NUMERICAL METHODS FOR ODE

MC317 Lab

ANEESH PANCHAL

2K20/MC/021

Department of Applied Mathematics,

Delhi Technological University

Submitted To –

Dr. Vivek K. Agarwal and Ms. Kirti

INDEX

S. No.	Experiment	Date	Sign & Remark
01.	To solve simultaneous ordinary differential equations using dsolve and ode45 methods.	12/Aug/2022	
02.	Implement Euler's method for solving differential equations.	26/Aug/2022	
03.	Implement Runge Kutta (RK) method of order 2 for solving differential equations.	02/Sept/2022	
04.	Implement Runge Kutta (RK) method of order 4 for solving differential equations.	09/Sept/2022	
05.	Implement Predictor Corrector method for solving differential equations.	16/Sept/2022	
06.	Implement Routh Hurwitz Criteria for Stability Analysis of a polynomial.	14/Oct/2022	
07.	Solving 2^{nd} Order Boundary Value problem with Dirichlet Boundary Condition.	21/Oct/2022	
08.	Solving 2^{nd} Order Boundary Value problem with Neumann Boundary Condition.	28/Oct/2022	
09.	Solving 2^{nd} Order Boundary Value problem with Robin (Mixed) Boundary Condition.	04/Nov/2022	

Experiment 1

Aim:

To solve simultaneous ordinary differential equations using dsolve and ode45 methods.

Code:

```
syms x y(x)

%Linear Differential Equations
%y' +2y = sin(x)
%y(0) = 0
eqn1 = diff(y,x) + 2*y - sin(x) == 0;
initcond = y(0)==0;
sol1 = dsolve(eqn1,initcond)

%y'' + xy' + y = 0
%y(0) = 0, y'(0) = 0
eqn2 = diff(y,x,2) + x*diff(y,x) + y == 0;
ny = diff(y,x);
cond = [y(0)==1, ny(0)==0];
sol2 = dsolve(eqn2,cond)

%Non Linear Differential Equations
%(y' + y)^2 = 1
%y(0) = 0
eqn3 = (diff(y,x) + y)^2 ==1;
sol3 = dsolve(eqn3,initcond)

%Non Linear Simultaneous Differential Equations using dsolve
syms t a(t) b(t)
eqn41 = 2*diff(a) - 2*diff(b) - 3*a == t;
eqn42 = 2*diff(a) + 2*diff(b) + 3*a + 8*b == 2;
eqn4 = [eqn41,eqn42];
condn = [a(0)==1, b(0)==0];
sol4 = dsolve(eqn4,condn);
sol4a = sol4.a
sol4b = sol4.b

%Non Linear Simultaneous Differential Equations using ode45
f = @(t,Y)[Y(1)-Y(1)*Y(2);Y(1)*Y(2)-Y(2)];
[tsol,ysol] = ode45(f,[0:0.1:40],[5,6]);
plot(tsol,ysol)
xlabel("Time (t)")
ylabel("Population")
legend("Prey (x)","Predator (y)")
```

Output:

>> Solve_Diff_Eqn

sol1 =

exp(-2*x)/5 - (5^(1/2)*cos(x + atan(2)))/5 sol2 =

exp(-x^2/2)

sol3 =

exp(-x) - 1
1 - exp(-x)

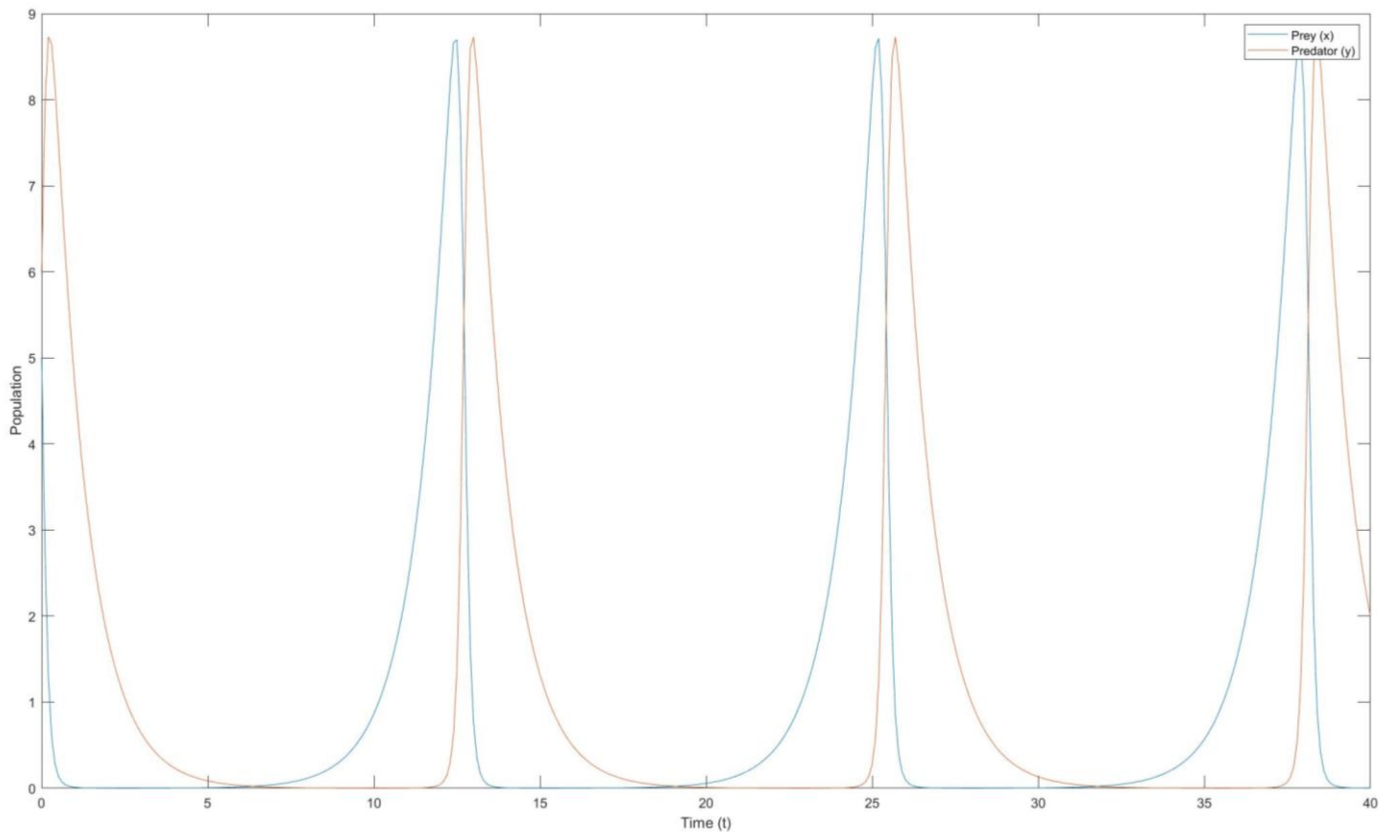
sol4a =

- 2*exp(t)*((exp(-t)*(5*t + 7))/32 - 19/32) - (2*exp(-3*t)*((exp(3*t)*(3*t - 19))/96 - 17/96))/3

sol4b =

exp(t)*((exp(-t)*(5*t + 7))/32 - 19/32) - exp(-3*t)*((exp(3*t)*(3*t - 19))/96 - 17/96)

>>



Experiment 2

Aim:

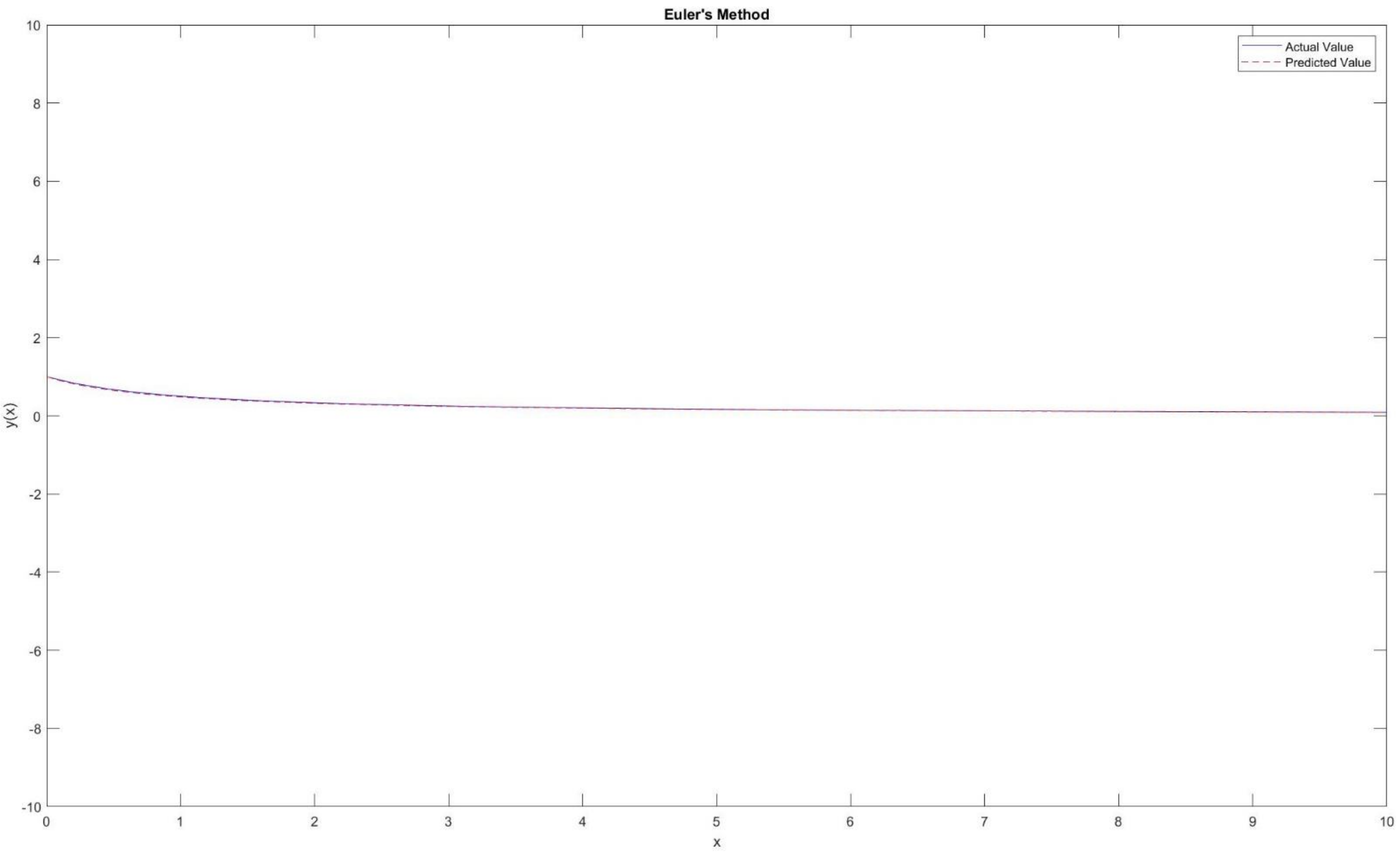
Implement Euler's method for solving differential equations.

Code:

```
syms f(x,y) func(x)

%f(x,y)=-y*y;
% x0=0 y0=1
f(x,y) = -y*y;
x0 = 0;
y0 = 1;

func(x) = 1/(x+1);

n = 100;
b = 10;
a = 0;
h = (b-a)/n;

ysol(1) = y0;
xsol(1) = x0;
yactual(1) = y0;
for i=1:n
    xsol(i+1) = x0 + i*h;
    ysol(i+1) = ysol(i) + h*f(xsol(i),ysol(i));
    yactual(i+1) = func(xsol(i+1));
end

%Root Mean Square Error
diff = (ysol-yactual).^2;
RMSE_Euler = sqrt(mean(diff))

figure
fplot(func,'b');
hold on
plot(xsol,ysol,'--r');
xlim([0 10])
title("Euler's Method")
xlabel("x")
ylabel("y(x)")
legend("Actual Value","Predicted Value")
hold off
```

Output:

```
>> Euler

RMSE_Euler =

   0.0089
```



Euler's Method
Actual Value
Predicted Value
10
8
6
4
2
0
-2
-4
-6
-8
-10
y(x)
0
1
2
3
4
5
6
7
8
9
10
x

Experiment 3

Aim:

Implement Runge Kutta (RK) method of order 2 for solving differential equations.

Code:

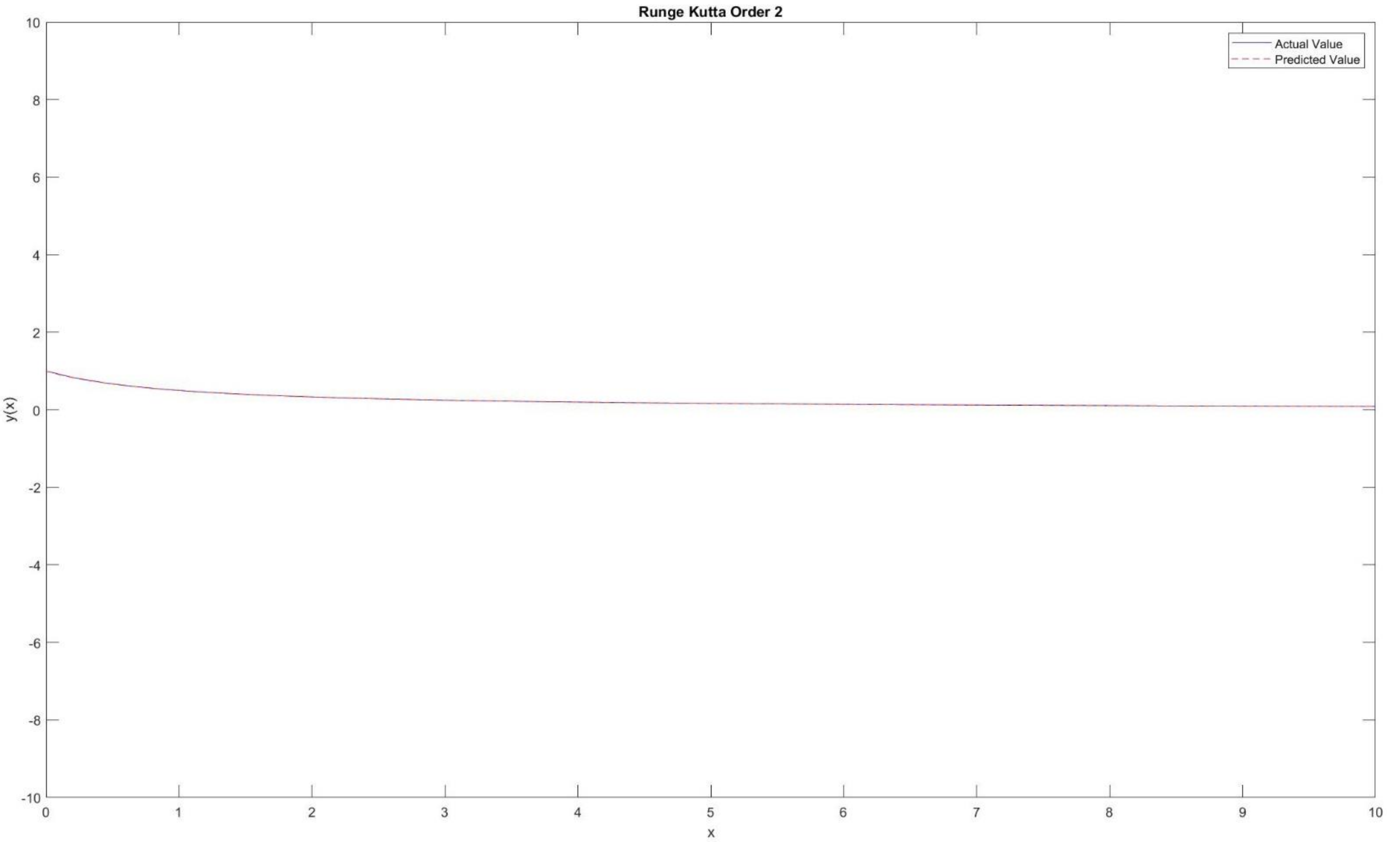
```
syms f(x,y) func(x)

%f(x,y)=-y*y;
% x0=0 y0=1
f(x,y) = -y*y;
x0 = 0;
y0 = 1;

func(x) = 1/(x+1);

n = 100;
b = 10;
a = 0;
h = (b-a)/n;

ysol(1) = y0;
xsol(1) = x0;
yactual(1) = y0;
for i=1:n
    k1=vpa(h*f(xsol(i),ysol(i)));
    k2=vpa(h*f(xsol(i)+((2*h)/3),ysol(i)+((2*k1)/3)));
    k=(1/4)*(k1+3*k2);
    ysol(i+1)=ysol(i)+k;
    xsol(i+1) = x0 + i*h;
    yactual(i+1) = func(xsol(i+1));
end

%Root Mean Square Error
diff = (ysol-yactual).^2;
RMSE_RK2 = sqrt(mean(diff))

figure
fplot(func,'b');
hold on
plot(xsol,ysol,'--r');

xlim([0 10])
title("Runge Kutta Order 2")
xlabel("x")
ylabel("y(x)")
legend("Actual Value","Predicted Value")
hold off
```

Output:

>> RungeKutta2nd

RMSE_RK2 =

4.2778e-04



Experiment 4

Aim:

Implement Runge Kutta (RK) method of order 4 for solving differential equations.

Code:

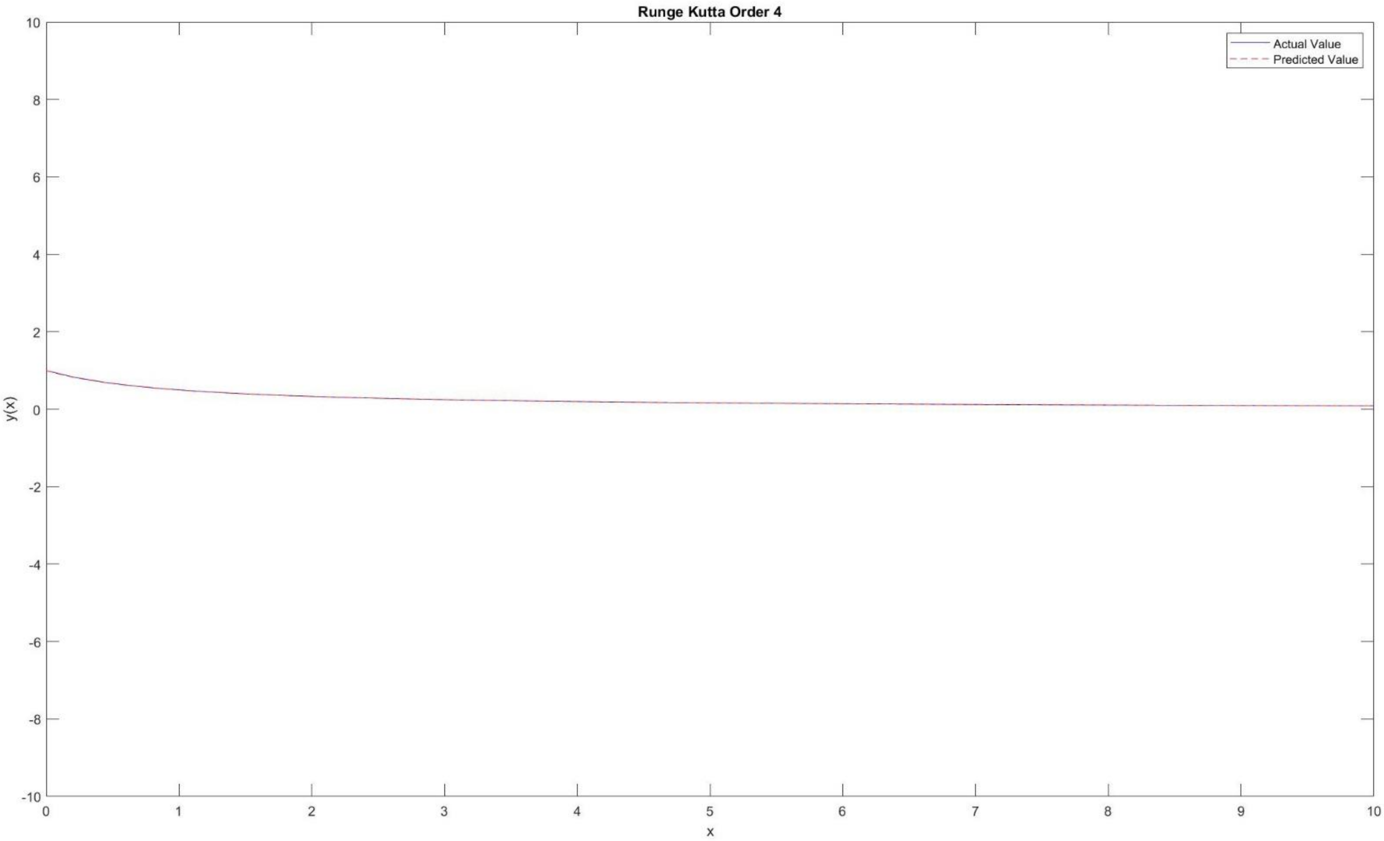
```
syms f(x,y) func(x)

%f(x,y)=-y*y;
% x0=0 y0=1
f(x,y) = -y*y;
x0 = 0;
y0 = 1;

func(x) = 1/(x+1);

n = 100;
b = 10;
a = 0;
h = (b-a)/n;

ysol(1) = y0;
xsol(1) = x0;
yactual(1) = y0;
for i=1:n
    k1=vpa(h*f(xsol(i),ysol(i)));
    k2=vpa(h*f(xsol(i)+(h/2),ysol(i)+(k1/2)));
    k3=vpa(h*f(xsol(i)+(h/2),ysol(i)+(k2/2)));
    k4=vpa(h*f(xsol(i)+h,ysol(i)+k3));
    k=(1/6)*(k1+2*(k2+k3)+k4);
    ysol(i+1)=ysol(i)+k;
    xsol(i+1) = x0 + i*h;
    yactual(i+1) = func(xsol(i+1));
end

%Root Mean Square Error
diff = (ysol-yactual).^2;
RMSE_RK4 = sqrt(mean(diff))

figure
fplot(func,'b');
hold on
plot(xsol,ysol,'--r');
xlim([0 10])
title("Runge Kutta Order 4")
xlabel("x")
ylabel("y(x)")
legend("Actual Value","Predicted Value")
hold off
```

Output:

>> RungeKutta4th

RMSE_RK4 =

1.4224e-07



Experiment 5

Aim:

Implement Predictor Corrector method for solving differential equations.

Code:

```
syms f(x,y)

f(x,y) = 2*x*(y-1);
x0 = 0;
y0 = 0;

%Actual Function
func = 1 - exp(x^2);

b = 0.4;
a = 0;
h = (b-a)/4;
err = 0.000000005;

%Values using 4th order RK Method
ysol(1) = y0;
xsol(1) = x0;
fun(1) = f(xsol(1),ysol(1));
for i = 1:4
    k1 = vpa(h*f(xsol(i),ysol(i)));
    k2 = vpa(h*f(xsol(i)+(h/2),ysol(i)+(k1/2)));
    k3 = vpa(h*f(xsol(i)+(h/2),ysol(i)+(k2/2)));
    k4 = vpa(h*f(xsol(i)+h,ysol(i)+k3));
    k = (1/6)*(k1+2*(k2+k3)+k4);
    ysol(i+1) = ysol(i)+k;
    xsol(i+1) = x0 + i*h;
    fun(i) = f(xsol(i),ysol(i));
end

%Predictor
ysol(5) = ysol(1) + ((4*h)/3)*(2*fun(4) - fun(3) + 2*fun(2));
Predicted_Value = vpa(ysol(5))

%Corrector
yc(1) = ysol(5);
fc(1) = f(xsol(5),ysol(5));
i = 1;
while true
    yc(i+1) = ysol(3) + (h/3)*(fc(i) + 4*fun(4) + fun(3));
    fc(i+1) = f(xsol(5),yc(i+1));
    if(yc(i+1)-yc(i) < err)
        if(yc(i)-yc(i+1) < err)
            break
        end
    end
    i = i+1;
end
```

```
Corrected_Value = vpa(yc(i+1))
Number_of_Corrections = i
Actual_Value = vpa(subs(func,x,xsol(5)))
```

Output:

```
>> PredCorrect

Predicted_Value =

-0.17342731697800903289952145769348


Corrected_Value =

-0.17351594646638596186605241200596


Number_of_Corrections =

     4


Actual_Value =

-0.17351087099181023501861108689197

>>
```

Experiment 6

Aim:

Implement Routh Hurwitz Criteria for Stability Analysis of a polynomial.

Code:

```
syms x
f = input('Input the polynomial: ');
deg = polynomialDegree(f);
coeff(1) = subs(f,x,0);
diff_f = f;

%Obtain Coefficient array for polynomial
for i=1:deg
    diff_f = diff(diff_f,x)/i;
    coeff(i+1) = subs(diff_f,x,0);
end

%Putting a_0 +ve
if coeff(deg+1)<0
    coeff = -coeff;
end

%Obtain Routh Hurwitz Matrix
coeff_matrix = zeros(deg);
r = 1; c = 1;
for i=1:2:2*deg-1
    for j=i:-1:0
        if c==deg+1
            break
        end
        if (deg+1-j)<=0
            coeff_matrix(c,r) = 0;
        else
            coeff_matrix(c,r) = coeff(deg+1-j);
        end
        c = c+1;
    end
    r = r+1;
    c = 1;
end
coeff_matrix

%Stability using Routh Hurwitz Criteria
if det(coeff_matrix)>0
    fprintf('Stable System\n\n');
else
    fprintf('Unstable System\n\n');
end
```

Output:

>> Routh

Input the polynomial: x^7 + 3*x^6 + 2*x^5 + 5*x^3 + 6*x^2 + 3*x + 1

coeff_matrix =

```
     3     0     6     1     0     0     0
     1     2     5     3     0     0     0
     0     3     0     6     1     0     0
     0     1     2     5     3     0     0
     0     0     3     0     6     1     0
     0     0     1     2     5     3     0
     0     0     0     3     0     6     1
```

Unstable System

>>

Experiment 7

Aim:

Solving 2nd Order Boundary Value problem with Dirichlet Boundary Condition.

Code:

```
syms u(x) a(x) b(x) f(x) p(i) q(i) r(i)
eqn = diff(u,x,2) + (x^2 + 1)*diff(u,x) + u*sin(x) == exp(x);
a(x) = x^2 + 1;
b(x) = sin(x);
f(x) = exp(x);

%Dirichlet Boundary Conditions
%u(0) = 0 and u(1) = 1
h = 0.1;
xa = 0;
xb = 1;
n = (xb-xa)/h;
ua = 0;
ub = 1;

TriMat = eye(n-1);
CoeffMat(1) = 1;

r(i) = (1/(h^2)) - (a(i)/(2*h));
p(i) = b(i) - (2/(h^2));
q(i) = (1/(h^2)) + (a(i)/(2*h));

for j=1:n-1
    if j == 1
        TriMat(j,1) = p(j);
        TriMat(j,2) = q(j);
        CoeffMat(j) = f(j) - r(j)*ua;
    elseif j == n-1
        TriMat(j,n-2) = r(j);
        TriMat(j,n-1) = p(j);
        CoeffMat(j) = f(j) - q(j)*ub;
    else
        TriMat(j,j) = p(j);
        TriMat(j,j+1) = q(j);
        TriMat(j,j-1) = r(j);
        CoeffMat(j) = f(j);
    end
end

Coefficient_Matrix = CoeffMat'

%TriDiagonal Matrix is the Derivative Matrix
Tri_Diagonal_Matrix = TriMat

Solution_values = (inv(TriMat)*CoeffMat')
```

Output:

```
Command Window
>> DirichletBC

Coefficient_Matrix =

   1.0e+03 *

    0.0027
    0.0074
    0.0201
    0.0546
    0.1484
    0.4034
    1.0966
    2.9810
    7.5931


Tri_Diagonal_Matrix =

 -199.1585  110.0000         0         0         0         0         0         0         0
   75.0000 -199.0907  125.0000         0         0         0         0         0         0
         0   50.0000 -199.8589  150.0000         0         0         0         0         0
         0         0   15.0000 -200.7568  185.0000         0         0         0         0
         0         0         0  -30.0000 -200.9589  230.0000         0         0         0
         0         0         0         0  -85.0000 -200.2794  285.0000         0         0
         0         0         0         0         0 -150.0000 -199.3430  350.0000         0
         0         0         0         0         0         0 -225.0000 -199.0106  425.0000
         0         0         0         0         0         0         0 -310.0000 -199.5879


Solution_values =

   -8.9762
  -16.2271
  -20.4004
  -21.6384
  -21.5322
  -20.9906
  -19.7571
  -17.1154
  -11.4601

fx >>
                                                    UTF-8      script      Ln 17   Col 23
```

Experiment 8

Aim:

Solving 2nd Order Boundary Value problem with Neumann Boundary Condition.

Code:

```
syms u(x) a(x) b(x) f(x) p(i) q(i) r(i)
eqn = diff(u,x,2) + (x^2 + 1)*diff(u,x) + u*sin(x) == exp(x);
a(x) = x^2 + 1;
b(x) = sin(x);
f(x) = exp(x);

%Neumann Boundary Conditions
%u(0) = 0 and u'(1) = -1
h = 0.1;
xa = 0;
xb = 1;
n = (xb-xa)/h;
ua = 0;
diff_ub = -1;
alpha = diff_ub;

TriMat = eye(n-1);
CoeffMat(1) = 1;

r(i) = (1/(h^2)) - (a(i)/(2*h));
p(i) = b(i) - (2/(h^2));
q(i) = (1/(h^2)) + (a(i)/(2*h));

for j=1:n-1
    if j == 1
        TriMat(j,1) = p(j);
        TriMat(j,2) = q(j);
        CoeffMat(j) = f(j) - r(j)*ua;
    elseif j == n-1
        TriMat(j,n-2) = r(j);
        TriMat(j,n-1) = p(j) + q(j);
        CoeffMat(j) = f(j) - q(j)*alpha*h;
    else
        TriMat(j,j) = p(j);
        TriMat(j,j+1) = q(j);
        TriMat(j,j-1) = r(j);
        CoeffMat(j) = f(j);
    end
end

Coefficient_Matrix = CoeffMat'

%TriDiagonal Matrix is the Derivative Matrix
Tri_Diagonal_Matrix = TriMat

Solution_values = (inv(TriMat)*CoeffMat')
```

Output:

```
Command Window
>> NeumannBC

Coefficient_Matrix =

   1.0e+03 *

    0.0027
    0.0074
    0.0201
    0.0546
    0.1484
    0.4034
    1.0966
    2.9810
    8.1541


Tri_Diagonal_Matrix =

 -199.1585  110.0000         0         0         0         0         0         0         0
   75.0000 -199.0907  125.0000         0         0         0         0         0         0
         0   50.0000 -199.8589  150.0000         0         0         0         0         0
         0         0   15.0000 -200.7568  185.0000         0         0         0         0
         0         0         0  -30.0000 -200.9589  230.0000         0         0         0
         0         0         0         0  -85.0000 -200.2794  285.0000         0         0
         0         0         0         0         0 -150.0000 -199.3430  350.0000         0
         0         0         0         0         0         0 -225.0000 -199.0106  425.0000
         0         0         0         0         0         0         0 -310.0000  310.4121


Solution_values =

   1.0e+06 *

    0.4820
    0.8727
    1.1007
    1.1757
    1.1866
    1.1901
    1.1902
    1.1880
    1.1864

fx >>
UTF-8    script    Ln  34   Col  12
```

Experiment 9

Aim:

Solving 2nd Order Boundary Value problem with Robin (Mixed) Boundary Condition.

Code:

```
syms u(x) a(x) b(x) f(x) p(i) q(i) r(i)
eqn = diff(u,x,2) + (x^2 + 1)*diff(u,x) + u*sin(x) == exp(x);
a(x) = x^2 + 1;
b(x) = sin(x);
f(x) = exp(x);

%Mixed Boundary Conditions
%u(0) + u'(0) = 1(alpha) & u(1) + u'(1) = 0(beta)
h = 0.1;
xa = 0;
xb = 1;
n = (xb-xa)/h;
alpha = 1;
beta = 0;

TriMat = eye(n-1);
CoeffMat(1) = 1;

r(i) = (1/(h^2)) - (a(i)/(2*h));
p(i) = b(i) - (2/(h^2));
q(i) = (1/(h^2)) + (a(i)/(2*h));

for j=1:n-1
    if j == 1
        TriMat(j,1) = p(j) + ((r(j))/(1-h));
        TriMat(j,2) = q(j);
        CoeffMat(j) = f(j) + (r(j)*alpha*h)/(1-h);
    elseif j == n-1
        TriMat(j,n-2) = r(j);
        TriMat(j,n-1) = p(j) + ((q(j))/(1+h));
        CoeffMat(j) = f(j) - (q(j)*beta*h)/(1+h);
    else
        TriMat(j,j) = p(j);
        TriMat(j,j+1) = q(j);
        TriMat(j,j-1) = r(j);
        CoeffMat(j) = f(j);
    end
end

Coefficient_Matrix = CoeffMat'

%TriDiagonal Matrix is the Derivative Matrix
Tri_Diagonal_Matrix = TriMat

Solution_values = (inv(TriMat)*CoeffMat')
```

Output:

```
Command Window
>> MixedBC

Coefficient_Matrix =

   1.0e+03 *

    0.0127
    0.0074
    0.0201
    0.0546
    0.1484
    0.4034
    1.0966
    2.9810
    8.1031


Tri_Diagonal_Matrix =

  -99.1585  110.0000         0         0         0         0         0         0         0
   75.0000 -199.0907  125.0000         0         0         0         0         0         0
         0   50.0000 -199.8589  150.0000         0         0         0         0         0
         0         0   15.0000 -200.7568  185.0000         0         0         0         0
         0         0         0  -30.0000 -200.9589  230.0000         0         0         0
         0         0         0         0  -85.0000 -200.2794  285.0000         0         0
         0         0         0         0         0 -150.0000 -199.3430  350.0000         0
         0         0         0         0         0         0 -225.0000 -199.0106  425.0000
         0         0         0         0         0         0         0 -310.0000  264.0485


Solution_values =

 -181.3608
 -163.3705
 -151.3288
 -147.0386
 -146.9971
 -146.9701
 -145.7067
 -142.8414
 -137.0118

fx >>
UTF-8    script    Ln 44 Col 29
```