Alexandra University **جامعة الإسكندرية**

Faculty of engineering **كلية هندسة**

Communication program **البرنامج اتصالات**

2ND term 2021/2022 **الفصل الدراسي الثاني**

Computational Mathematics **الرياضيات الحسابية**

(EMP 219)

Numerical Method Report

Department: Communication and Electronics Engineering

Level: 2

#	الاسم	الرقم الجامعي
1	مصطفي سيد احمد طه احمد	19016660
2	مروان محمد احمد ابو العلا	19016615
3	محمد رزق امين محمد	19016365
4	هيثم عطية السيد	19016857
5	هيثم احمد شعبان	19016856
6	ساهر طارق أنور زايد أحمد	19015763
7	مروان مجدي مرزوق	19016614
8	هشام هيثم عبدالوهاب	19016850

Solving Linear System

Jacobi method

Example:

Example: Use the Jacobi iterative method to solve

$$\begin{aligned} 3x_1 - 0.1x_2 - 0.2x_3 &= 7.85 \\ 0.1x_1 + 7x_2 - 0.3x_3 &= -19.3 \\ 0.3x_1 - 0.2x_2 + 10x_3 &= 71.4 \end{aligned}$$

Use 3 iterations and 5 decimal places in your calculations.

Code:

```
%%
% Jacobi Method
clc;
clear;
format LONG;
A = zeros(3,3); % coefficients matrix
B = zeros(1,3); % RHS matrix
for i = 1 :3
    for j = 1:3
     A(i,j) = input(['Enter a' num2str(i) num2str(j) ' = ' ]); % a loop for the user to input both As and Bs to the matrix created earlier
    end
    B(i,1) = input(['Enter b' num2str(i) ' = ' ]);            % a loop for the user to input both As and Bs to the matrix created earlier
end

x1 = input('Enter x1 = ');
x2 = input('Enter x2 = ');
x3 = input('Enter x3 = '); %in initial guess size(X) = size(B)


n = input('Enter Number of ilterations = '); % Maximum iteration number

%Method_____________________________________________________________
```

```
%Method_____________________________________________________________
sum = 0;
flag = zeros(1,3);
k =  0;
A_B = [A B];
A_B_new =  A_B;
while( k < 6)
    A = A_B_new([1 2 3],[1 2 3]);
for i = 1:3
   sum = 0;
    for j = 1 : 3
        if(i == j)
            continue;
        end
        sum = sum + abs(A(i,j));
    end
    if(abs(A(i,i)) > sum)
        flag(1,i) = 1;
    end
end
if(flag == [1 1 1] )
    break;
end
k = k+1;
switch(k)
    case 1
        A_B_new([1 2 3],:) = A_B([1 3 2],:);
    case 2
        A_B_new([1 2 3],:) = A_B([2 1 3],:);
```

```
    case 2
        A_B_new([1 2 3],:) = A_B([2 1 3],:);
    case 3
        A_B_new([1 2 3],:) = A_B([2 3 1],:);
    case 4
        A_B_new([1 2 3],:) = A_B([3 2 1],:);
    case 5
        A_B_new([1 2 3],:) = A_B([3 1 2],:);
end

if(k == 6)
    fprintf('Matrix cant be strictly dominant\n');
end
end
A = A_B_new(1:3,1:3);
B = A_B_new(:,4);
%Output_____________________________________________________
fprintf('iteration\tx1\tx2\tx3\n');
fprintf('-----------------------------------------------');

for i = 1:n                                          % This is a loop to print the solutions of each successive iteration
   x1_new = (B(1,1) - A(1,2)*x2 - A(1,3)*x3) / A(1,1);    % Formula of Jacobi method
   x2_new = (B(2,1) - A(2,1)*x1 - A(2,3)*x3) / A(2,2);    % Formula of Jacobi method
   x3_new = (B(3,1) - A(3,1)*x1 - A(3,2)*x2) / A(3,3);    % Formula of Jacobi method
   x1 = x1_new;
   x2 = x2_new;
   x3 = x3_new;
   fprintf('\n%d        \t %.5f\t %.5f\t %.5f\n',i,x1, x2,x3 ); % Each variable is printed with a .5 precision
end
```

Output:

The user is asked to input the coefficients of matrix A and Vector B, and the number of iterations

```
Enter a11 = 3
Enter a12 = -0.1
Enter a13 = -0.2
Enter b1 = 7.85
Enter a21 = 0.1
Enter a22 = 7
Enter a23 = -0.3
Enter b2 = -19.3
Enter a31 = 0.3
Enter a32 = -0.2
Enter a33 = 10
Enter b3 = 71.4
```

```
Enter Number of ilterations = 3
```

The output is in tabular form:

iteration	x1	x2	x3
1	2.61667	-2.75714	7.14000
2	3.00076	-2.48852	7.00636
3	3.00081	-2.49974	7.00021

Solving nonlinear system using Bracket method

Bisection method

Example:

- **Example 1:**

  Find a solution to the equation

  $$x^2 + \ln(x) = 0$$

  using 4 iterations of the bisection method over the interval [0.5, 1].

Code:

```
%%
% Bisection Method
% Method is used to evaluate zeroes of input function bonded by limits

clc;
clear;

format long
% this line request user to enter function but user must follow some rules:
% 1- function must be in one variable which is x
% 2- variable x must be in closed bracket
% 3- operator are the same as used in matlab code
input_function_by_user = input('Enter function in x (note use parathess with x e.g: (x) ): ','s');
% input function is converted from string to function stored in f_x
f_x = str2func(['@(x)' input_function_by_user ]);

% user is asked to enter limits of function
A = input('Enter start at a = '); %left limit
B = input('Enter end at b = ');  % Right limit
```

```
% menu is used to offer user to choose termination method either error or
% iterations returning value corresponding to the choice in option.
option = menu ('Choose which terminate Method','Error','Ilterations');
switch(option)
    case 1
         error   = input('Enter Maximum Error : '); % maximum Error
         n = ceil(log2((B-A)/error));
    case 2
         n = input('Enter Number of ilterations = '); % Maximum itration
         error = (B - A)/(2^n);
end

List = zeros(n,7);
```

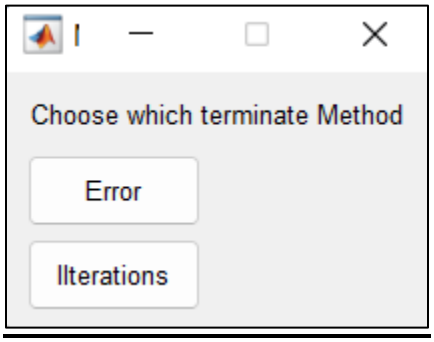
```
List = zeros(n,7);
%Method_____________________________________________________________
fa = f_x(A);
fb = f_x(B);
for i = 1:n
    r  = (A+B)/2;
    fr = f_x(r);
    List(i,:) = [i, A, fa, B, fb, r, fr]; %for illustration
    if (i~=1 && abs(r-r0) <= error)
        break;
    end
    if  fa*fr < 0,    B = r;  fb = fr;
    else,             A = r;  fa = fr;
    end
    r0 = r;
end

%Illustration_______________________________________________________
fprintf(['%3s',repmat('%11s',[1,6]),'\n'],...
    'itr','a','f(a)','b','f(b)','r','f(r)');
div = ['---',repmat('-----------',[1,6]), '\n'];        fprintf(div);
fprintf(['%3.0f', repmat('%11.4g',[1,6]), '\n'],List.'); fprintf(div);
fprintf('x = %10.4g\n',r);
fprintf('Error = %f\n',error);
```

Output:

Firstly, the user is asked to enter the function, starting point, and ending point:

```
Command Window
  Enter function in x (note use parathess with x e.g: (x) ): (x)^2 + log(x)
  Enter start at a = 0.5
  Enter end at b = 1
```

Secondly, a menu is shown to choose either iteration or error termination



Thirdly, the user types the number of iterations, and the output is viewed in tabular form

```
Enter Number of ilterations = 4
itr          a        f(a)          b        f(b)          r        f(r)
-------------------------------------------------------------------------
  1        0.5     -0.4431          1           1       0.75      0.2748
  2        0.5     -0.4431       0.75      0.2748      0.625    -0.07938
  3      0.625    -0.07938       0.75      0.2748     0.6875     0.09796
  4      0.625    -0.07938     0.6875     0.09796     0.6563    0.009451
-------------------------------------------------------------------------
x =     0.6563
Error = 0.031250
```

Solving nonlinear systems using Open method

Newton-Raphson method

Example:

Example 3:

Find a solution to the equation

$$x^2 + \ln(x) = 0$$

using Newton's method with a maximum error bound of $\varepsilon = 0.0001$, with an initial value $x_0 = 0.5$.

Code:

```
% Newton-Raphson method
% Method is used to evaluate zeroes of function input

clear;
clc;

% format function is used to change number of number after decimal
% displayed, it is chosen long to make it able to display 5 numbers.
format long;


% this line request user to enter function but user must follow some rules:
% 1- function must be in one variable which is x
% 2- variable x must be in closed bracket
% 3- operator are the same as used in matlab code
input_function_by_user = input('Enter function in x (note use parathess with x e.g: (x) ): ','s');
% input function is converted from string to function stored in f_x
f_x = str2func(['@(x)' input_function_by_user ]);

% syms fucntion is used to declare that x is varible in function that will
% be derivatived.
syms x;
% Evaluating differentiation of input function
df_x = eval(['@(x)' char(diff(f_x(x)))]);

% User is asked to enter initial x
x0 = input('Enter initial x = ');
```

```
% menu is used to offer user to choose termination method either error or
% iterations returning value corresponding to the choice in option.
option = menu ('Choose which terminate Method','Error','Iterations');
switch(option)
    case 1
        error   = input('Enter Maximum Error : '); % maximum Error
        i=0;
        fprintf('(0) X = %.5f\n',x0);
        x = x0 - (f_x(x0)/df_x(x0));

        while( abs(x-x0) > error)
            i=i+1;
            fprintf('(%d) X ',i);
            fprintf('= %.5f',x);
            fprintf('\t  E = %.5f\n',abs(x-x0));
            x0 = x;
            x = x0 - (f_x(x0)/df_x(x0));
        end
    case 2
        n = input('Enter Number of iterations = '); % Maximum itrations
        fprintf('(0) X = %.5f\n',x0);
         for i = 1:n

            x = x0 - (f_x(x0)/df_x(x0));
            if(i==n)
                break;
            end
```

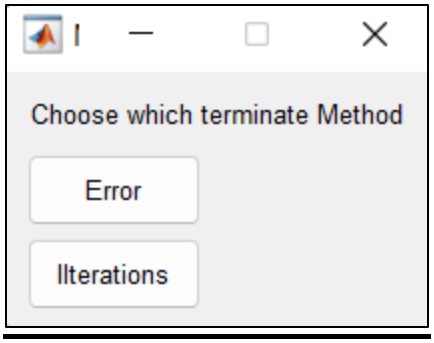
```
            i=i-1;

end
% Displaying Output
 fprintf('(%d) X = %.5f',i+1,x);
 fprintf('\t  E = %.5f\n',abs(x-x0));
 fprintf('\nX = %.5f\n',x);
```

Output:

At the beginning , the user is asked to enter the function and the initial guess.

```
Command Window
  Enter function in x (note use parathess with x e.g: (x) ): (x)^2 + log(x)
  Enter initial x = 0.5
```

Then, offered two methods of ways of termination



Finally, the user enters the error or iteration bound and the output is shown on the screen.

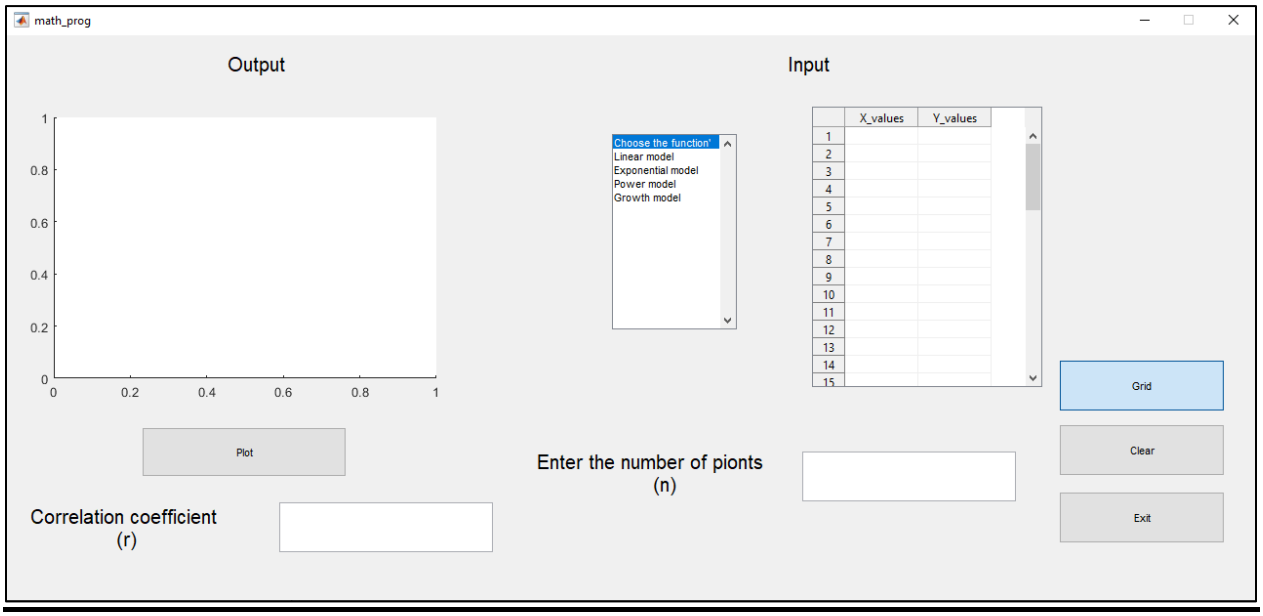
```
Enter Maximum Error : 0.0001
```

```
(0) X = 0.50000
(1) X = 0.64772   E = 0.14772
(2) X = 0.65292   E = 0.00520
(3) X = 0.65292   E = 0.00000

X = 0.65292
```

Linear Regression

Graphic user interface using matlab:

GUI:



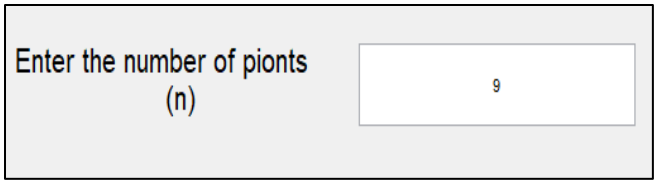
The GUI consists of :

1) **Table :** the table has two columns one column is for the x_values and the other column is for the y_values
2) **Text boxes :** there are two text boxes the first box of name “Enter the number of pionts” is used to take the number of points from the user . the second box of name ”correlation coefficient ” is used to display the correlation coefficient value
3) **List box:** the list box consists of the four models where you can choose the needed model
4) **Push buttons:** the pushbuttons is used to do aspecific function when you clicked on it with the mouse. There are four push buttons **“Grid”**: is used to make agrid on the figure
   the button **“clear”**: is used to clear the plot in the figure box to be able to plot another figure.
   The button **“ Exit”**: is used to exit the program
   The button **“Plot”**: is used to plot the points and acurve according to the chosen model from the list

How the program works:

First : the user inser the points manually in the table then insert the number of points in the text box called **“Enter the number of pionts”**

	X_values	Y_values
1	1	5
2	2	7.5
3	3	11
4	4	12
5	5	15
6	6	17.5
7	7	22
8	8	25
9	9	33
10		
11		
12		
13		
14		
15		

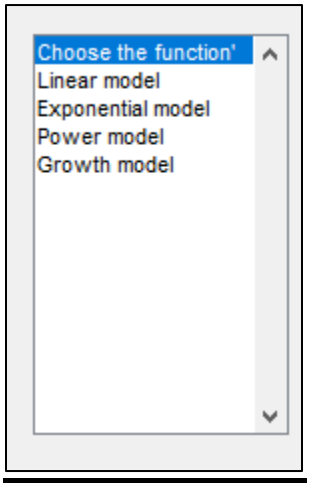


code:

Inside the callback function of the push button “plot” this code receives amatrix from the table that consists of two columns as string, so we change the type to “double” and split the columns to two columns one for x_values and the other for y_values .

```
% --- Executes on button press in plot.
function plot_Callback(hObject, eventdata, handles)
n=str2double(get(handles.num_1,'string'))   %number of pionts
m=str2double(get(handles.table_1,'data'))   %matrix of two columns
t=m(:,1)     %the first column
for i=1:1:n

    x(i)=t(i)
end
x                        %x is avector consists of x_values from 1 to n

k=m(:,2)    %the second column
for i=1:1:n

    y(i)=k(i)
end
y                %y is avector consists of x_values from 1 to n
```

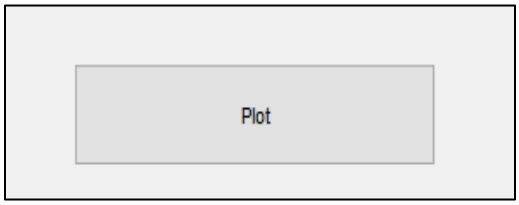
Second: the user choose the required model from the list box

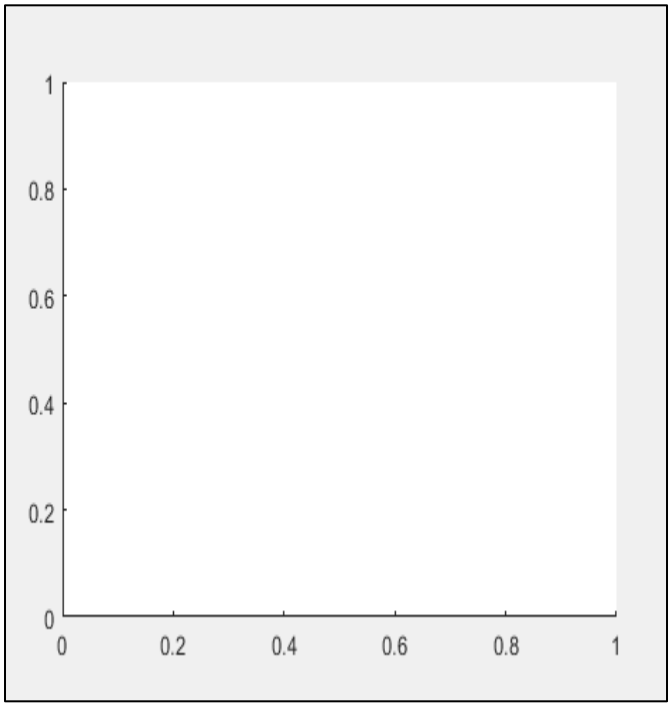


Code : after the user chooses the required model the value of this choice is sent to the variable " f " . so we used switch-case statement to choose the required model

```
f=get(handles.list,'value')

switch f

    case 2
   % The code of the linear model
    case 3
    %The code of the Exponential model
    case 4
    %The code of the Power model
    case 5
    %The code of the Growth rate model
end
```

Third : the user click on the plot push button and the plot is plotted in the figure box





Code: To plot the curve of the chosen model and the points on the same figure

```
set(handles.num_2,'string',r); % To display the correlation coeficient
axes(handles.axes1)
plot(x,y_new);  %To plot rhe choosen model
hold on;
plot(x,y,'.');   %To plot the pionts
title('Data Point VS Least Square Fit');
xlabel('x');
ylabel('y');
```

Linear Model:

Code of linear model:

If the user choose the “Linear model ” from the list box the function will run the

Code inside case 2

```
switch f
    case 2

  % sum are used for storing sum of x , x^2 , y and xy to be used in solution
 sum_X = 0;
sum_X2 = 0;
sum_Y = 0;
sum_XY = 0;

% sums are evaluated using sum function that add elements of vector
sum_X = sum(x);
sum_Y = sum(y);
sum_X2 = sum(x.^2);
sum_XY = sum(x.*y);

% soln matrix is formed then applying rref (reduced row echelon form) and
% taking it's 3rd column element which are ao and a1
soln = [n sum_X sum_Y ; sum_X sum_X2 sum_XY];
soln = rref(soln);
ao = soln(1,3);
a1 = soln(2,3);
% Evaluation a and b of the original equation
b = ao ;
a = a1 ;
```

```
% Evaluating y after linearization

 y_new= a*x + b;

avg_Y = mean(y);                % Average of y
St = sum((y-avg_Y).^2);         % sum of difference between avg_y and y
Sr = sum((y-ao-al*x).^2);       % sum of squared difference
r2=(St-Sr)/St ;                 % coefficient of determination
r=sqrt(r2);                     % coorelation coefficient
```

After the code is evaluated the function after linearization will be evaluated

Now we can plot the function and display the value of correlation coefficient (r) and the entered points by the user

```
set(handles.num_2,'string',r); % To display the correlation coeficient
axes(handles.axes1)
plot(x,y_new);  %To plot rhe choosen model
hold on;
plot(x,y,'.');   %To plot the pionts
title('Data Point VS Least Square Fit');
xlabel('x');
ylabel('y');
```

Example on linear model :

Inputs:

X	1	2	3	4	5	6	7	8	9
Y	5	7.5	11	12	15	17.5	22	25	33

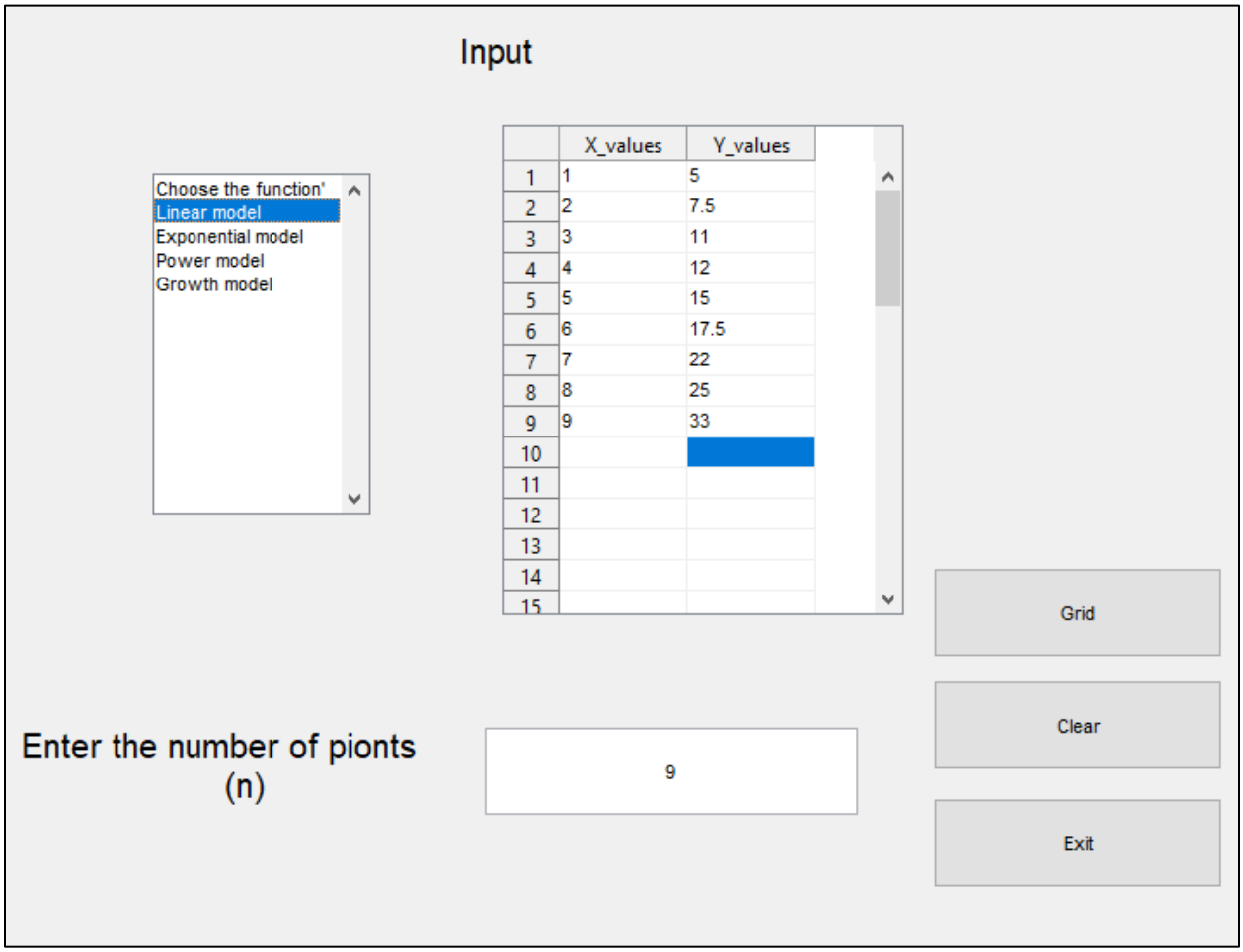
Steps:

1. enter the x and y values in the table in the GUI
2. enter the number of points in the text box

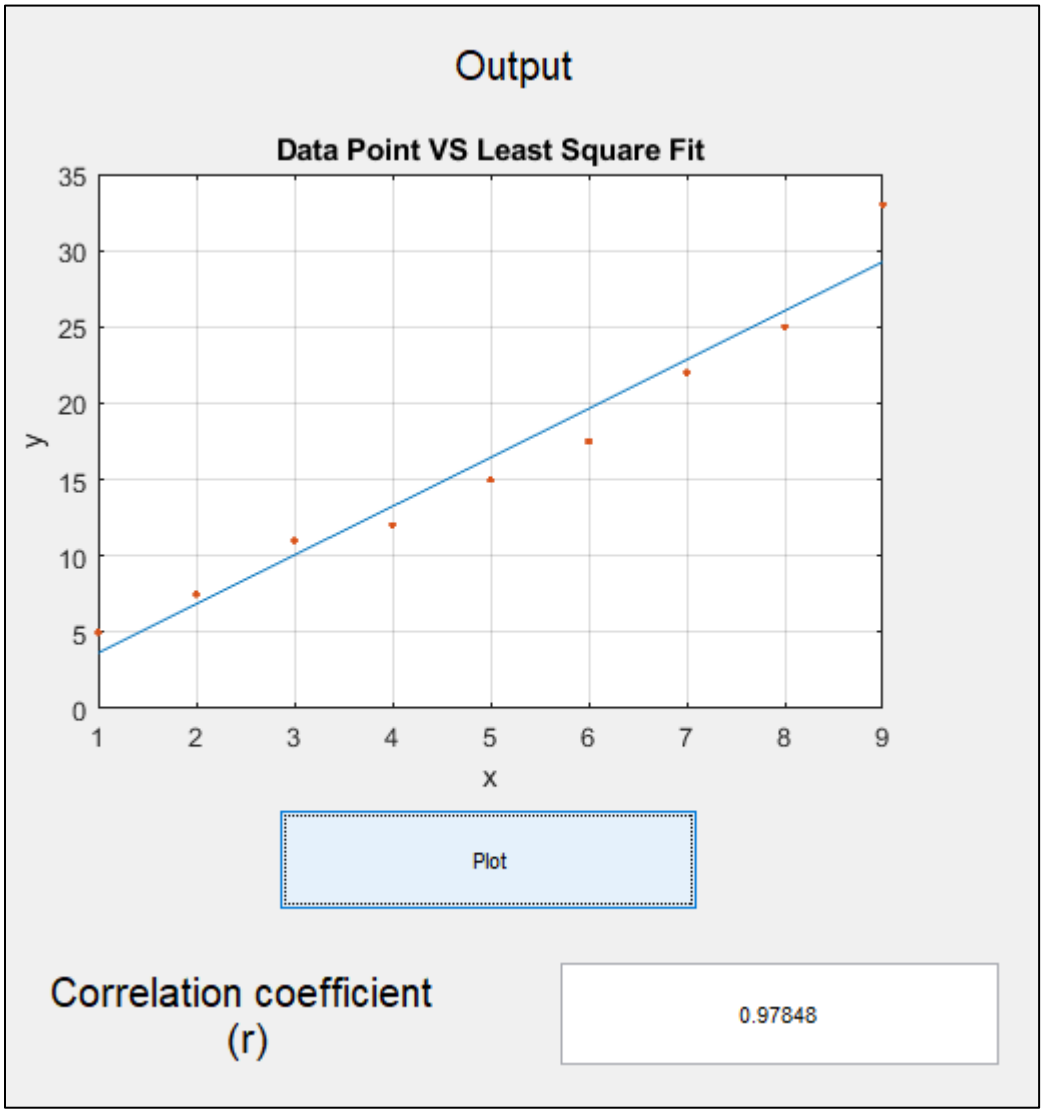
3. choose “Linear model ” from the List box

4. click the “Plot” bush button

5. if you want to make a grid on the figure click on “Grid” bush button



Result:



The correlation coefficient (r) =0.97848

(good fit) as r is very near to 1

Exponential model:

Code of Exponential model:

If the user choose the “Exponential model” from the list box the function will run the

Code inside **case 3**

```
    case 3

        %%Exponetial_model
    % sum are used for storing sum of x , x^2 , y and xy to be used in solution
sum_X = 0;
sum_X2 = 0;
sum_Y = 0;
sum_XY = 0;
    %since y = ax^b is linearized into ln(y) = ln(a) + bx then the
    % stored values in x and y arenot the input ones

 X =x;
 Y =log(y);

 % sums are evaluated using sum function that add elements of vector
sum_X = sum(X);
sum_Y = sum(Y);
sum_X2 = sum(X.^2);
sum_XY = sum(X.*Y);

soln = [n sum_X sum_Y ; sum_X sum_X2 sum_XY];
soln = rref(soln);
ao = soln(1,3);
al = soln(2,3);
```

```
% Evaluation a and b of the original equation
a = exp(ao) ;
b = al ;

% Evaluating y after linearization
y_new = a*exp(x.*b);

avg_Y = mean(Y);                % Average of y
St = sum((Y-avg_Y).^2);         % sum of difference between avg_y and y
Sr = sum((Y-ao-al*X).^2);       % sum of squared difference
r2=(St-Sr)/St ;                 % coefficient of determination
r=sqrt(r2);                     % coorelation coefficient
```

After the code is evaluated the function after linearization will be evaluated

Now we can plot the function and display the value of correlation coefficient (r) and the entered points by the user

```
set(handles.num_2,'string',r); % To display the correlation coeficient
axes(handles.axes1)
plot(x,y_new);  %To plot rhe choosen model
hold on;
plot(x,y,'.');   %To plot the pionts
title('Data Point VS Least Square Fit');
xlabel('x');
ylabel('y');
```

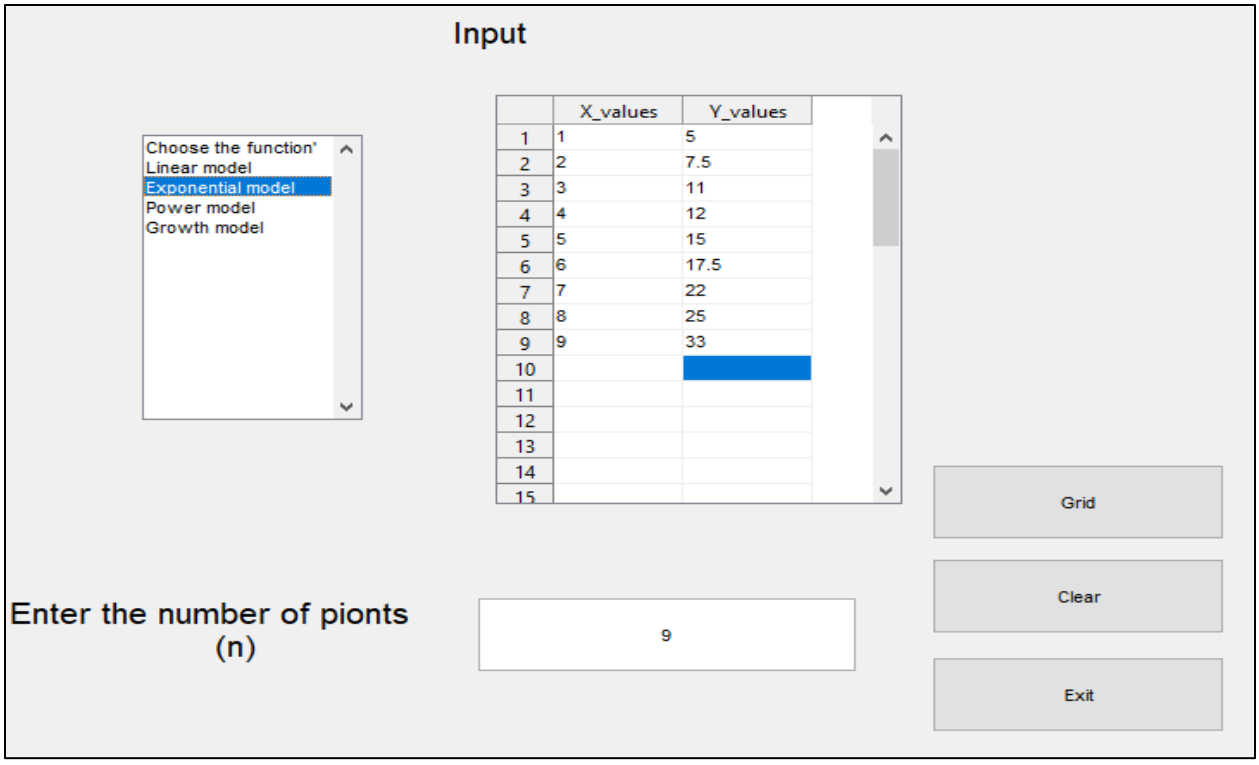
Example on Exponential model :

Inputs:

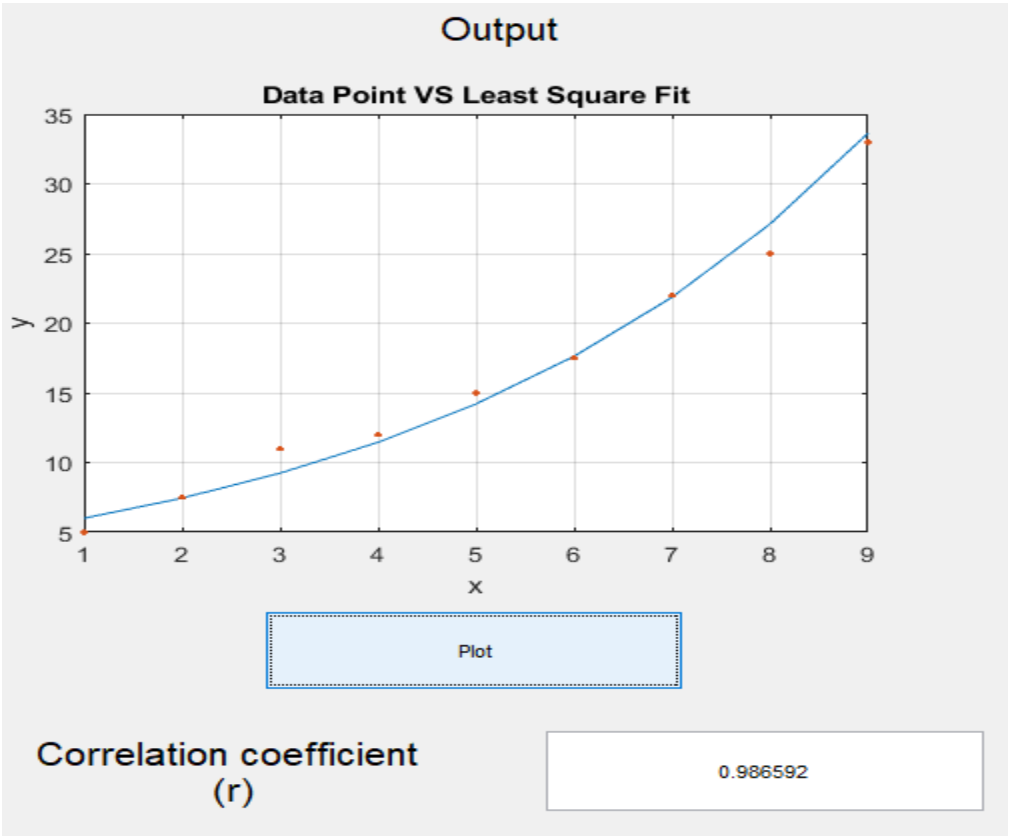
X	1	2	3	4	5	6	7	8	9
Y	5	7.5	11	12	15	17.5	22	25	33

Steps:

1. enter the x and y values in the table in the GUI
2. enter the number of points in the text box
3. choose “Exponential model ” from the List box
4. click the “Plot” bush button
5. if you want to make a grid on the figure click on “Grid” bush button


Input
Choose the function'
Linear model
Exponential model
Power model
Growth model
X_values
Y_values
1 1 5
2 2 7.5
3 3 11
4 4 12
5 5 15
6 6 17.5
7 7 22
8 8 25
9 9 33
10
11
12
13
14
15
Grid
Clear
Exit
Enter the number of pionts (n)
9

Result:


Output
Data Point VS Least Square Fit
35
30
25
20
15
10
5
y
1
2
3
4
5
6
7
8
9
x
Plot
Correlation coefficient (r)
0.986592

The correlation coefficient (r) =0.986592

(good fit) as r is very near to 1

Power model:

Code of Power model:

If the user choose the “Exponential model” from the list box the function will run the

Code inside case 4

```
    case 4
        %%power_model

         % sum are used for storing sum of x , x^2 , y and xy to be used in solution
sum_X = 0;
sum_X2 = 0;
sum_Y = 0;
sum_XY = 0;
    %since y = ax^b is linearized into log(y) = log(a) + b log(x) then the
    % stored values in x and y arenot the input ones but their log
 X =log10(x);
 Y =log10(y);

 % sums are evaluated using sum function that add elements of vector
sum_X = sum(X);
sum_Y = sum(Y);
sum_X2 = sum(X.^2);
sum_XY = sum(X.*Y);

soln = [n sum_X sum_Y ; sum_X sum_X2 sum_XY];
soln = rref(soln);
ao = soln(1,3);
al = soln(2,3);
```

```
% Evaluation a and b of the original equation
a = 10^ao ;
b =  al ;

% Evaluating y after linearization
y_new = a*x.^b;

avg_Y = mean(Y);                % Average of y
St = sum((Y-avg_Y).^2);         % sum of difference between avg_y and y
Sr = sum((Y-ao-al*X).^2);       % sum of squared difference
r2=(St-Sr)/St ;                 % coefficient of determination
r=sqrt(r2);                     % coorelation coefficient
```

After the code is evaluated the function after linearization will be evaluated

Now we can plot the function and display the value of correlation coefficient (r) and the entered points by the user

```
set(handles.num_2,'string',r); % To display the correlation coeficient
axes(handles.axes1)
plot(x,y_new);  %To plot rhe choosen model
hold on;
plot(x,y,'.');   %To plot the pionts
title('Data Point VS Least Square Fit');
xlabel('x');
ylabel('y');
```

Example on Power model :

Inputs:

X	1	2	3	4	5	6	7	8	9
Y	5	7.5	11	12	15	17.5	22	25	33

Steps:

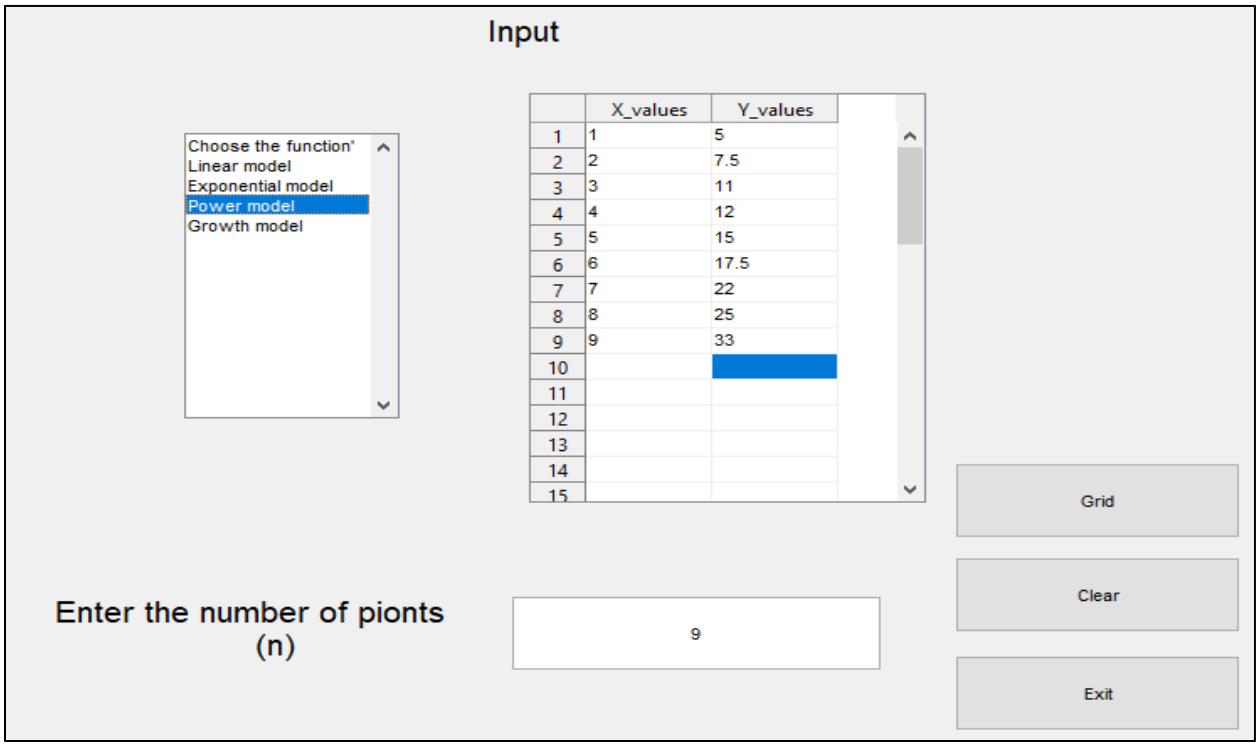
1. enter the x and y values in the table in the GUI

2. enter the number of points in the text box

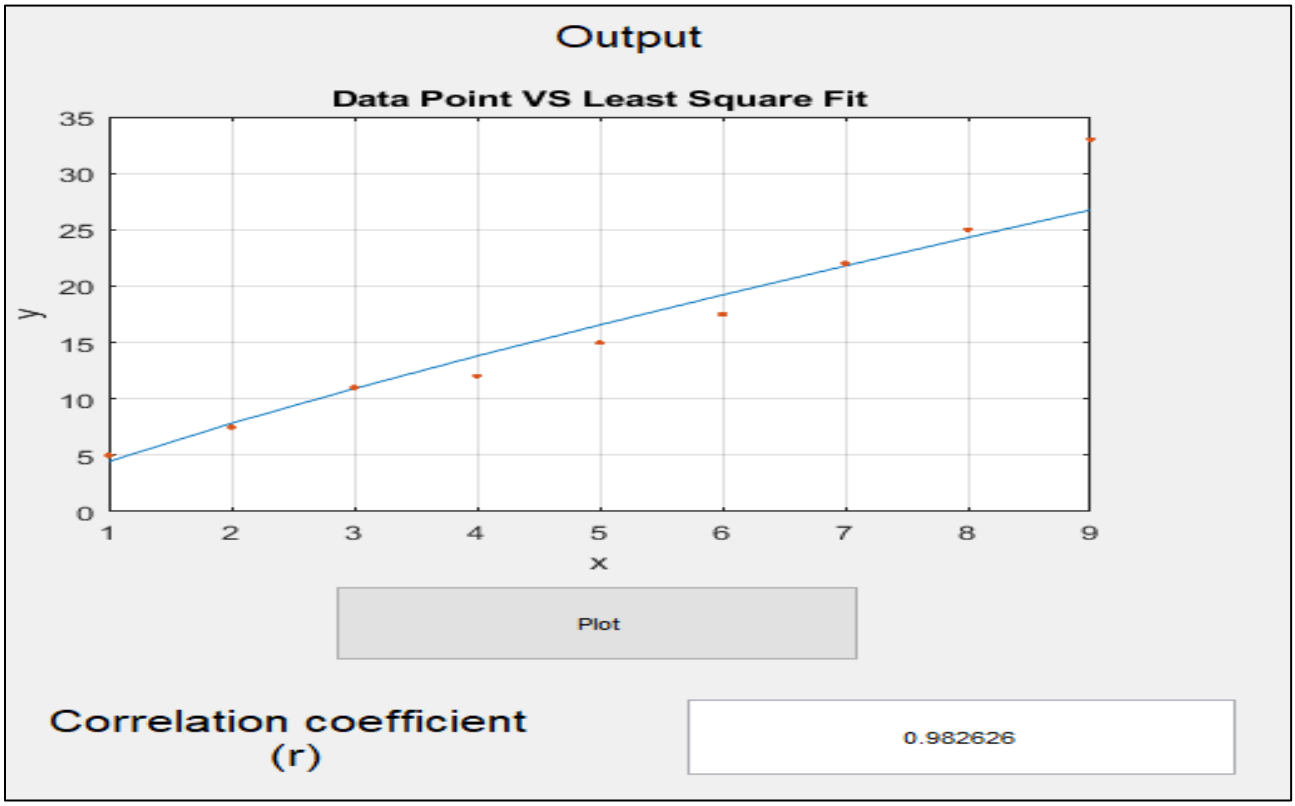
3. choose “Power model” from the List box

4. click the “Plot” bush button

5. if you want to make a grid on the figure click on “Grid” bush button



Result:



The correlation coefficient (r) =0.982626

(good fit) as r is very near to 1

Growth-rate model:

Code of Power model:

If the user choose the “Exponential model ” from the list box the function will run the

Code inside case 5

```
    case 5
            %%Growthrate_model
         % sum are used for storing sum of x , x^2 , y and xy to be used in solution
sum_X = 0;
sum_X2 = 0;
sum_Y = 0;
sum_XY = 0;
    % since y = ax^b is linearized into 1/y = 1/a + b/a *(1/x) then the
    % stored values in x and y arenot the input ones but their log
 X =1./x;
 Y =1./y;

 % sums are evaluated using sum function that add elements of vector
sum_X = sum(X);
sum_Y = sum(Y);
sum_X2 = sum(X.^2);
sum_XY = sum(X.*Y);

soln = [n sum_X sum_Y ; sum_X sum_X2 sum_XY];
soln = rref(soln);
ao = soln(1,3);
al = soln(2,3);
```

```
% Evaluating y after linearization
y_new = a * x ./( b + x);

avg_Y = mean(Y);                % Average of y
St = sum((Y-avg_Y).^2);         % sum of difference between avg_y and y
Sr = sum((Y-ao-al*X).^2);       % sum of squared difference
r2=(St-Sr)/St ;                 % coefficient of determination
r=sqrt(r2);                     % coorelation coefficient

end
```

After the code is evaluated the function after linearization will be evaluated

Now we can plot the function and display the value of correlation coefficient (r) and the entered points by the user

```
set(handles.num_2,'string',r); % To display the correlation coeficient
axes(handles.axes1)
plot(x,y_new);  %To plot rhe choosen model
hold on;
plot(x,y,'.');   %To plot the pionts
title('Data Point VS Least Square Fit');
xlabel('x');
ylabel('y');
```

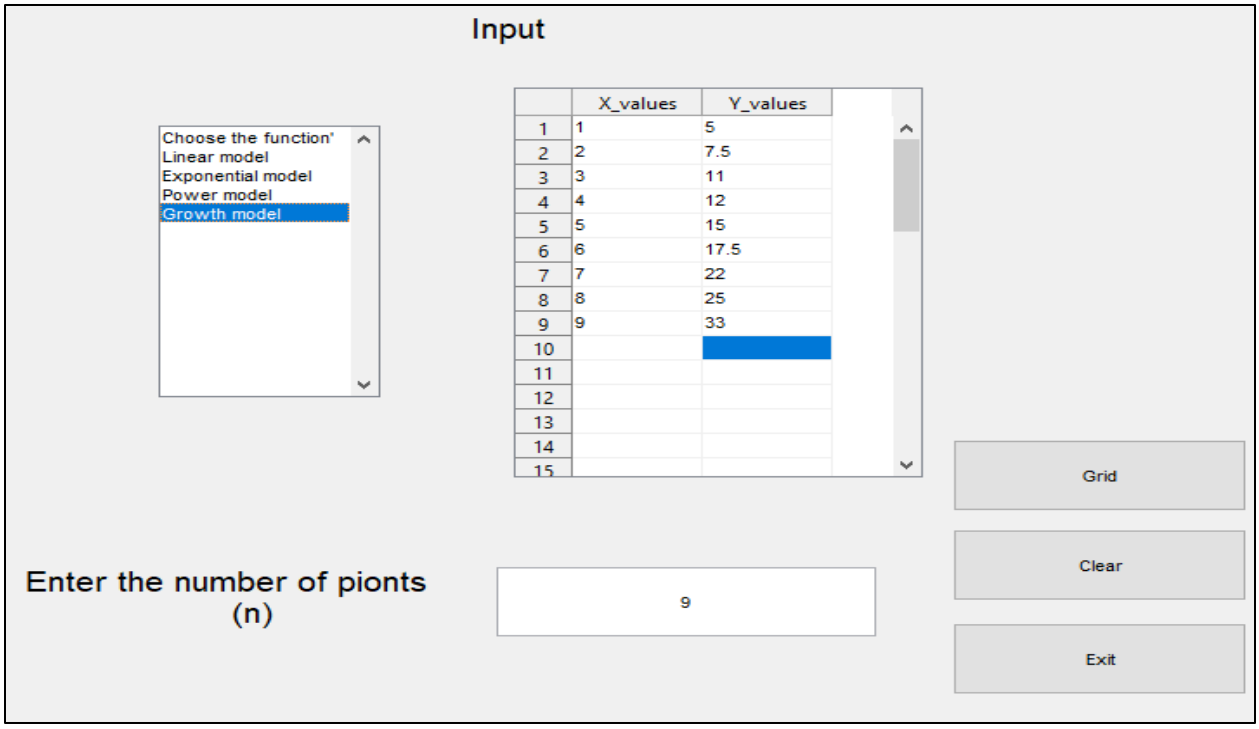
Example on Growth-rate model :

Inputs:

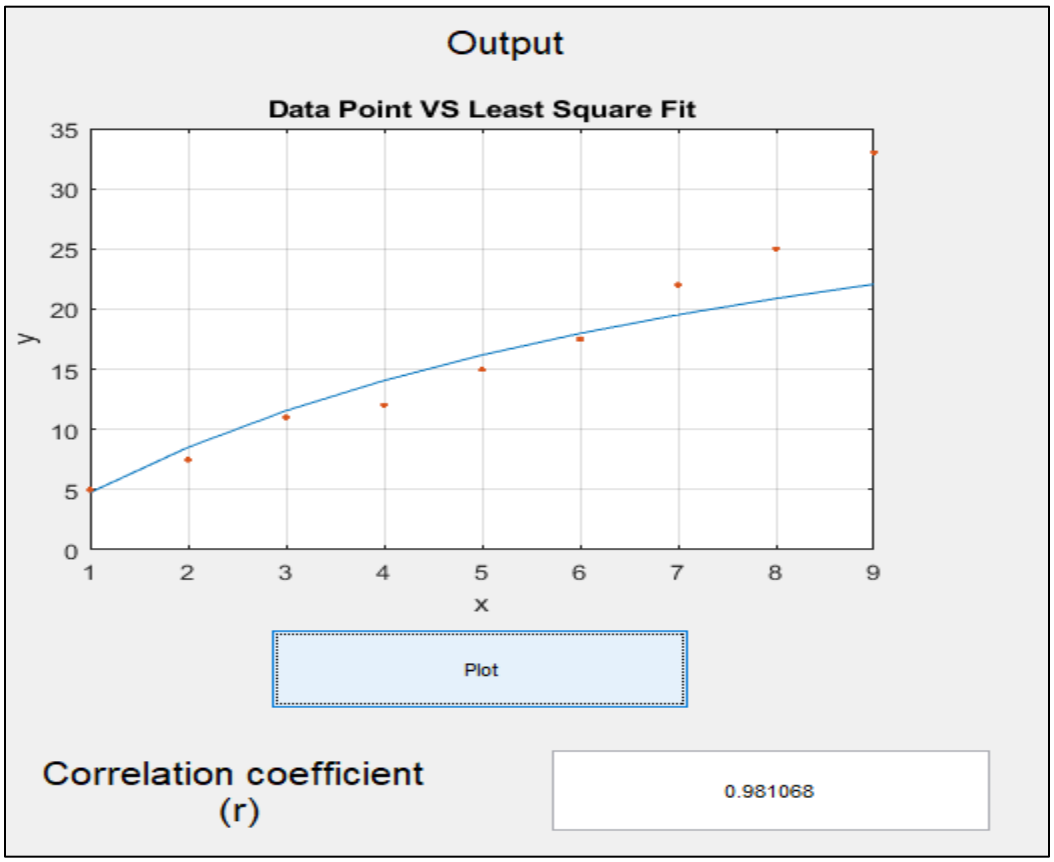
X	1	2	3	4	5	6	7	8	9
Y	5	7.5	11	12	15	17.5	22	25	33

Steps:

1. enter the x and y values in the table in the GUI
2. enter the number of points in the text box
3. choose “Growth model ” from the List box
4. click the “Plot” bush button
5. if you want to make a grid on the figure click on “Grid” bush button


Input
Choose the function'
Linear model
Exponential model
Power model
Growth model
X_values
Y_values
1 1 5
2 2 7.5
3 3 11
4 4 12
5 5 15
6 6 17.5
7 7 22
8 8 25
9 9 33
10
11
12
13
14
15
Grid
Clear
Exit
Enter the number of pionts
(n)
9

Result


Output
Data Point VS Least Square Fit
35
30
25
20
15
10
5
0
y
1
2
3
4
5
6
7
8
9
x
Plot
Correlation coefficient
(r)
0.981068

The
correlation coefficient (r) =0.981068

(good fit) as r is very near to 1

The perfect fit model:

the perfect fit model is the model that perfectly fit the entered points by the user
,

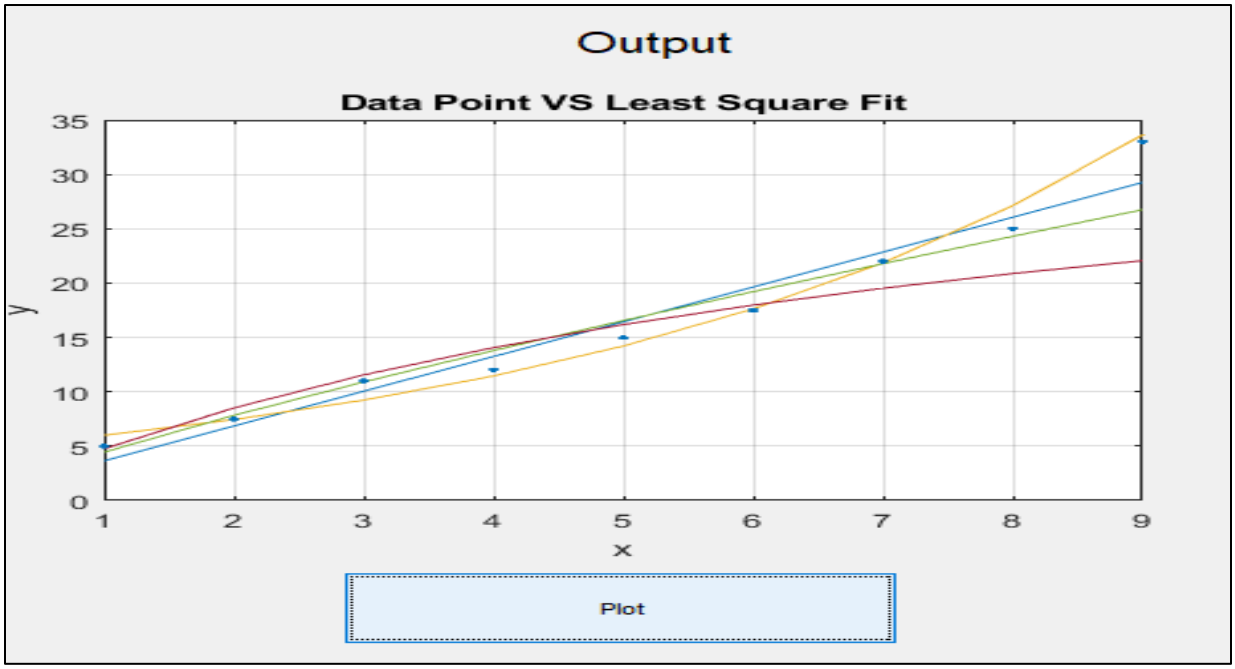
We can know the perfect fit model from the value of the

Correlation coefficient (r) , perfect fit model is the model that has the greatest value of (r)

So from the previous examples

We noticed that the **exponential model** is the perfect fit model as it has the largest Correlation coefficient (r) = **0.986592**

We can also know the perfect fit model from the plot by viewing the four models in one plot and see which model fits the entered points perfectly



we noticed that the **exponential curve (YELOW CURVE)** is the best model that fits the entered pionts

Numerical Integration:

Trapezoidal method

Example:

Evaluate the following integral using Trapezoidal with step size of 0.2

$$\int_0^1 x\, e^{-x}\, dx$$

Inputs:

h = 0.2

a = 0

b = 1

f_x = (x)* exp(-x)

Code:

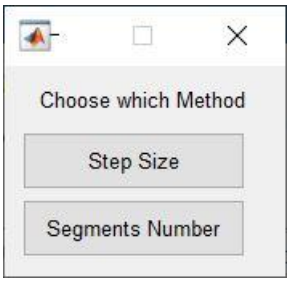
the user is asked to enter a and b

```
% Trapezoidal method
% Method is used to calculate approxmiately value of definite integration

clear;
clc;

% User is asked to enter:
% initial point of integeration a.
% final point of integration b.
A = input('Enter start at a = ');
B = input('Enter end at b = ');
```

```
Enter start at a = 0
Enter end at b = 1
```

User is offered to choose either to enter step size of number of segments

```
% User is asked to either enter step size or segments number.
% then the other parameter is calculated
option = menu ('Choose which Method','Step Size','Segments Number');
switch(option)
    case 1
        h = input('Enter Step size h = ');
        n=ceil((B-A)/h) ;
    case 2
        n = input('Enter number of segments n = ');
        h = (B-A)/n ;
end
```




```
Enter Step size h = 0.2
```

User is asked to enter equation that will be integrated but user must follow certain rule when entering equation:

1- function must be in one variable which is x
2- variable x must be in closed bracket
3- operators are the same as used in MATLAB code

Program then evaluates f at points from a to b with step h and store the result in vector f_n. then the result of integration is displayed

```
% this line request user to enter function but user must follow some rules:
% 1- function must be in one variable which is x
% 2- variable x must be in closed bracket
% 3- operator are the same as used in matlab code
input_function_by_user = input('Enter function in x (note use parathess with x e.g: (x) ): ','s');
% input function is converted from string to function stored in f_x
f_x = str2func(['@(x)' input_function_by_user ]);

% Evaluating f at b and f at a
f_b = f_x(B);
f_a = f_x(A);
```

Then the rule is applied, and output is displayed.

```
% f_n is vector used to store f_x at each step to be summed later
f_n = [];
for i = 1 : n-1
    f_n = [f_n f_x(A + i*h)];
end

% Evaluating sum
sum_points = sum(f_n);

% Applying Trapeziodal Method rule
integration = (h/2)*(f_a + f_b + 2*sum_points);

% displaying the result of integration using Trapeziodal rule
fprintf(['\napproxmiate Integration = ' num2str(integration) '\n']);
```

Result:

```
approxmiate Integration = 0.26091
```

Outputs:

Integration = 0.26091

Simpson's 1/3 rule

Example:

Evaluate the following integral using Simpson's 1/3 rule with step size of 0.25

$$\int_0^2 x\,cos(e^x)\,dx$$

Inputs:

h = 0.25

a = 0

b = 2

f_x = (x)*cos(exp(x))

Code:

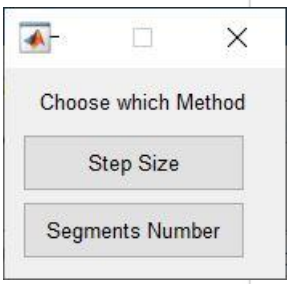
the user is asked to a and b and menu pop up asking user whether termination method was by step size or max error then code check if the resulted n of the previous inputs is even or odd in case if n is odd, the code display message

'Number of segments isn't even number, enter different h and try again'. In case if n is even, code continues.

```
% Simpson's 1/3 rule
% Method is used to calculate approxmiately value of definite integration

clear;
clc;
```

```
Enter start at a = 0
Enter end at b = 2
```

```
% n is number of segment and it is initailized as 1 then it is checked if
% it is even or not as this method require odd number of points thus even
% number of segments
n=1;

% while loop check if n is even or not.
% in case of odd, repeat.
% in case of even, continue the rest of the code.
while(mod(n,2)==1)

% User is asked to enter:
% initial point of integeration a.
% final point of integration b.
A = input('Enter start at a = ');
B = input('Enter end at b = ');

% User is asked to either enter step size or segments number.
% then the other parameter is calculated
option = menu ('Choose which Method','Step Size','Segments Number');
switch(option)
    case 1
        h = input('Enter Step size h = ');
        n=ceil((B-A)/h) ;
    case 2
        n = input('Enter number of segments n = ');
        h = (B-A)/n ;
end
```



Then user is asked to enter equation that will be integrated but user must follow certain rule when entering equation:

1- function must be in one variable which is x
2- variable x must be in closed bracket
3- operators are the same as used in MATLAB code

```
% this line request user to enter function but user must follow some rules:
% 1- function must be in one variable which is x
% 2- variable x must be in closed bracket
% 3- operator are the same as used in matlab code
input_function_by_user = input('Enter function in x (note use parathess with x e.g: (x) ): ','s');
% input function is converted from string to function stored in f_x
f_x = str2func(['@(x)' input_function_by_user ]);

% Evaluating f at b and f at a
f_b = f_x(b);
f_a = f_x(a);

% this vector is used to store f_n elements for evaluating odd and even sum
f_n = [];

% Loop used for evaluating f at each step
for i = 1 : n-1
    f_n = [f_n f_x(a + i*h)];
end

% Evaluating even and odd sums
sum_odd = sum(f_n(1:2:n-1));
sum_even =  sum(f_n(2:2:n-1));
```

```
Enter function in x (note use parathess with x e.g: (x) ): x*cos(exp(x))
```

Program then evaluates f at points from a to b with step h and store the result in vector f_n. the sum of odd segments is evaluated and stored in sum_odd while sum of even segments is also evaluated and stored in sum_even

the rule is then applied, and the result of integration is displayed

```
% Applying Simpsons 1/3 rule
integration = (h/3)*(f_a + f_b + 4*sum_odd + 2*sum_even);

% displaying the result of integration using Simpson's 1/3 rule
fprintf(['\nIntegration = ' num2str(integration) '\n']);
```

Result:

```
approxmiate Integration = -0.13503
```

Outputs:

Integration = -0.13503

Solving Ordinary Differential Equation Numerically

Euler's method

Example:

Use Euler's method to find the value of y over the interval t = 0 to t = 1 with a step size of 0.25 given that y(0) = 1

$$\frac{dy}{dt} = yt^3 - 1.5y$$

Inputs:

h = 0.25

starting point = 0

ending point = 1

y at starting point = 1

Differential Equation = (y)*(x)^3 – 1.5 * (y)

Code:

The user is asked to then user is asked to enter Differential Equation but must follow certain rules:

1- function must be in one or two variables which are x,y
2- variable x, variable y must be in closed bracket
3- operators are the same as used in MATLAB code

then the user is asked to enter step size h, starting point x_start, ending point x_end and y at the is starting point y_o.

```
%%
% Euler's method
% Method used to estimate solution of ODE approximately

clear;
clc;

% this line request user to enter function but user must follow some rules:
% 1- function must be in one or two variable which are x and y
% 2- variables x and y must be in closed bracket
% 3- operator are the same as used in matlab code
input_function_by_user = input('Enter function in x (note use parathess with x e.g: (x) ): ','s');
% input function is converted from string to function stored in f_x
f_x = str2func(['@(x,y)' input_function_by_user ]);

% User is asked to enter:
% step size h.
% Starting point of the curve x_start.
% ending point of the curve x_end.
h = input('Enter step size h = ');
x_start = input('Enter initial x = ');
x_end = input('Enter final x = ');

% n is evaluated as ceil of the (x_end - x_start)/h as n must be integer
n = ceil((x_end - x_start)/h );

% user is asked to enter initial y at x = 0
y_o = input(['Enter at x = ' num2str(x_start) ' , y = ']);
```

```
Enter function in x (note use parathess with x e.g: (x) ): (y)*(x)^3 - 1.5*(y)
Enter step size h = 0.25
Enter initial x = 0
Enter final x = 1
Enter at x = 0 , y = 1
```

the Program begins to apply method by evaluating x from starting point to end and storing the y at starting point in vector y.

```
% y_old is initialized as y_o so that it will be used in first loop
y_old = y_o;

% y vector will store all values of y as result of Euler's method
y = [y_o];

% x is vector used as independent variables in Euler's Method
x = x_start:h:x_end;
for i = 1 : n
y_new = y_old + h* f_x(x(i),y_old);
y_old = y_new;
y = [y y_new];
end

% Displaying final result
disp(['y = ' num2str(y)]);

% Ploting the final result
figure;
plot(x,y);
```

Result:

Outputs:

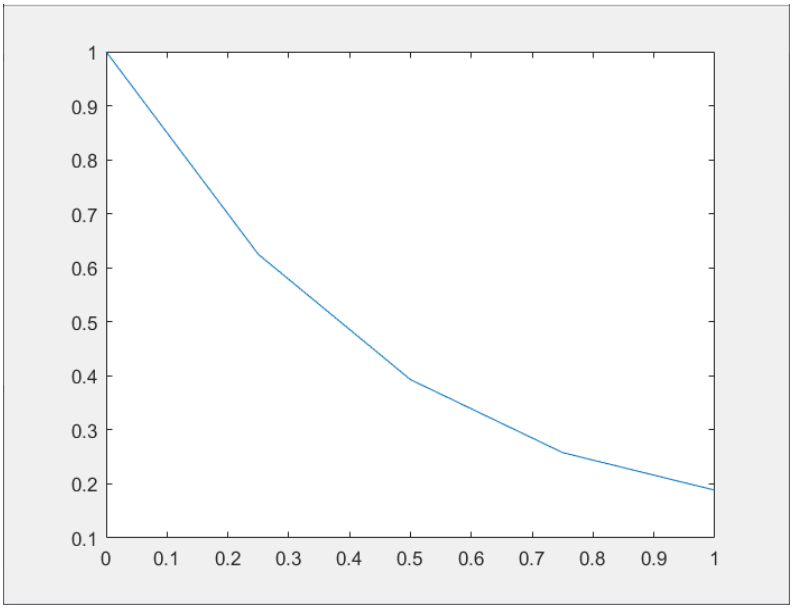
y(0) = 1

y(0.25) = 0.6250

y(0.5) = 0.3931

y(0.75) = 0.2579

y(1) = 0.1884




```
y = 1        0.625      0.39307      0.25795      0.18842
```

Heun's method (Improved Euler's method)

Example:

Use Henu's method to find the value of y over the interval t = 0 to t = 1 with a step size of 0.25 given that y(0) = 1

$$\frac{dy}{dt} = yt^3 - 1.5y$$

Inputs:

h = 0.25

starting point = 0

ending point = 1

y at starting point = 1

Differential Equation = (y)*(x)^3 – 1.5 * (y)

Code:

The user is asked to then user is asked to enter Differential Equation but must follow certain rules:

4- function must be in one or two variables which are x,y
5- variable x, variable y must be in closed bracket
6- operators are the same as used in MATLAB code

then the user is asked to enter step size h, starting point x_start, ending point x_end and y at the is starting point y_o.

the Program begins to apply method by evaluating x from starting point to end and storing the y at starting point in vector y.

the method is done in two steps.

```
% Heun's method

clear;
clc;
% this line request user to enter function but user must follow some rules:
% 1- function must be in one or two variable which are x and y
% 2- variables x and y must be in closed bracket
% 3- operator are the same as used in matlab code
input_function_by_user = input('Enter function in x (note use parathess with x e.g: (x) ): ','s');
% input function is converted from string to function stored in f_x
f_x = str2func(['@(x,y)' input_function_by_user ]);

% User is asked to enter:
% step size h.
% Starting point of the curve x_start.
% ending point of the curve x_end.
h = input('Enter step size h = ');
x_start = input('Enter initial x = ');
x_end = input('Enter final x = ');

% n is evaluated as ceil of the (x_end - x_start)/h as n must be integer
n = ceil((x_end - x_start)/h );


% user is asked to enter initial y at x = 0
y_o = input(['Enter at x = ' num2str(x_start) ', y = ']);
```

```
Enter function in x (note use parathess with x e.g: (x) ): (y)*(x)^3 - 1.5*(y)
Enter step size h = 0.25
Enter initial x = 0
Enter final x = 1
Enter at x = 0, y = 1
```

first step is predictor where y_star is evaluated using euler method

second step is corrector where y_new is evaluated using euler method but with average of y_star and y_old value in differential equation

y_old become equal to y_new in preparation for next step

y_new is then stored in y by concatenation with original y.

```
% y_old is initialized as y_o so that it will be used in first loop
y_old = y_o;

% y vector will store all values of y as result of Heun's method
y = [y_o];

% x is vector used as independent variables in Heun's Method
x = x_start : h : x_end;
for i = 1 : n
y_star = y_old + h* f_x(x(i),y_old); % Predictor
y_new = y_old + (h/2)*( f_x(x(i+1),y_star) + f_x(x(i) ,y_old)); % Corrector

% y_old is then become y_new in preperation for next step
y_old = y_new;

% Storing value of y_new in y for final result
y = [y y_new];
end
```

the result is then displayed, and figure is plotted between x and y using plot function.

```
% Displaying final result
disp(['y = ' num2str(y)]);

% Ploting the final result
figure;
plot(x,y);
xlabel('x');
ylabel('y');
```

Result:

Outputs:

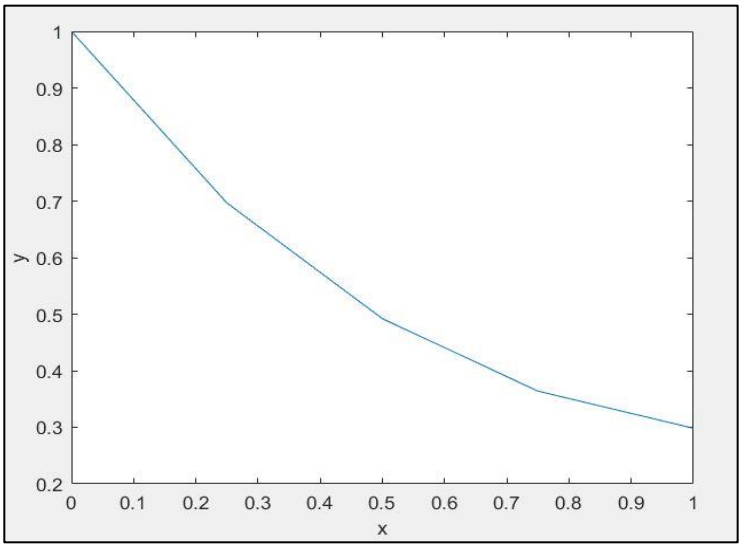
y(0) = 1

y(0.25) = 0.69653

y(0.5) = 0.492

y(0.75) = 0.3639

y(1) = 0.29827




```
y = 1      0.69653      0.492      0.36393      0.29827
```