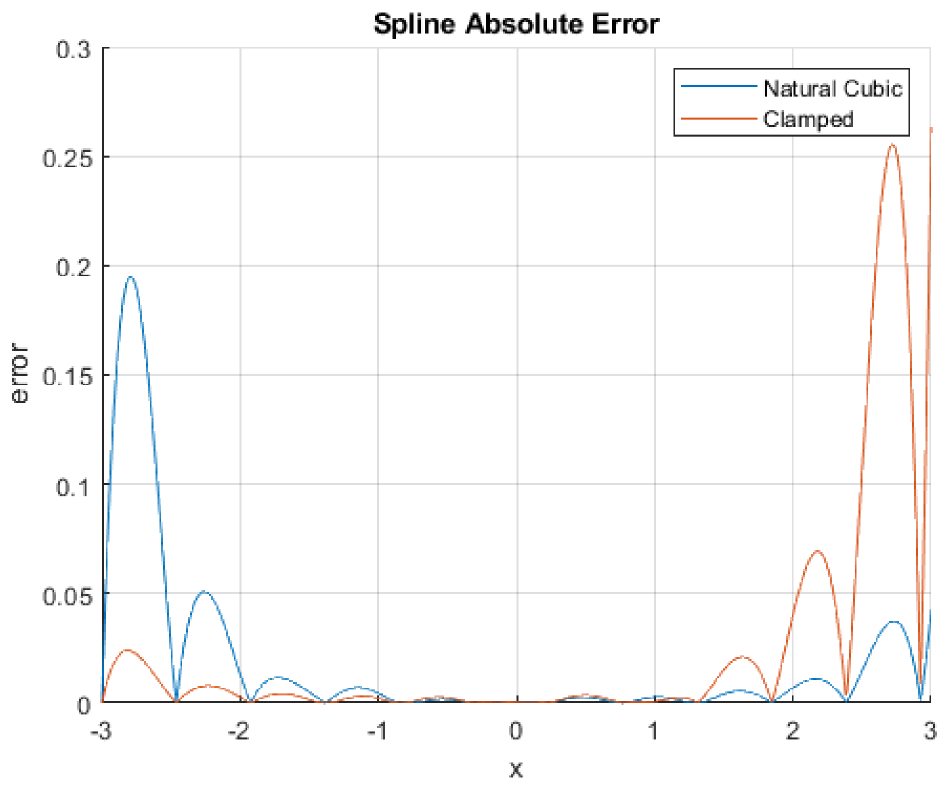


Fig 3. Spline plot



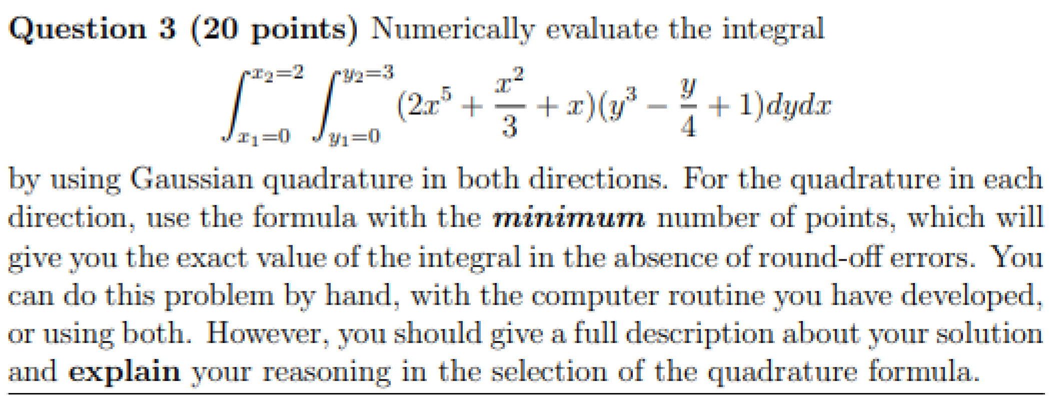
Fig, 4 Spline Error

Table 2 Question 2

Tabulated Spline error					
x	f(x)	NC Approximation	error	Clamped Approximation	error
-2.8	2.973693	3.168418896	0.194725994	2.949798978	0.023893925
-1.4	0.868519	0.869209611	0.000691081	0.868470224	4.83E-05
0	0	0.004362808	0.004362808	0.004930831	0.004930831
1.4	-0.86852	-0.866883809	0.00163472	-1.043067271	0.174548742
2.8	-2.97369	-2.953694023	0.019998879	-2.818552933	0.15513997

The Natural cubic spline has less error at the positive end and more at the negative end than the Clamped spline, so you would choose the type of spline with the least error at the desired location.

Q3:



Question 3 (20 points) Numerically evaluate the integral

$$\int_{x_1=0}^{x_2=2}\int_{y_1=0}^{y_2=3}(2x^5+\frac{x^2}{3}+x)(y^3-\frac{y}{4}+1)dydx$$

by using Gaussian quadrature in both directions. For the quadrature in each direction, use the formula with the ***minimum*** number of points, which will give you the exact value of the integral in the absence of round-off errors. You can do this problem by hand, with the computer routine you have developed, or using both. However, you should give a full description about your solution and **explain** your reasoning in the selection of the quadrature formula.

The exact value of an integral can be achieved using the Gauss quadrature method if the max degree of the function times 2 minus 1is less than or equal to the number of points used in the Gauss quadrature method. The max degree of the function in the y-direction is 3, so the minimum number of points to get the exact value of the integral would be **two**. The minimum points in the x-direction would be **three**.

$$2 * p - 1 \geq \max degree$$

Where p is the point used in the quadrature method.

gaussQuad.m

Table 3 Question 3

Evaluated integral	
Exact	Quadrature 2 to 3
535.9166667	535.9166667

Q4:

Question 4 (30 points)

$$\frac{dy}{dt} = -ty + \frac{4t}{y}, \qquad 0.0 \le t \le 2.5, \qquad y(0) = 1.0$$

Approximate the solution to the above initial value problem using

(a) 4-step Runge-Kutta Method

(b) 2-step Adams-Bashforth Method

(c) Euler Explicit Method

with a time step of $h = 0.1$. For the Adams-Bashforth Method, use starting values obtained from 4-stage Runge-Kutta method. For each method, calculate the error at each time step using the exact solution to this problem:

$$y(t) = \sqrt{4 - 3e^{-t^2}}$$

Show the approximate solutions and the exact function in the same graph. In another graph, plot the error distributions. Note that you may need to give a separate plot for the error of 4-stage Runge-Kutta scheme to see the trend in a larger scale. **Comment** on the performance of each method by specifying possible advantages and/or disadvantages.

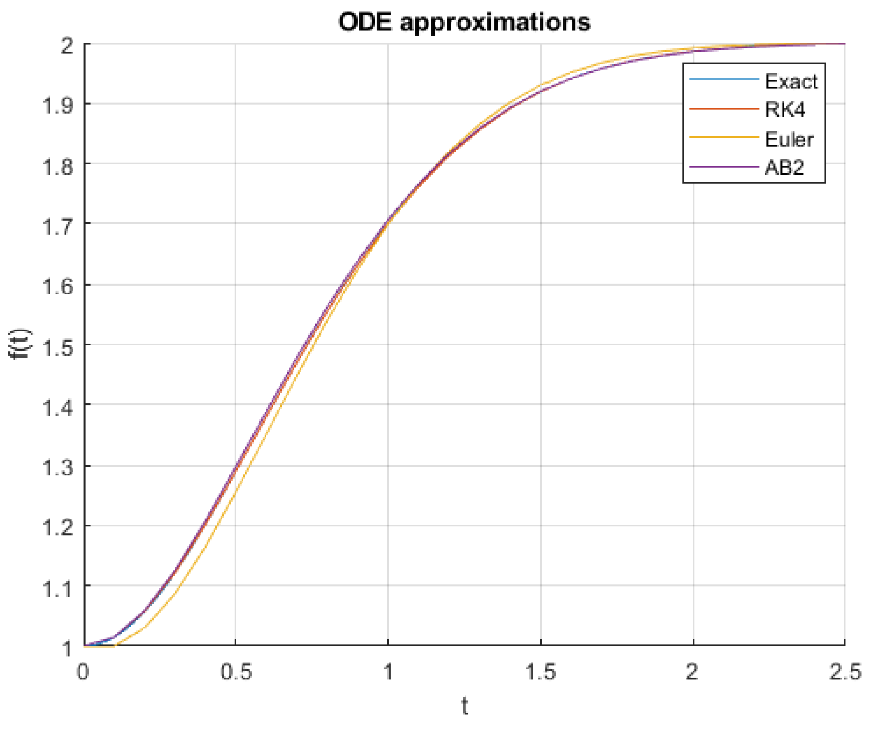


Fig. 5 All ODE Approximations

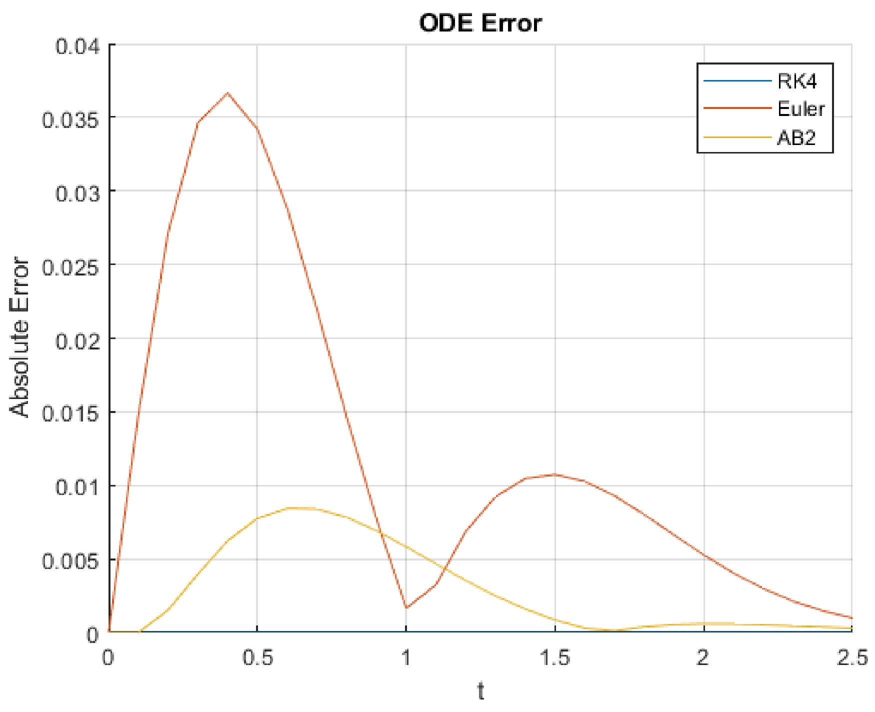


Fig. 6 All Errors

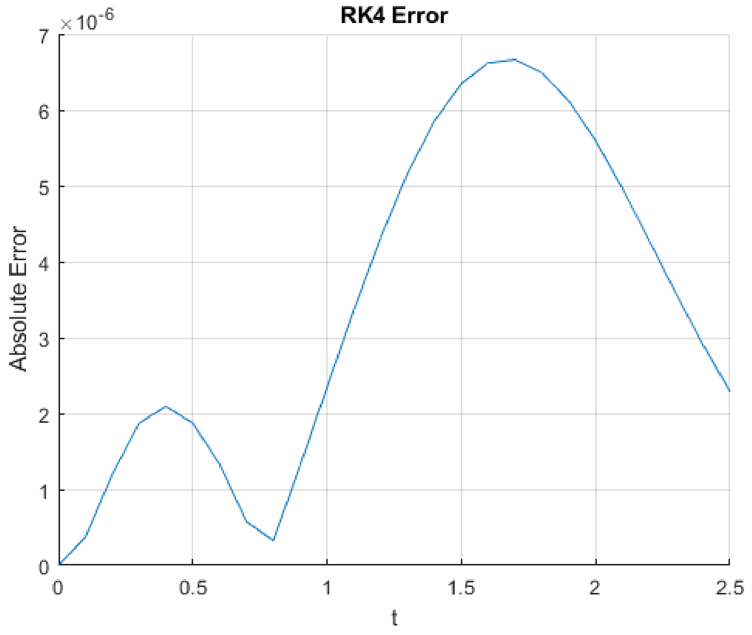


Fig. 7 RK4 Error

Table 4 Question 4

Tabulated Approximations and Errors							
t	exact	RK4	error	AB2	error	Euler	error
0	1	1	0	1	0	1	0
0.1	1.014815	1.014816	3.67E-07	1.014816	3.67E-07	1	0.014815
0.2	1.057181	1.057182	1.19E-06	1.058718	0.001537	1.03	0.027181
0.3	1.121698	1.1217	1.87E-06	1.125667	0.003969	1.08707	0.034628
0.4	1.201486	1.201488	2.09E-06	1.207723	0.006237	1.164846	0.03664
0.5	1.289805	1.289807	1.87E-06	1.297564	0.007758	1.25561	0.034196
0.6	1.380931	1.380933	1.33E-06	1.389363	0.008432	1.352114	0.028817
0.7	1.470415	1.470416	5.74E-07	1.478803	0.008388	1.448487	0.021928
0.8	1.555031	1.555031	3.26E-07	1.562853	0.007821	1.540398	0.014633
0.9	1.632613	1.632612	1.32E-06	1.639528	0.006915	1.624905	0.007708
1	1.70187	1.701868	2.34E-06	1.707692	0.005822	1.700215	0.001655
1.1	1.762217	1.762213	3.36E-06	1.766881	0.004664	1.765458	0.003241
1.2	1.81362	1.813615	4.32E-06	1.817153	0.003533	1.820485	0.006865
1.3	1.856459	1.856454	5.17E-06	1.858955	0.002495	1.865692	0.009233
1.4	1.891408	1.891402	5.85E-06	1.893004	0.001595	1.901869	0.010461
1.5	1.919323	1.919317	6.35E-06	1.920181	0.000857	1.930055	0.010731
1.6	1.941156	1.941149	6.62E-06	1.941443	0.000287	1.951419	0.010263
1.7	1.957874	1.957868	6.66E-06	1.957752	0.000122	1.967158	0.009284
1.8	1.970408	1.970402	6.49E-06	1.970021	0.000387	1.978418	0.008009
1.9	1.979607	1.979601	6.12E-06	1.979073	0.000534	1.98623	0.006622
2	1.986216	1.98621	5.59E-06	1.985627	0.000589	1.991481	0.005265
2.1	1.990863	1.990858	4.96E-06	1.990285	0.000578	1.994896	0.004033
2.2	1.994061	1.994057	4.27E-06	1.993535	0.000525	1.997042	0.002981
2.3	1.996215	1.996212	3.57E-06	1.995764	0.000451	1.998345	0.002129
2.4	1.997635	1.997632	2.90E-06	1.997265	0.00037	1.999106	0.001471
2.5	1.998552	1.998549	2.29E-06	1.99826	0.000291	1.999535	0.000984

An advantage of the RK4 is that it is easy to code, and it is quite accurate. It is not good at discontinuities. The two step Adams-Bashforth method is second most accurate method out of the three. A disadvantage of this is that it is not self-starting, i.e., you need a self-starting method with an order greater than or equal to this method. The Major advantage of Euler's method is that it is the simplest algorithm, but it has the worst accuracy of the three methods; to get more reliable results the step size must be decreased further which may or may not be costly.

RK4.m, AB2.m, and Euler.m

Appendix

```
% main.m
clc
clear all
close all
format longg

%==========================================================================
                                %q1
%==========================================================================

fun = @(x) x^2*sin(x)-2*x;
fp(1,1) = fwd_fprimeO3(fun,-3,.1);
fp(2,1) = fwd_fprimeO3(fun,-3,.01);
fp(3,1) = fwd_fprimeO3(fun,-3,.001);
h(1,1) = .1;
h(2,1) = .01;
h(3,1) = .001;
exact = -10.063212421;
err(1,1) = abs(exact-fp(1,1));
err(2,1) = abs(exact-fp(2,1));
err(3,1) = abs(exact-fp(3,1));

[h fp err]

%==========================================================================
                                %q2
%==========================================================================
f = @(x) x.^2.*sin(x)-2.*x;
syms p
xend = 4;
xstart = -3;
b = xend - xstart;
exact = -10.063212421;
intervals = 13;
n = intervals;
x = transpose(xstart:(xend-xstart)/intervals:xend);
g = f(x);
pClamp = Spline.Clamp(f,g,n,x,xstart,.5,xend,exact);
pNC = (Spline.NC(g,n,x));
NC = sym(zeros(b*100+1,1));
Clamp = NC;

i = xstart ;
j=1;
while(true)
    if i>=-3 && i<xstart+b/n
        NC(j) = subs(pNC(1),p,i);
        Clamp(j) = subs(pClamp(1),p,i);
    end
    if i>=xstart+b/n && i<xstart+2*b/n
        NC(j) = subs(pNC(2),p,i);
        Clamp(j) = subs(pClamp(2),p,i);
    end
    if i>=xstart+2*b/n && i<xstart+3*b/n
```

```
            NC(j) = subs(pNC(3),p,i);
            Clamp(j) = subs(pClamp(3),p,i);
        end
        if i>=xstart+3*b/n && i<xstart+4*b/n
            NC(j) = subs(pNC(4),p,i);
            Clamp(j) = subs(pClamp(4),p,i);
        end
        if i>=xstart+4*b/n && i<xstart+5*b/n
            NC(j) = subs(pNC(5),p,i);
            Clamp(j) = subs(pClamp(5),p,i);
        end
        if i>=xstart+5*b/n && i<xstart+6*b/n
            NC(j) = subs(pNC(6),p,i);
            Clamp(j) = subs(pClamp(6),p,i);
        end
        if i>=xstart+6*b/n && i<xstart+7*b/n
            NC(j) = subs(pNC(7),p,i);
            Clamp(j) = subs(pClamp(7),p,i);
        end
        if i>=xstart+7*b/n && i<xstart+8*b/n
            NC(j) = subs(pNC(8),p,i);
            Clamp(j) = subs(pClamp(8),p,i);
        end
        if i>=xstart+8*b/n && i<xstart+9*b/n
            NC(j) = subs(pNC(9),p,i);
            Clamp(j) = subs(pClamp(9),p,i);
        end
        if i>=xstart+9*b/n && i<xstart+10*b/n
            NC(j) = subs(pNC(10),p,i);
            Clamp(j) = subs(pClamp(10),p,i);
        end
        if i>=xstart+10*b/n && i<xstart+11*b/n
            NC(j) = subs(pNC(11),p,i);
            Clamp(j) = subs(pClamp(11),p,i);
        end
        if i>=xstart+11*b/n && i<xstart+12*b/n
            NC(j) = subs(pNC(12),p,i);
            Clamp(j) = subs(pClamp(12),p,i);
        end
        if i>=xstart+12*b/n && i<xstart+13*b/n
            NC(j) = subs(pNC(13),p,i);
            Clamp(j) = subs(pClamp(13),p,i);
        end
        i=i+.01;
        j =j+1;
        if i>=b
            break;
        end
    end

    dom = transpose(xstart:.01:xend);
    exact = f(dom);
    errNC = abs(NC-exact);
    errClamp = abs(Clamp-exact);

    figure(1)
```

```
hold on
plot(dom,NC)
plot(dom,Clamp)
fplot(f)
xlim([-3 3])
xlabel('x')
ylabel('f(x)')
legend('Natural Cubic','Clamped','Exact')
title('Spline Approximations')
grid on
hold off

figure(2)
hold on
plot(dom,errNC)
plot(dom,errClamp)
xlim([-3 3])
xlabel('x')
ylabel('error')
legend('Natural Cubic','Clamped')
title('Spline Absolute Error')
grid on
hold off

sp = [-2.8;-1.4;0;1.4;2.8];
exactSP = f(sp);

NCsp = double([subs(pNC(1),p, -2.8); subs(pNC(3),p, -1.4);
subs(pNC(7),p,0);subs(pNC(10),p,1.4);subs(pNC(12),p,2.8)])
Clampsp = double([subs(pClamp(1),p, -2.8); subs(pClamp(3),p, -1.4);
subs(pClamp(7),p,0);subs(pClamp(10),p,1.4);subs(pClamp(12),p,2.8)])

errNCsp = abs(exactSP-NCsp)
errClampsp = abs(exactSP-Clampsp)

%=========================================================================
                                %q3
%=========================================================================
syms x % y-direction is going first
f = @(y) (2*x^5+x^2/3+x)*(y^3-y/4+1)
g(x) = gaussQuad(f,0,3,2)
I = gaussQuad(g,0,2,3)
double(I)

%=========================================================================
                                %q4
%=========================================================================

syms t
f(t) = sqrt(4-3*exp(-t^2));
xi = 0;
yi = 1;
h = .1;
steps = 25+1;
```

```
[xRK4,yRK4] = RK4(xi,yi,h,steps);
[xe,ye] = Euler(xi,yi,h,steps);
[xab2,yab2] = AB2(xi,yi,h,steps);

figure (1)
hold on
fplot(f)
plot(xRK4,yRK4)
plot(xe,ye)
plot(xab2,yab2)
xlim([xi 2.5])
grid on
xlabel('t')
ylabel('f(t)')
title('ODE approximations')
legend('Exact','RK4','Euler','AB2')

x = transpose(0:.1:2.5);
exact = zeros(steps,1);
for i = 1:steps
    exact(i,1) = f(x(i,1));
end

errRK4 = abs(yRK4-exact);
errAB2 = abs(yab2-exact);
errE = abs(ye-exact);
tab = [x exact yRK4 errRK4 yab2 errAB2 ye errE];
figure (2)
hold on
plot(xRK4,errRK4)
plot(xe,errE)
plot(xab2,errAB2)
xlim([xi 2.5])
grid on
xlabel('t')
ylabel('Absolute Error')
title('ODE Error')
legend('RK4','Euler','AB2')

figure (4)
hold on
plot(xRK4,errRK4)
xlim([xi 2.5])
grid on
xlabel('t')
ylabel('Absolute Error')
title('RK4 Error')


function dfdx = fwd_fprimeO3(f,x,h)
    dfdx = (-11/6*f(x)+3*f(x+h)-3/2*f(x+2*h)+1/3*f(x+3*h))/h;
end


function I = gaussQuad(f,a,b,n)
```

```
%myFun - Description
%
% Syntax: output = myFun(input)
%
% Long description

switch n
    case 2
        c = ones(2,1);
        t = [-1/sqrt(3);1/sqrt(3)];
    case 3
        c = [5/9;8/9;5/9];
        t = [-sqrt(3/5);0;sqrt(3/5)];
    case 4
        c = [.347855;.652145;.652145;.347855];
        t = [-.861136;-.339981;.339981;.861136];
    otherwise
        disp('nope')
end

x = zeros(n,1);
for i = 1:n
    x(i,1) = (b+a)/2-(b-a)/2*t(i);
end
I = 0;
for i = 1:n
    I = (b-a)/2*c(i)*f(x(i)) + I;
end


classdef Spline
    %UNTITLED Summary of this class goes here
    %   Detailed explanation goes here

    methods (Static)
        function [s] = NC(g,n,x)
            %sysMake will Construct a system for NC spline
            %   it will find the coefficents of each spline and will output
            %   a column vector of the spline equations as syms
            % g is the vector of known points, x is the vector coresponding
            % to the points
            % n is the number of intervals inbetween all points
            syms p
            h = zeros(n,1);
            for i = 1:n-1
                h(i,1) = x(i+1)-x(i);
            end
            A = zeros(n);
            A(1,1) = 1;
            A(n,n) = 1;
            for i = 2:n-1
                A(i,i) = 2*(h(i-1)+h(i));
            end
            for i = 1:n-1
                A(i+1,i) = h(i);
                A(i,i+1) = h(i);
            end
            A(1,2) = 0;
            A(n,n-1) = 0 ;
            b = zeros(n,1);
            for i = 2:n-1
                b(i,1) = 3/h(i)*(g(i+1)-g(i))-3/h(n-1)*(g(i)-g(i-1));
            end
            c = gauss(A,b);
            k = zeros(n,1);
            d = zeros(n,1);
            for i = 1:n-1
                k(i,1) = (g(i+1)-g(i))/h(i)-h(i)/3*(2*c(i)+c(i+1));
                d(i,1) = (c(i+1)-c(i))/3/h(i);
            end
            s = sym(zeros(n,1));
            for i = 1 :n
                s(i,1) = (g(i)+k(i)*(p-x(i))+c(i)*(p-x(i))^2+d(i)*(p-
x(i))^3);
            end
        end

        function [s] = Clamp(f,g,n,x,x0,h0,xn,exact)
            %sysMake will Construct a system for NC spline
            %   it will find the coefficents of each spline and will output
            %   a column vector of the spline equations as syms
            % g is the vector of known points, x is the vector coresponding
            % to the points
            % n is the number of intervals inbetween all points
            syms p
            h = zeros(n,1);
            for i = 1:n-1
                h(i,1) = x(i+1)-x(i);
            end
            A = zeros(n);
            A(1,1) = 2*h(i-1);
            A(n,n) = A(1,1);
            for i = 2:n-1
                A(i,i) = 2*(h(i-1)+h(i));
            end
            for i = 1:n-1
                A(i+1,i) = h(i);
                A(i,i+1) = h(i);
            end
            b = zeros(n,1);
            b(1,1) = 3/h(1)*(g(2)-g(1))-3*fwd_fprimeO3(f,x0,h0);
            b(n,1) = -3/h(end-1)*(g(end)-g(end-1))+3*exact;
            for i = 2:n-1
                b(i,1) = 3/h(i)*(g(i+1)-g(i))-3/h(i-1)*(g(i)-g(i-1));
            end
            c = gauss(A,b);
            k = zeros(n,1);
            d = zeros(n,1);
            for i = 1:n-1
                k(i,1) = (g(i+1)-g(i))/h(i)-h(i)/3*(2*c(i)+c(i+1));
                d(i,1) = (c(i+1)-c(i))/3/h(i);
            end
            s = sym(zeros(n,1));
            for i = 1 :n
                s(i,1) = (g(i)+k(i)*(p-x(i))+c(i)*(p-x(i))^2+d(i)*(p-
x(i))^3);
            end
        end

    end
end


function dydx = Deriv(x,y)
    dydx = -x*y+4*x/y;
%     dxdy = x*exp(3*x)-2*y;
%     dxdy = y/x-(y/x)^2;
end


function [x,y] = RK4(xi,yi,h,steps)
    x = zeros(steps,1);
    y = x;
    x(1,1) = xi ;
    y(1,1) = yi ;
    for i = 2:steps
        x(i,1) = x(i-1) + h;
        k1 = h*Deriv(x(i-1),y(i-1));
        k2 = h*Deriv(x(i-1)+h/2,y(i-1)+k1/2);
        k3 = h*Deriv(x(i-1)+h/2,y(i-1)+k2/2);
        k4 = h*Deriv(x(i-1)+h,y(i-1)+k3);
        y(i) = y(i-1) + 1/6*(k1+2*k2+2*k3+k4);
    end
end


function [x,y] = Euler(xi,yi,h,steps)
    x = zeros(steps,1);
    y = x;
    dydx = x;
    x(1,1) = xi;
    y(1,1) =yi;
    for i = 2:steps
        x(i,1) = x(i-1) + h;
        dydx(i,1) = Deriv(x(i-1),y(i-1));
        y(i,1) = y(i-1)+h*dydx(i,1);
    end
end


function [x,y] = AB2(xi,yi,h,steps)
    % Rk4
    x = zeros(steps,1);
    y = x;
    x(1,1) = xi ;
    y(1,1) = yi ;
    for i = 2:steps
        x(i,1) = x(i-1) + h;
        k1 = h*Deriv(x(i-1),y(i-1));
        k2 = h*Deriv(x(i-1)+h/2,y(i-1)+k1/2);
        k3 = h*Deriv(x(i-1)+h/2,y(i-1)+k2/2);
        k4 = h*Deriv(x(i-1)+h,y(i-1)+k3);
        y(i) = y(i-1) + 1/6*(k1+2*k2+2*k3+k4);
    end

    %AB2
    for i = 3:steps
        x(i,1) = x(i-1) + h;
        y(i) = y(i-1) + h*(3/2*Deriv(x(i-1),y(i-1))-1/2*Deriv(x(i-2),y(i-
2)));
    end
end


function [x] = gauss(a,b)
% gauss elimination

n = length(a);

k = 1 ;
p = k ;
big = abs(a(k,k));

%****************************************************
% pivoting portion
%****************************************************
for ii=k+1:n
    dummy = abs(a(ii,k));
    if dummy > big
       big = dummy;
       p = ii ;
    end
end
if p ~= k
    for jj = k:n
        dummy = a(p,jj);
        a(p,jj) = a(k,jj);
        a(k,jj) = dummy;
    end
    dummy = b(p);
    b(p)=b(k);
    b(k) = dummy;
end

%****************************************************
% elimination step
%****************************************************
for k=1:(n-1)
    for i=k+1:n
        factor = a(i,k)/a(k,k);
       for j=k+1:n
           a(i,j) = a(i,j) - factor*a(k,j);
       end
       b(i) = b(i) - factor*b(k);
    end
```

```
end

%**************************************************
% back substitution
%**************************************************
x(n,1) = b(n)/a(n,n);
for i = n-1:-1:1
    sum = b(i);
   for j = i + 1:n
       sum = sum - a(i,j)*x(j,1);
   end
   x(i,1) = sum/a(i,i);
end
end
```