Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

LAB REPORT

on

Sessional based on ETE 3115

Course Code: ETE 3116

SUBMITTED BY	**SUBMITTED TO**
Name: **Md. Darul Atfal Palash** Roll: **1804005** Year: 3rd year (Odd Semester) Session: 2018-2019 Date of Submission:17.08.2022	**Dr. Shah Ariful Hoque Chowdhury** **Associate Professor** Dept. of ETE, RUET

Index

Experiment No: 01

Experiment Name: Determining the Roots of Equation-1 (Bracketing Methods).

Objectives:
The purpose of this experiment is:

1. To know about bisection and false position methods.
2. To develop basic idea of implementing numerical methods using Python.
3. To know some Python functions required for the task.
4. To write separate programs for bisection and false position methods.
5. To compare their performance in terms of accuracy for a given no. of iterations.

Required Software: Python

Experimental Analysis:

Problem Statement: Use the bisection & false position method to determine the drag coefficient c needed for a parachutist of mass $m = 568.1\ kg$ to have a velocity of 40 m/s after free falling for time $t = 510\ s$. *Note:* The acceleration due to gravity is 9.81 ms^{-2}.

Solution: Required equation is given below:

$$f(c) = \frac{9.81(68.1)}{c}\left(1 - e^{-\left(\frac{c}{68.1}\right)10}\right) - 40$$

or, $$f(c) = \frac{668.061}{c}\left(1 - e^{-0.146843c}\right) - 40 \qquad \text{..................(1.1)}$$

(i) For Bisection Method:

Python Code:

```
#Import libraries as necessary
import math
import numpy as np
from xlwt import Workbook


#Take necessary input
#For bisection, two input is required to bracket the root
xl=float(input ('Enter 1st guess: '))   #1st input
xu=float(input ('Enter 2nd guess: '))   #2nd input
```

```
#computing function values corresponding to initial values
fxl=(668.061/xl)*(1-math.exp(-0.146843*xl))-40
fxu=(668.061/xu)*(1-math.exp(-0.146843*xu))-40

#checking initial input values
if fxl*fxu>0:
        print('Wrong initial input')
#if the initial input is correct
elif fxl*fxu<0:
    #taking input
    err=float(input('Enter desired percentage relative error: '))
    ite=int(input('Enter number of iterations: '))
    #initialization
    x_l=np.zeros([ite])
    x_u=np.zeros([ite])
    x_c=np.zeros([ite])

    f_xl=np.zeros([ite])
    f_xu=np.zeros([ite])
    f_xc=np.zeros([ite])

    rel_err=np.zeros([ite])
    itern=np.zeros([ite])
    #storing initial computed values into array
    x_l[0]=xl
    x_u[0]=xu

    f_xl[0]=fxl
```

```
f_xu[0]=fxu
#begin iteration
for i in range(ite):
    #storing the values of iteration
    itern[i]=i+1
    #Bisection Formula
    x_c[i]=(x_l[i]+x_u[i])/2

    f_xl[i]=(668.061/x_l[i])*(1-math.exp(-0.146843*x_l[i]))-40
    f_xu[i]=( 668.061/x_u[i])*(1-math.exp(-0.146843*x_u[i]))-40
    f_xc[i]=( 668.061/x_c[i])*(1-math.exp(-0.146843*x_c[i]))-40
    #calculating error
    if i>0:
        rel_err[i]=((x_c[i]-x_c[i-1])/x_c[i])*100
    #terminate if error criteria meets
    if all ([i>0, abs(rel_err[i])<err]):
        break
    elif f_xc[i]==0:
        break

    if i==ite-1:
        break
    #replacement of the new estimate
    if all ([f_xc[i]>0, f_xl[i]>0]):
        x_l[i+1]=x_c[i]
        x_u[i+1]=x_u[i]
    elif all ([f_xc[i]>0, f_xu[i]>0]):
        x_u[i+1]=x_c[i]
        x_l[i+1]=x_l[i]
    elif all ([f_xc[i]<0, f_xl[i]<0]):
        x_l[i+1]=x_c[i]
        x_u[i+1]=x_u[i]
    elif all ([f_xc[i]<0, f_xu[i]<0]):
        x_u[i+1]=x_c[i]
        x_l[i+1]=x_l[i]

#Writing the results on an excel sheet
#Workbook is created
wb = Workbook()

# add_sheet is used to create sheet.
sheet1 = wb.add_sheet('Sheet 1')
num_of_iter=i

#writing on excel
sheet1.write(0,0,'Number of iteration')
sheet1.write(0,1,'x_l')
sheet1.write(0,2,'x_u')
sheet1.write(0,3,'f(x_l)')
sheet1.write(0,4,'f(x_u)')
sheet1.write(0,5,'x_c')
sheet1.write(0,6,'f(x_c)')
sheet1.write(0,7,'Relative error')

#writing values on excel
for n in range(num_of_iter+1):
    sheet1.write(n+1,0,itern[n])
    sheet1.write(n+1,1,x_l[n])
    sheet1.write(n+1,2,x_u[n])
    sheet1.write(n+1,3,f_xl[n])
    sheet1.write(n+1,4,f_xu[n])
    sheet1.write(n+1,5,x_c[n])
    sheet1.write(n+1,6,f_xc[n])
    sheet1.write(n+1,7,rel_err[n])

sheet1.write(n+4,2,'The')
sheet1.write(n+4,3,'root')
sheet1.write(n+4,4,'is')
sheet1.write(n+4,5,x_c[i])

#save the excel file
wb.save('bisection example.xls')
```

Result:

```
Enter 1st guess: 12
Enter 2nd guess: 16
Enter desired percentage relative error: 0.05
Enter number of iterations: 10
```

Data from the Excel sheet, named "bisection example.xls"

Number of iteration	x_l	x_u	f(x_l)	f(x_u)	x_c	f(x_c)	Relative error
1	12	16	6.113957	-2.23025	14	1.611127	0
2	14	16	1.611127	-2.23025	15	-0.38445	6.666667
3	14	15	1.611127	-0.38445	14.5	0.593708	-3.44828
4	14.5	15	0.593708	-0.38445	14.75	0.099839	1.694915
5	14.75	15	0.099839	-0.38445	14.875	-0.14349	0.840336

6	14.75	14.875	0.099839	-0.14349	14.8125	-0.02212	-0.42194
7	14.75	14.8125	0.099839	-0.02212	14.78125	0.038784	-0.21142
8	14.78125	14.8125	0.038784	-0.02212	14.79688	0.008312	0.105597
9	14.79688	14.8125	0.008312	-0.02212	14.80469	-0.00691	0.05277
10	14.79688	14.80469	0.008312	-0.00691	14.80078	0.0007	-0.02639
		The	**root**	**is**	**14.80078**		

(ii) For False Position Method:

Python Code:

```
import math
import numpy as np
import sys
from xlwt import Workbook
print("f(x)= (668.061/x)*(1-e^-0.146843x)")
a = int(input("Enter 1st Guess:  "))
b = int(input("Enter 2nd Guess:  "))
fxa = (668.061/a)*(1-math.exp(-0.146843*a))-40
fxb = (668.061/b)*(1-math.exp(-0.146843*b))-40
if fxa*fxb>0:
   print("wrong inital value .......",fxa*fxb,"....value must be less than 0,plz try again")
   sys.exit()

elif fxa*fxb<0:
    err = float(input("enter the error value :"))
    ite = int(input("enter the iteration :"))
    x_a = np.zeros([ite])#making array
    x_b = np.zeros([ite])#making array
    x_c = np.zeros([ite])#making array
    f_a = np.zeros([ite])
    f_b = np.zeros([ite])
    f_c = np.zeros([ite])
    real_err = np.zeros([ite])
    itern = np.zeros([ite])
    x_a[0]= a
    x_b[0]= b
    f_a[0]=fxa
    f_b[0]=fxb
    for i in range(ite):
        itern[i]=i+1
        f_a[i] = (668.061/x_a[i])*(1-math.exp(-0.146843*x_a[i]))-40
        f_b[i] = (668.061/x_b[i])*(1-math.exp(-0.146843*x_b[i]))-40
        x_c[i]= x_a[i] - (x_b[i]-x_a[i]) * f_a[i]/( f_b[i] - f_a[i])

        f_c[i] = (668.061/x_c[i])*(1-math.exp(-0.146843*x_c[i]))-40
        if i>0:
            real_err[i]=((x_c[i]-x_c[i-1])/x_c[i])*100 # error estimation = ({xc[new]-
xc(old)}/xc[new])x100

        if all ([i>0, abs(real_err[i])<err]):
            break
        elif f_c[i]==0:
            break

        if i==ite-1:
            break
        if all ([f_c[i]>0, f_a[i]>0]):
            x_a[i+1]=x_c[i]
            x_b[i+1]=x_b[i]
```

```
elif all ([f_c[i]>0, f_b[i]>0]):
    x_b[i+1]=x_c[i]
    x_a[i+1]=x_a[i]
elif all ([f_c[i]<0, f_a[i]<0]):
    x_a[i+1]=x_c[i]
    x_b[i+1]=x_b[i]
elif all ([f_c[i]<0, f_b[i]<0]):
    x_b[i+1]=x_c[i]
    x_a[i+1]=x_a[i]
#creating excel
wb = Workbook()
sheet1 = wb.add_sheet('Sheet 1')
num_of_iter=i

#writing on excel
sheet1.write(0,2,'False')
sheet1.write(0,3,'position')
sheet1.write(0,4,'method')

sheet1.write(1,0,'Number of iteration') # row, column , value
sheet1.write(1,1,'x_a')
sheet1.write(1,2,'x_b')
sheet1.write(1,3,'f(a)')
sheet1.write(1,4,'f(b)')
sheet1.write(1,5,'x_c')
sheet1.write(1,6,'f(c)')
sheet1.write(1,7,'Relative error')
```

```
    #writing values on excel
    for n in range(num_of_iter+1):

        sheet1.write(n+2,0,itern[n])
        sheet1.write(n+2,1,x_a[n])
        sheet1.write(n+2,2,x_b[n])
        sheet1.write(n+2,3,f_a[n])
        sheet1.write(n+2,4,f_b[n])
        sheet1.write(n+2,5,x_c[n])
        sheet1.write(n+2,6,f_c[n])
        sheet1.write(n+2,7,real_err[n])

    sheet1.write(n+5,2,'The')
    sheet1.write(n+5,3,'root')
    sheet1.write(n+5,4,'is')
    sheet1.write(n+5,5,x_c[i])

    #save the excel file
    wb.save('false position example.xls')

print([np.transpose(x_a)],[np.transpose(x_b)])
print ("The root is",x_c[i])
```

Result:

```
f(x)= (668.061/x)*(1-e^-0.146843x)
Enter 1st Guess:  12
Enter 2nd Guess:  16
```

```
enter the error value :0.05

enter the iteration :10

The root is 14. 80264
```

Data from the Excel sheet, named "false position example.xls"

		False	**position**	**method**			
Number of iteration	x_a	x_b	f(a)	f(b)	x_c	f(c)	Relative error
1	12	16	6.113957	-2.23025	14.93087	-0.25149	0
2	12	14.93087	6.113957	-0.25149	14.81508	-0.02715	-0.78159
3	12	14.81508	6.113957	-0.02715	14.80264	-0.00292	-0.08406
		The	**root**	**is**	**14.80264**		

<u>Discussion & Conclusion</u>: In this experiment, bisection & false position method is used to get the root of the following equation. Basic idea about the methods & implementing process is clear after this experiment. The initial guesses are taken from the user by using "input" function. To continue iterative process and to stop the iterative process, 'for' loop & 'break' function is used. To get the output in an excel sheet, each variable declared as vector so that values of each variable in every iteration can be displayed in tabular format. Thus, the experiment was successfully done.

Experiment No: 02

Experiment Name: Determining the Roots of Equation-2 (Open Methods).

Objectives:
The purpose of this experiment is:

1. To Know about Newton-Raphson and secant methods.
2. To write separate programs for Newton Raphson and secant methods.
3. To compare their performance in terms of accuracy for a given no. of iterations.
4. To compare the performance of open methods with respect to bracketing methods.

Required Software: Python

Experimental Analysis:

Problem Statement: Use the Newton-Raphson & secant method to estimate the root of $f(x) = e^{-x} - x$, employing an initial guess of $x_0 = 0$.

Solution: Required equation is given below:

$$f(x) = e^{-x} - x \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(2.1)}$$

(i) For Newton- Raphson Method:

Python Code:

```
import numpy as np

import math

from xlwt import Workbook


a = float(input("Enter a initial guess: "))

ite=int(input("Enter number of iterations: "))

tol = float(input("Enter desired percentage relative error: "))


x_a=np.zeros([ite])

x_new=np.zeros([ite])


f_a=np.zeros([ite])

f_xnew=np.zeros([ite])
```

```
rel_err=np.zeros([ite])
itern=np.zeros([ite])


def y(x):
    return math.exp(-x)-x
def dy(x):
    return (-math.exp(-x))-1
x_a[0]= a


f_a[0]=y(a)
f_xnew[0]=dy(a)
for i in range(ite):
    f_a[i]=y(x_a[i])
    f_xnew[i]= dy(x_a[i])
    x_new[i]=x_a[i]-((f_a[i])/(f_xnew[i]))


#calculating error
    if i>0:
        rel_err[i]=((x_new[i]-x_new[i-1])/x_new[i])*100
    if all([i>0,abs(rel_err[i]<tol)]):
        break
    elif f_xnew[i]==0:
        break


    if i==ite-1:
        break
    x_a[i+1]=x_new[i]


#Writing the results on an excel sheet
wb = Workbook()
```

```
sheet= wb.add_sheet("Newt_raphson")
num_of_iter = ite
sheet.write(0,2,"Newton Raphson Method")

sheet.write(1,0,"No of Iteration")
sheet.write(1,1,"Guess = a")
sheet.write(1,2,"f(a)")
sheet.write(1,3,"f'(a)")
sheet.write(1,4,"Xnew")

#column Width
sheet.col(7).width = 30032
sheet.col(0).width = 1000

for n in range( num_of_iter):
    sheet.write(n+2,0,itern[n+1])
    sheet.write(n+2,1,x_a[n])
    sheet.write(n+2,2,f_a[n])
    sheet.write(n+2,3,f_xnew[n])
    sheet.write(n+2,4,x_new[i])

sheet.write(ite+6,1,'The')
sheet.write(ite+6,2,'root')
sheet.write(ite+6,3,'is')
sheet.write(ite+6,4,x_new[i])
print("The root is",x_new[i])
wb.save('Newton Raphson example.xls')
```

Result:

```
Enter a initial guess: 0

Enter number of iterations: 10

Enter desired percentage relative error: 0.05

The root is 0.5671432904097811
```

Data from the Excel sheet, named "Newton Raphson example.xls"

		Newton Raphson Method		
No of Iteration	Guess = a	f(a)	f'(a)	Xnew
1	0	1	-2	0.567143
2	0.5	0.106531	-1.60653	0.567143
3	0.566311	0.001305	-1.56762	0.567143
4	0.567143	1.96E-07	-1.56714	0.567143
	The	root	is	0.567143

(i) <u>For Secant Method:</u>

Python Code:

```
import numpy as np

import math

from xlwt import Workbook

x1 = int(input("Enter a initial Value: "))

x2= x1-1

ite=int(input("Enter number of iterations: "))

tol = float(input("Enter desired percentage relative error: "))

def y(x):

    return math.exp(-x)-x

fnx1 = y(x1)

fnx2 = y(x2)


x_1=np.zeros([ite])

x_2=np.zeros([ite])

f_nx1=np.zeros([ite])
```

```
f_nx2=np.zeros([ite])

x_new=np.zeros([ite])
f_xnew = np.zeros([ite])

rel_err=np.zeros([ite])
itern=np.zeros([ite])

x_1[0]= x1
x_2[0]=x2
f_nx1[0]=fnx1
f_nx2[0]=fnx2

for i in range(ite):
   f_nx1[i]=y(x_1[i])
   f_nx2[i]= y(x_2[i])

   x_new[i]=x_2[i]-(x_1[i]-x_2[i])/(f_nx1[i]-f_nx2[i])*f_nx2[i] #formula of Secant
   f_xnew[i] = math.exp(-x_new[i]) - x_new[i]
   itern[i]= i+1

#calculating error
   if i>0:
      rel_err[i]=((x_new[i]-x_new[i-1])/x_new[i])*100
   if all([i>0,abs(rel_err[i]<tol)]):
      break
   elif f_xnew[i]==0:
      break
```

```
if i==ite-1:
    break
x_1[i+1]=x_2[i]
x_2[i+1]=x_new[i]
wb = Workbook()

# add_sheet is used to create sheet.
sheet4 = wb.add_sheet('Sheet 4')
num_of_iter=i

#writing on excel
sheet4.write(0,0,'Number of iteration')
sheet4.write(0,1, 'Xi')
sheet4.write(0,2, 'Xi-1')
sheet4.write(0,3, 'f(xi)')
sheet4.write(0,4, 'f(xi-1)')
sheet4.write(0,5, 'xi+1')

#writing values on excel
for n in range(num_of_iter+1):

    sheet4.write(n+1,0,itern[n])
    sheet4.write(n+1,1,x_1[n])
    sheet4.write(n+1,2,x_2[n])
    sheet4.write(n+1,3,f_nx1[n])
    sheet4.write(n+1,4,f_nx2[n])
    sheet4.write(n+1,5,x_new[n])
```

```
        sheet4.write(n+4,2,'The')
        sheet4.write(n+4,3,'root')
        sheet4.write(n+4,4,'is')
        sheet4.write(n+4,5,x_new[i])

        #save the excel file
        wb.save('Secant example.xls')
    print("The root is",x_new[i])
```

Result:

```
Enter a initial Value: 0
Enter number of iterations: 10
Enter desired percentage relative error: 0.05
The root is 0.5671432754095816
```

Data from the Excel sheet, named "Secant example.xls"

Number of iteration	Xi	Xi-1	f(xi)	f(xi-1)	xi+1
1	0	-1	1	3.718282	0.367879
2	-1	0.367879	3.718282	0.324321	0.498592
3	0.367879	0.498592	0.324321	0.108794	0.564573
4	0.498592	0.564573	0.108794	0.004031	0.567111
		The	root	is	0.567111

Experiment No: 03

Experiment Name: Solving Linear Algebraic Equations-1 (Gauss Elimination Method).

Objectives:
The purpose of this experiment is:

1. To know about Gauss Elimination method.
2. To write a generalized program for Gauss Elimination method.
3. To analyze the performance of the method.
4. To implement pivoting and scaling to improve the performance.

Required Software: Python

Experimental Analysis:

Problem Statement: Use Gauss elimination to solve

$$3x_1 - 0.1x_2 - 0.2x_3 = 7.85$$

$$0.1x_1 + 7x_2 - 0.3x_3 = -19.3$$

$$0.3x_1 - 0.2x_2 + 10x_3 = 71.4$$

Solution: The augmented matrix: (from Excel File)

3	-0.1	-0.2	7.85
0.1	7	-0.3	-19.3
0.3	-0.2	10	71.4

Python Code:

```
import math

import numpy as np

import xlrd


n=int(input ('Enter number of unknown variables: '))

#print(n)

#Initialize variables

a=np.zeros([n,n+1])

b=np.zeros([n,n+1])

B=np.zeros([n,n+1])

C=np.zeros([n,n+1])

X=np.zeros([n])
```

```
p=np.zeros([n])


#Reading data from excel file

loc = ('H:/Education/3-1 3rd Year Odd Semester - ETE 18/Numerical Method/Lab/Lab
Final/data_for_elimination_jordan_seidal.xls')


wb = xlrd.open_workbook(loc)

sheet = wb.sheet_by_index(0)


#creating matrix from the data

for i in range(sheet.ncols):

    for j in range(sheet.nrows):

        #print(sheet.cell_value(1, i))

        a[j,i]=sheet.cell_value(j, i)


#Forward Elimination

for i in range(n):

    for j in range(n+1):

        if i==0:

            b[i,j]=a[i,j]

        if i>0:

            b[i,j]=a[i,j]-(a[0,j]*(a[i,0]/a[0,0]))

if n<3:

    C=b

if n>2:

    B=b

    for k in range(n-1):

        for i in range(n):

            for j in range(n+1):
```

```
            if all ([i>k+1, j<k+1]):
                C[i,j]=B[i,j]-(B[k,j]*(B[i,k]/B[k,k]))

            if all ([i>k+1, j>k]):
                 C[i,j]=B[i,j]-(B[k+1,j]*(B[i,k+1]/B[k+1,k+1]))

            if i<k+2:
                C[i,j]=B[i,j]

        B=C

#Backward Substitution
X[n-1]=C[n-1,n]/C[n-1,n-1]
for i in range(n-2,-1,-1):
    summation=0
    for k in range(i+1,n):
        summation=summation+C[i,k]*X[k]

    X[i]=(C[i,n]-summation)/C[i,i]

print('The values of the unknown variables are respectively:')
print(X)

#Result Verification
for i in range(n):
    summation=0
    for j in range(n):
        summation=summation+a[i,j]*X[j]
    p[i]=summation-a[i,j+1]


print('The verification results are:')
print(p)
print('The implementation is correct if verification results are all zero.')
```

Result:

```
Enter number of unknown variables: 3
The values of the unknown variables are respectively: [ 3.  -2.5  7. ]
The verification results are: [0.00000000e+00 0.00000000e+00 1.42108547e-14]
The implementation is correct if verification results are all zero.
```

Experiment No: 04

Experiment Name: Solving Linear Algebraic Equations -2 (Gauss Jordan and Gauss Seidel Methods).

Objectives:
The purpose of this experiment is:

1. To know about Gauss Jordan and Gauss Seidel methods.
2. To write separate programs for Gauss Jordan and Gauss Seidel methods.
3. To analyze and compare the performance of these methods.
4. To implement pivoting and scaling to improve the performance.

Required Software: Python

Experimental Analysis:

Problem Statement: Use Gauss Jordan and Gauss Seidel method to solve

$$3x_1 - 0.1x_2 - 0.2x_3 = 7.85$$

$$0.1x_1 + 7x_2 - 0.3x_3 = -19.3$$

$$0.3x_1 - 0.2x_2 + 10x_3 = 71.4$$

Solution: The augmented matrix: (from Excel File)

3	-0.1	-0.2	7.85
0.1	7	-0.3	-19.3
0.3	-0.2	10	71.4

(i) For Gauss Jordan:

Python Code:

```
import numpy as np

import xlrd

n=int(input ('Enter number of unknown variables: '))

#Initialize variables

a=np.zeros([n,n+1])

b=np.zeros([n,n+1])

B=np.zeros([n,n,n+1])

X=np.zeros([n])

p=np.zeros([n])
```

```
#Reading data from excel file

loc = ('H:/Education/3-1 3rd Year Odd Semester - ETE 18/Numerical Method/Lab/Lab
Final/data_for_elimination_jordan_seidal.xls')

wb = xlrd.open_workbook(loc)

sheet = wb.sheet_by_index(0)


#creating matrix from the data

for i in range(sheet.ncols):

   for j in range(sheet.nrows):

      #print(sheet.cell_value(1, i))

      a[j,i]=sheet.cell_value(j, i)

#Forward Elimination

for i in range(n):

   for j in range(n+1):

      if i==0:

         b[i,j]=a[i,j]/a[0,0]

      if i>0:

         b[i,j]=a[i,j]-(a[0,j]*(a[i,0]/a[0,0]))

if n<2:

   B[n-1,:,:]=b

if n>=2:

   B[0,:,:]=b


   for k in range(n-1):

      for i in range(n):

         for j in range(n+1):

            if i==k:

               B[k+1,i,j]=B[k,i,j]

            if i==k+1:
```

```
            B[k+1,i,j]=B[k,i,j]/B[k,i,i]
          if all([i>k+1 or i<k+1]):
            B[k+1,i,j]=B[k,i,j]-(B[k,k+1,j]*(B[k,i,k+1]/B[k,k+1,k+1]))

#Gathering results
for i in range(n):
  X[i]=B[n-1,i,n]
#printing the results
print('The values of the unknown variables are respectively:')
print(X)

#Result Verification
for i in range(n):
  summation=0
  for j in range(n):
    summation=summation+a[i,j]*X[j]

  p[i]=summation-a[i,j+1]

print('The verification results are:')
print(p)
print('The implementation is correct if verification results are all zero')
```

Result:

```
Enter number of unknown variables: 3
The values of the unknown variables are respectively: [ 3.  -2.5  7. ]
The verification results are: [ 0.00000000e+00 -3.55271368e-15  1.42108547e-14]
The implementation is correct if verification results are all zero
```

(ii) For Gauss Seidel Method:

Python Code:

```
import numpy as np

import xlrd

def brcond_Seidel(E,err,n):

   a=0

   for i in range(n):

      if E[i]<err:

         a=a+1

   return a/n


#taking necessary input values from keyboard

err=float(input('Enter desired percentage relative error: '))

ite=int(input('Enter number of iterations: '))

#Reading data from excel file

loc = ('H:/Education/3-1 3rd Year Odd Semester - ETE 18/Numerical Method/Lab/Lab
Final/data_for_elimination_jordan_seidal.xls')


wb = xlrd.open_workbook(loc)

sheet = wb.sheet_by_index(0)


n=sheet.nrows   #number of unknown variables

#Initialize variables

a=np.zeros([n,n+1])

E=np.zeros([n])

rel_err=np.zeros([ite,n])

X=np.zeros([n])

x=np.zeros([ite,n])

itern=np.zeros([ite])
```

```
p=np.zeros([n])
#creating matrix from the data
for i in range(sheet.ncols):
   for j in range(sheet.nrows):
      #print(sheet.cell_value(1, i))
      a[j,i]=sheet.cell_value(j, i)

#Iteration for Gauss Seidel begins here.
for j in range(ite):
   #storing the values of iteration
   itern[j]=j+1

   for i in range(n):
      summation=0
      for k in range(n):
         if k>i or k<i:
            summation=summation+a[i,k]*x[j,k]
      x[j,i]=(a[i,n]-summation)/a[i,i]

      #Error calculation
      if j>0:
         rel_err[j,i]=((x[j,i]-x[j-1,i])/x[j,i])*100
         E[i]=rel_err[j,i]

   #Breaking condition calculation
   if j>0:
      Q=brcond_Seidel(E,err,n)
      if Q==1:
         break
```

```
    #To stop the iteration when maximum iteration is reached
    if j==ite-1:
        break
    x[j+1,:]=x[j,:]

num_of_iter=j
X=x[j,:]

print('The values of the unknown variables are respectively:')
print(X)

#Result Verification
for i in range(n):
    summation=0
    for j in range(n):
        summation=summation+a[i,j]*X[j]

    p[i]=summation-a[i,j+1]
print('The verification results are:')
print(p)
print('The implementation is correct if verification results are all zero')
```

Result:

```
Enter desired percentage relative error: 0.005
Enter number of iterations: 10
The values of the unknown variables are respectively: [ 3.00000035 -2.50000004  6.99999999]
The verification results are: [ 1.06324111e-06 -2.11648565e-07  0.00000000e+00]
The implementation is correct if verification results are all zero
```

Experiment No: 05

Experiment Name: Curve Fitting with Least Square Regression.

Objectives:
The purpose of this experiment is:
1. To know about least square regression method.
2. To write a generalized program for least square regression method.
3. To analyze and compare the performance of least square regression method for different order of polynomial.

.

Required Software: Python

Experimental Analysis:

Problem Statement: Fit a straight line to the x and y values of the following table:

x	1	2	3	4	5	6	7
y	0.5	2.5	2	4	3.5	6	5.5

Solution: Data is taken from Excel sheet.

Python Code:

```
from matplotlib import pyplot as plt

import math

import numpy as np

from numpy import array, mean,zeros

import xlrd


loc = (r"H:/Education/3-1 3rd Year Odd Semester - ETE 18/Numerical Method/Lab/Lab
Final/data_Polynomial Regression.xls")

wb = xlrd.open_workbook(loc)

sheet = wb.sheet_by_index(1)

N=sheet.ncols-1

x=zeros([N])

y=zeros([N])


#creating matrix from the data
```

```
for i in range(1,sheet.ncols):


   x[i-1]=sheet.cell_value(0, i)

   y[i-1]=sheet.cell_value(1, i)

print('x =',x) # x values from excel

print('y = ',y) # y values from excel


n = len(x)

o = int(input("order:- "))

#initializing matrix to store value of a matx, b matx , and final estimates values

a = zeros((o+1,o+1))

b = zeros(o+1)

s = zeros(o+1)


for i in range (1,o+2):

   # a matrix creating

   for j in range (1,i+1):

      k = i+j-2

      Sum =0

      for l in range (0,n):

         Sum = Sum + x[l]**k

      a[i-1,j-1] = Sum

      a[j-1,i-1] = Sum

   # b matx

   Sum = 0

   for l in range (0,n):

      Sum = Sum + y[l]*x[l]**(i-1)

   b[i-1] = Sum
```

```
#gauss Elimination
for k in range (o):
   for i in range (k+1,o+1):
      fctr = a[i,k]/a[k,k]
      for j in range(k,o+1):
         a[i,j]=  a[i,j]- fctr * a[k,j]
      b[i] =  b[i] - fctr * b[k]
#back substituion
s[o]= b[o]/a[o,o]
for i in range (o-1,-1,-1):
   Sum = b[i]
   for j in range(i+1,o+1):
      Sum = Sum - a[i,j]*s[j]
   s[i]= Sum/a[i,i]

#printing outputs
print("The polynomial equation: ")
print(' =',s[0])
for i in range(1,o+1):
   print('\t +',s[i],'x','^',[i])
print("coefficients: ")
for i in range(0,o+1):

   print("a",[i],"=",s[i])

#getting total polynomial equation to plot
def f(w):
   sum = 0
   for i in range(0,o+1):
        g = s[i]
        g = g* math.pow(w,i)
        sum = sum + g
    return sum
#plotting the curve
s_est = zeros(n)
for i in range(0,n):
    s_est[i] = f(x[i])

plt.figure(1)
plt.plot(x,y,'o')
plt.plot(x,s_est)
plt.xlabel('Values of x')
plt.ylabel('Values of y')
plt.title('Curve fitting using polynomial regression')
plt.legend(['Measured','Estimated'], loc='upper right')
plt.show()
```

Result:

```
x = [1. 2. 3. 4. 5. 6. 7.]
y =  [0.5 2.5 2.  4.  3.5 6.  5.5]
order:- 1
The polynomial equation:
 = 0.07142857142857142 + 0.8392857142857143 x ^ [1]
coefficients:
a [0] = 0.07142857142857142
a [1] = 0.8392857142857143
```

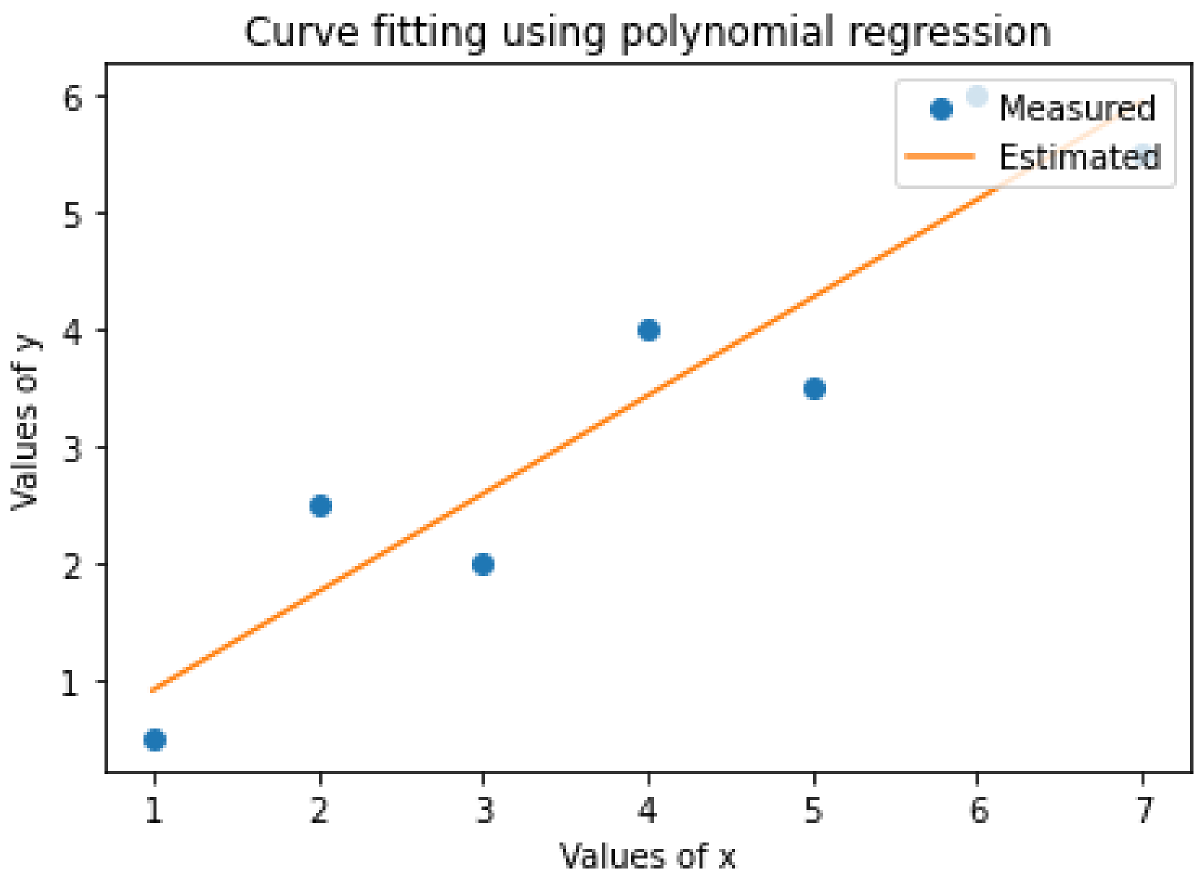

Figure 5.1: Curve Fitting using polynomial regression

Experiment No: 06

Experiment Name: Interpolation using Newton’s Polynomial and Lagrange Polynomial.

Objectives:
The purpose of this experiment is:

1. To know about Newton’s and Lagrange interpolating polynomials.
2. To write separate programs to perform interpolation using Newton’s and Lagrange polynomials of different order.
3. To analyze and compare the performance of Newton’s and Lagrange interpolating polynomial.

Required Software: Python

Experimental Analysis:

Problem Statement: Use Newton’s divided difference and Lagrange interpolating polynomials to estimate the at x=3 from the following table:

x	1.6	2	2.5	3.2	4	4.5
y=f(x)	2	8	14	15	8	2

Solution: Data is taken from Excel sheet.

(i) **For Newton’s interpolating polynomials:**

Python Code:

```
import numpy as np

import xlrd

from matplotlib import pyplot as plt

X=float(input('Enter the interpolating point: ' ))


#Reading data from excel file

loc = ('H:/Education/3-1 3rd Year Odd Semester - ETE 18/Numerical
Method/Lab/Lab Final/data_interpolation.xls')

wb = xlrd.open_workbook(loc)

sheet = wb.sheet_by_index(0)


n=sheet.ncols-1    #number of data points

#initialize variables
```

```
x=np.zeros([n])
y=np.zeros([n])
F=np.zeros([n-1,n])

#creating vectors from the data
for i in range(1,sheet.ncols):
    #print(sheet.cell_value(1, i))
    x[i-1]=sheet.cell_value(0, i)
    y[i-1]=sheet.cell_value(1, i)

#Computing divided difference
y1=y
for j in range(1,n):
    for i in range(j+1,n+1):
        F[j-1,i-1]=(y1[i-1]-y1[i-2])/(x[i-1]-x[i-1-j])
    y1=F[j-1,:]
Y=0
summation=0
b=np.zeros([n])
b[0]=1

#Computing the function value at the interpolating point
for j in range(n):
    if j==0:
        summation=y1[j]

    if j>0:
        a=F[j-1,j]
        for i in range(j):
            b[i+1]=(X-x[i])*b[i]
        summation=a*b[i+1]
    Y=summation+Y
print('The interpolating result at x = '+str(Y))
plt.figure(1)
plt.plot(x,y)
plt.plot(X,Y,'o')
plt.xlabel('Values of x')
plt.ylabel('Values of y')
plt.title('Graphical verification of the interpolation result')
plt.legend(['Measured','Estimated / Interpolated'], loc='best')
plt.show()
```

Result:

```
Enter the interpolating point: 3
The interpolating result at x = 13.611458333333335
```

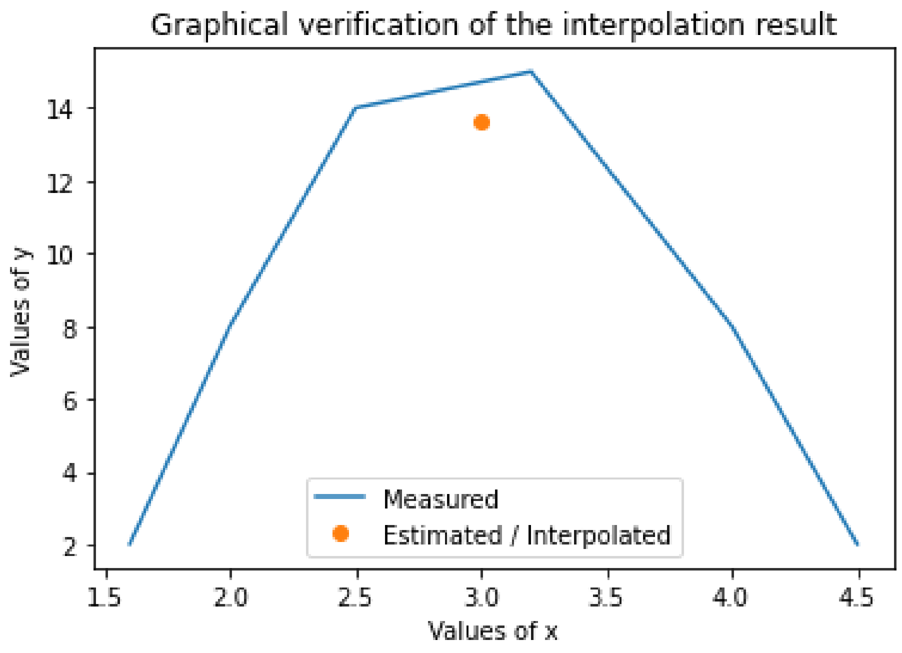

Figure 6.1: Newton’s divided-difference interpolation

(ii) **For Lagrange Polynomial:**

Python Code:

```
import numpy as np
import xlrd
from matplotlib import pyplot as plt
#taking necessary input values from keyboard
X=float(input('Enter the interpolating point: ' ))

#Reading data from excel file
loc = ('H:/Education/3-1 3rd Year Odd Semester - ETE 18/Numerical
Method/Lab/Lab Final/data_interpolation.xls')

wb = xlrd.open_workbook(loc)
sheet = wb.sheet_by_index(0)

n=sheet.ncols-1    #number of data points
#initialize variables
x=np.zeros([n])
y=np.zeros([n])
Y=0
#creating vectors from the data
for i in range(1,sheet.ncols):
    #print(sheet.cell_value(1, i))
    x[i-1]=sheet.cell_value(0, i)
    y[i-1]=sheet.cell_value(1, i)

#Performing Lagrange interpolation
for i in range(n):
    a=1
    b=1
    for j in range(n):
        if j>i:
            a=a*(X-x[j])
            b=b*(x[i]-x[j])

        if j<i:
            a=a*(X-x[j])
            b=b*(x[i]-x[j])

    Y=Y+(a/b)*y[i]
print('The interpolating result at x = '+str(Y))

plt.figure(1)
plt.plot(x,y)
plt.plot(X,Y,'o')
plt.xlabel('Values of x')
plt.ylabel('Values of y')
plt.title('Graphical verification of the interpolation result')
plt.legend(['Measured','Estimated / Interpolated'], loc='best')
plt.show()
```

Result:

```
Enter the interpolating point: 3

The interpolating result at x = 15.6114583333333333
```

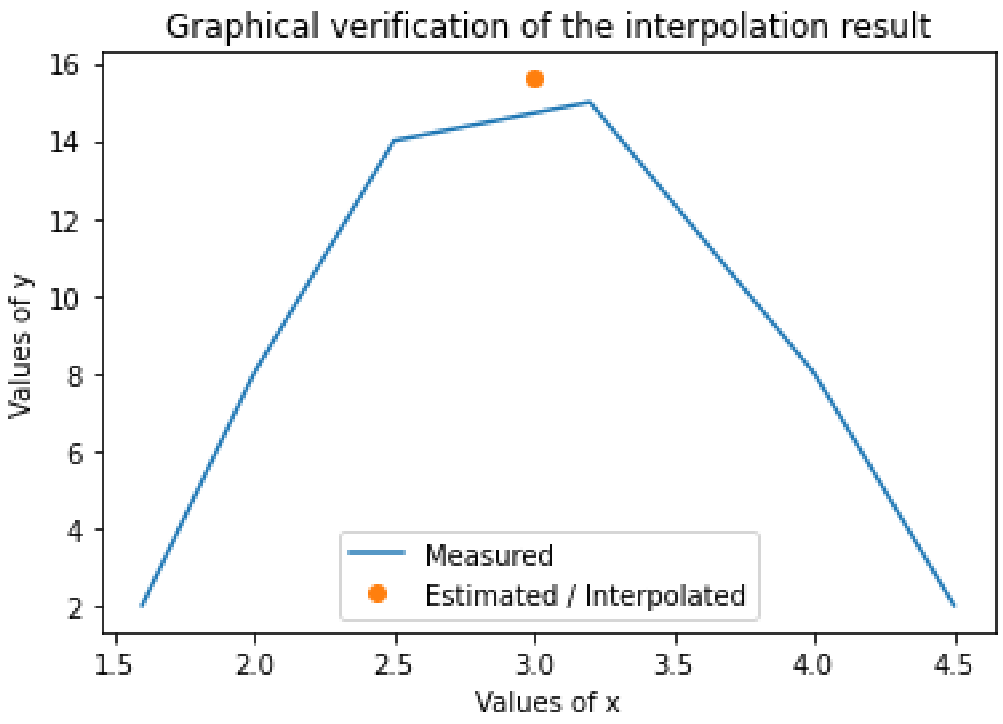

Figure 6.2: Lagrange interpolation

Experiment No: 07

Experiment Name: Integration (Trapezoidal, Simpson’s 1/3 and 3/8 Rules).

Objectives:
The purpose of this experiment is:
1. To know about trapezoidal, Simpson’s 1/3 and 3/8 rules.
2. To write separate programs for trapezoidal, Simpson’s 1/3 and 3/8 rules to perform integration.
3. To analyze and compare the performance of these methods.

Required Software: Python

Experimental Analysis:

Problem Statement: Perform trapezoidal integration:

$$\int_0^2 \frac{1}{1+x^2}$$

Solution:

Python Code:

```
def f(x):
    return 1/(1 + x**2)
# Implementing trapezoidal method
def trapezoidal(x0,xn,n):
    # calculating step size
    h = (xn - x0) / n

    # Finding sum
    integration = (f(x0) + f(xn))/2

    for i in range(1,n):
        k = x0 + i*h
        integration = integration + f(k)
    # Finding final integration value
    integration = integration * h
    return integration

# Input section
lower_limit = float(input("Enter lower limit of integration: "))
upper_limit = float(input("Enter upper limit of integration: "))
sub_interval = int(input("Enter number of sub intervals: "))

# Call trapezoidal() method and get result
result = trapezoidal(lower_limit, upper_limit, sub_interval)
print("Integration result by Trapezoidal method is: %0.6f" % (result) )
```

Result:

```
Enter lower limit of integration: 0
Enter upper limit of integration: 2
Enter number of sub intervals: 6
Integration result by Trapezoidal method is: 1.105671
```

Problem Statement: Perform Simpson 1/3 integration:

$$\int_{\mathbf{0}}^{\mathbf{0.8}} (\mathbf{0.2} + \mathbf{25}\boldsymbol{x} - \mathbf{200}\boldsymbol{x}^{\mathbf{2}} + \mathbf{675}\boldsymbol{x}^{\mathbf{3}} - \mathbf{900}\boldsymbol{x}^{\mathbf{4}} + \mathbf{400}\boldsymbol{x}^{\mathbf{5}})$$

Solution:

Python Code:

```
import numpy as np
# Defining the function
def f(x):
    return ((400*x**5)-(900*x**4)+(675*x**3)-(200*x**2)+(25*x)+(0.2))
```

```
# Input section
x0= float(input("Enter lower limit of integration: ")) # a
xn= float(input("Enter upper limit of integration: ")) # b
n= int(input("Enter number of sub intervals: "))  # n

# calculating step size
h=((xn-x0)/n)
def simpson13(x0,xn,n):
    # Finding sum
    integration = f(x0) + f(xn)

    for i in range(1,n):
        xi = x0 + i*h # xi = a+i*h

        if i%2 == 0:
          integration = integration + 2 * f(xi)
        else:
           integration = integration + 4 * f(xi)
     # Finding final integration value
    integration = integration * h/3
    return integration
result = simpson13(x0,xn,n)  # function calling
print("Integration result by Simpson's 1/3 method is: %0.6f" % (result))
```

Result:

```
Enter lower limit of integration: 0
Enter upper limit of integration: 0.8
Enter number of sub intervals: 2
Integration result by Simpson's 1/3 method is: 1.367467
```

Problem Statement: Perform Simpson 3/8 integration:

$$\int_{0}^{0.8} (0.2 + 25x - 200x^2 + 675x^3 - 900x^4 + 400x^5)$$

Solution:

Python Code:

```
lower_limit = float(input("Enter lower limit of integration: "))
upper_limit = float(input("Enter upper limit of integration: "))
sub_interval = int(input("Enter number of sub intervals: "))
def f(x):
    return 0.2+25*x-200*x**2+675*x**3-900*x**4+400*x**5
def simpson38(x0,xn,n):
    h = (xn - x0) / n
    integration = f(x0) + f(xn)
    for i in range(1,n):
        k = x0 + i*h
        if i%3 == 0:
            integration = integration + 2 * f(k)
        else:
            integration = integration + 3 * f(k)
    integration = integration * 3 * h / 8
    return integration
result = simpson38(lower_limit, upper_limit, sub_interval)
print("Integration result by Simpson's 3/8 method is: %0.6f" % (result) )
```

Result:

```
Enter lower limit of integration: 0
Enter upper limit of integration: 0.8
Enter number of sub intervals: 3
Integration result by Simpson's 3/8 method is: 1.519170
```

Experiment No: 08

Experiment Name: Differentiation (Forward, Backward and Centered Formulas).

Objectives:
The purpose of this experiment is:
1. To know about forward, backward and centered finite divided difference formulas.
2. To write a generalized program using these formulas to differentiate up to third order.
3. To analyze and compare the performance of different differentiation formulas for various orders.

Required Software: Python

Experimental Analysis:

Problem Statement: Use high accuracy differentiation formulas at x=0.5 and a step size h = 0.25.

$$\boldsymbol{f(x) = 1.2 - 0.25x - 0.5x^2 + 0.15x^3 - 0.1x^4}$$

Solution:

Python Code:

```
import numpy as np

xd = float(input("Enter the point of differentiation: " ))

h = float(input("Enter the step size: "))

def f(x):

    return -0.1*x**4-0.15*x**3-0.5*x**2-0.25*x+1.2


order=int(input("Enter the order of differentiation (1 - 3) : "))


# for 1st order

if order==1:

    method=int(input("Choose any method : \n Forward = 1 \n Backward = 2 \n Centered = 3 \n
Enter 1/2/3 :  "))

    if method == 1:

        def forward():

            diff_f=(f(xd+h)-f(xd))/h

            print(f"Forward diff= {diff_f}")
        print(forward())

    if method == 2:
        def backward():
            diff_f=(f(xd)-f(xd-h))/h
            print(f"Backward diff= {diff_f}")
        print(backward())

    if method == 3:
        def centered():
            diff_f=(f(xd+h)-f(xd-h))/(2*h)
            print(f"Centered diff= {diff_f}")
        print(centered())

# for 2nd order
elif order==2:
    method=int(input("Choose any method : \n Forward = 1 \n Backward = 2 \n Centered = 3 \n
Enter 1/2/3 :  "))
    if method == 1:
        def forward():
            diff_f=(f(xd+2*h)-2*f(xd+h)+f(xd))/(h**2)
            print(f"Forward diff= {diff_f}")
        print(forward())

    if method == 2:
        def backward():
            diff_f=(f(xd)-2*f(xd-h)+f(xd-2*h))/(h**2)
            print(f"Backward diff= {diff_f}")
        print(backward())
    if method == 3:
        def centered():
            diff_f=(f(xd+h)-2*f(xd)+f(xd-h))/(h**2)
            print(f"Centered diff= {diff_f}")
        print(centered())

# for 3rd order
elif order==3:
    method=int(input("Choose any method : \n Forward = 1 \n Backward = 2 \n Centered = 3 \n 
Enter 1/2/3 :  "))
    if method == 1:
        def forward():
            diff_f=(f(xd+3*h)-3*f(xd+2*h)+3*f(xd+h)-f(xd))/(h**3)
            print(f"Forward diff= {diff_f}")
        print(forward())

    if method == 2:
        def backward():
            diff_f=(f(xd)-3*f(xd-h)+3*f(xd-2*h)+f(xd-3*h))/(h**3)
            print(f"Backward diff= {diff_f}")
        print(backward())
    if method == 3:
        def centered():
            diff_f=(f(xd+2*h)-2*f(xd+h)+2*f(xd-h)-f(xd-2*h))/(2*h**3)
            print(f"Centered diff= {diff_f}")
        print(centered())
else:
 print('Invalid Order')
```

Result:

```
Enter the point of differentiation: 0.5

Enter the step size: 0.25

Enter the order of differentiation (1 - 3) : 1

Choose any method :

 Forward = 1

 Backward = 2

 Centered = 3

 Enter 1/2/3 :  2

Backward diff= -0.7140625000000003
```

Experiment No: 09

Experiment Name: Solving Ordinary Differential Equations (Euler’s and R-K Methods).

Objectives:
The purpose of this experiment is:

1. To know about Euler’s and R-K methods.
2. To write a separate program for Euler’s and R-K methods to solve ordinary differential equations.
3. To analyze and compare the performance of Euler’s and R-K methods for different step size and range.

Required Software: Python

Experimental Analysis:

Problem Statement 1: Use Euler’s and Modified Euler’s method to solve the following ODE from x=0 to x=4 with initial condition at x=0 is y=1.

$$\boldsymbol{y' = -2x^3 + 12x^2 - 20x + 8.5}$$

Solution:

Python Code:

```
import numpy as np

l = float(input("Enter the lower range of x: " ))

u = float(input("Enter the higher range of x: " ))

h = float(input("Enter the step size: "))


def f(x,y):

   return -2*x**3+12*x**2-20*x+8.5


n = int(((u-l)/h)+1)

x = np.zeros(n)

y = np.zeros(n)

print("\n \n Enter the intial condition :")

x[0] = input("x[0]: ")

y[0] = input("y[0]: ")
```

```
def eulers(h):
    for i in range(1,n):
        y[i]=y[i-1]+h*f(x[i-1],y[i-1])
        x[i] = x[i-1]+h
    print('Solution from Eulers Method')
    print(f"x= {x}")
    print(f"y= {y}")
print(eulers(h))


def modifiedheuns(h):
    yp = np.zeros(n)
    for i in range(1,n):
        yp[i]=y[i-1]+h*f(x[i-1],y[i-1]) # predictor
        y[i]=y[i-1]+(h/2)*(f(x[i-1],y[i-1])+f(x[i],y[i])) # corrector
        x[i] = x[i-1]+h
    print('\n Solution from Modified Eulers Method')
    print(f"x= {x}")
    print(f"y= {y}")
print(modifiedheuns(h))
```

Result:

```
Enter the lower range of x: 0
Enter the higher range of x: 4
Enter the step size: 0.5

 Enter the intial condition :
x[0]: 0
y[0]: 1
```

```
Solution from Eulers Method
x= [0.  0.5 1.  1.5 2.  2.5 3.  3.5 4. ]
y= [1.    5.25  5.875 5.125 4.5   4.75  5.875 7.125 7.   ]


 Solution from Modified Eulers Method
x= [0.  0.5 1.  1.5 2.  2.5 3.  3.5 4. ]
y= [1.     3.4375 3.375  2.6875 2.5    3.1875 4.375  4.9375 3.    ]
```

Problem Statement 2: Use 2nd oder RK schemes (Heun's, Midpoint, Ralston method) to solve the following ODE from x=0 to x=4 with initial condition at x=0 is y=1.

$$y' = -2x^3 + 12x^2 - 20x + 8.5$$

Solution:

Python Code:

```
import numpy as np
l = float(input("Enter the lower range of x: " ))
u = float(input("Enter the higher range of x: " ))
h = float(input("Enter the step size: "))


def f(x,y):
    return -2*x**3+12*x**2-20*x+8.5
n = int(((u-l)/h)+1)
x = np.zeros(n)
y = np.zeros(n)


print("\n \n Enter the intial condition :\n")
x[0] = input("x[0]: ")
y[0] = input("y[0]: ")
def heuns(h):
    for i in range(1,n):
        k1 = f(x[i-1],y[i-1])
        k2 = f(x[i-1]+h,y[i-1]+k1*h)
        y[i] = y[i-1]+0.5*h*(k1+k2)
        x[i] = x[i-1]+h
    print('Solution from Heuns Method')
    print(f"x= {x}")
    print(f"y= {y}")
print(heuns(h))
def midpoint(h):
    for i in range(1,n):
        k1 = f(x[i-1],y[i-1])
        k2 = f(x[i-1]+0.5*h,y[i-1]+0.5*k1*h)
        y[i] = y[i-1]+k2*h
        x[i] = x[i-1]+h
    print('Solution from Midpoint Method')
    print(f"x= {x}")
    print(f"y= {y}")
print(midpoint(h))


def ralston(h):
    for i in range(1,n):
        k1 = f(x[i-1],y[i-1])
        k2 = f(x[i-1]+3*h/4,y[i-1]+3*k1*h/4)
        y[i] = y[i-1]+((1/3)*k1+(2/3)*k2)*h
        x[i] = x[i-1]+h
    print('Solution from Ralston Method')
    print(f"x= {x}")
    print(f"y= {y}")
print(ralston(h))
```

Result:

```
Enter the lower range of x: 0

Enter the higher range of x: 4

Enter the step size: 0.5

 Enter the intial condition :

x[0]: 0

y[0]: 1


Solution from Heuns Method

x= [0.  0.5 1.  1.5 2.  2.5 3.  3.5 4. ]

y= [1.     3.4375 3.375  2.6875 2.5    3.1875 4.375  4.9375 3.    ]


Solution from Midpoint Method

x= [0.  0.5 1.  1.5 2.  2.5 3.  3.5 4. ]

y= [1.       3.109375 2.8125   1.984375 1.75     2.484375 3.8125   4.609375
 3.      ]


Solution from Ralston Method

x= [0.  0.5 1.  1.5 2.  2.5 3.  3.5 4. ]

y= [1.         3.27734375 3.1015625  2.34765625 2.140625   2.85546875
 4.1171875  4.80078125 3.03125   ]
```

<u>Discussion & Conclusion</u>: In this experiment, Euler’s and R-K method is used to solve ordinary differential equations. Basic idea about the methods & implementing process is clear after this experiment. The lower and upper range of x, step size and the initial condition are taken from the user by using “input” function. All of the R-K methods (Heun, Midpoint, Ralston) is performing very well and giving us almost similar result, but Euler’s method is giving us the worst result. Thus, the experiment was successfully done.