Cairo University

Faculty of Engineering

Computer Engineering Department

First Year

Numerical Differentiation using Richardson Extrapolation

Supervised by Dr. Amani Elgammal

Presented by: Team 1

Abdelrahman Jamal Sayed	sec: 1	BN: 39
Ahmed Jamal Abdelsamie	sec: 1	BN: 3
Maryem Salah	sec: 2	BN: 20
Nadeen Ayman	sec: 2	BN: 31
Tarek Yasser Ismaiel	sec: 1	BN: 37

Spring 2020

We have designed a program that computes the derivative of a function at a point using finite central difference formulas, then improved the results using Richardson's extrapolation method.

Method used:

A finite difference formula is a mathematical expression of the form $f(x+b) - f(x+a)$. If we divide this expression by $b - a$, we get the difference quotient. Approximating derivatives using finite differences plays an important role in numerically solving differential problems.

We've designed the program to compute the first, second, and third derivatives of polynomial functions using the following formulas central difference formulas respectively:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

$$f'''(x) \approx \frac{f(x+2h) - 2f(x+h) + 2f(x-h) - f(x-2h)}{2h^3}$$

Where h is the step size = $b - a$, note that the error of all three previous equations is of order $O(h^2)$.

We then applied Richardson extrapolation which is a method used to improve the performance of any iterative formula by reducing the step size h while increasing the order of error $O(h)$. Richardson extrapolation basically uses the estimates of two derivatives to compute a third, more accurate approximation.

$$A_{k,j} = \frac{4^{j-1}A_{k,j-1} - A_{k-1,j-1}}{4^{j-1} - 1}$$

This iterative formula can be represented in the following tabular form

k	h_k	$A_{k,0}$	$A_{k,1}$	$A_{k,2}$	$A_{k,3}$	$A_{k,4}$
0	h	$A_{0,0}$				
1	$h/2$	$A_{1,0}$	$A_{1,1}$			
2	$h/4$	$A_{2,0}$	$A_{2,1}$	$A_{2,2}$		
3	$h/8$	$A_{3,0}$	$A_{3,1}$	$A_{3,2}$	$A_{3,3}$	
4	$h/16$	$A_{4,0}$	$A_{4,1}$	$A_{4,2}$	$A_{4,3}$	$A_{4,4}$

Where k is the index of the vertical dimension representing the step sizes $h_k = h/2^k$ and j is the index of the horizontal dimension representing the method obtained the j level of iteration

Applications in Computer Engineering:

- Graphs and Visuals

Numerical differentiation and calculus in general are used in creating 3D graphs and visuals, which are later used in video games and physics engines. Physics engines are used to define the physics in a video game like momentum, collision, and gravity for example. Visuals are also used by the military to simulate flights, artillery paths, and satellite images. Architects may also use them to graph buildings.

- Programs to solve mathematical problems

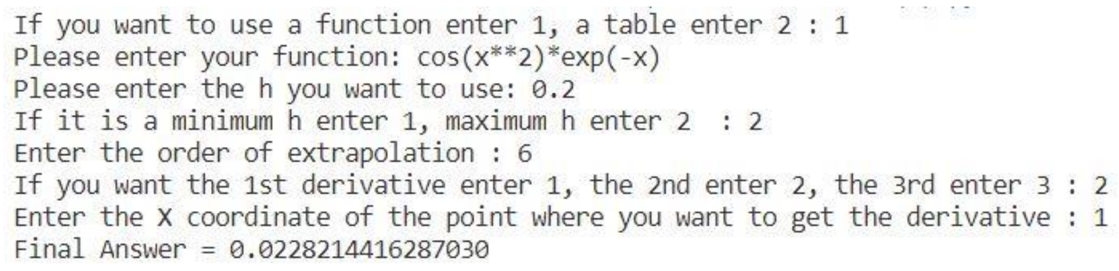
Numerical calculus is used in general problem solving programs and simulations. For instance, programs that compute integrals and derivatives like the ones used in a scientific calculator. Simulations in videogames calculate probabilities which are then solved by programs rather than the programmer himself.

- Information processing

Programmers use numerical calculus in creating statistics solvers which deal with discrete math. This is a gateway to many applications in machine learning and data mining

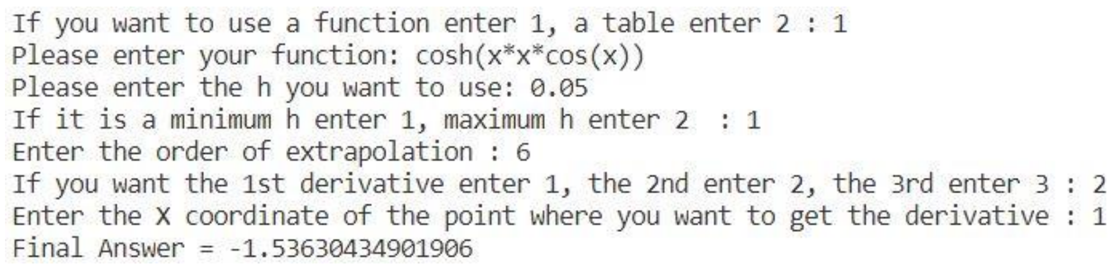
Sample Runs:

1. Evaluating $y''(1)$ for $y(x) = \cos(x^2)e^{-x}$ up to *O(h^6), h_{max} = 0.2*
   Exact answer = 0.02282142



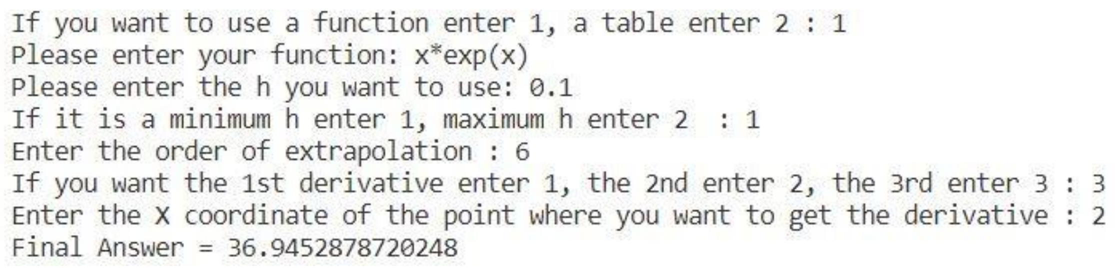
```
If you want to use a function enter 1, a table enter 2 : 1
Please enter your function: cos(x**2)*exp(-x)
Please enter the h you want to use: 0.2
If it is a minimum h enter 1, maximum h enter 2  : 2
Enter the order of extrapolation : 6
If you want the 1st derivative enter 1, the 2nd enter 2, the 3rd enter 3 : 2
Enter the X coordinate of the point where you want to get the derivative : 1
Final Answer = 0.0228214416287030
```

2. Evaluating $y''(1)$ for $y(x) = \cosh(x^2\cos(x))$ up to *O(h^6), h_{min} = 0.05*
   Exact answer = -1.536302



```
If you want to use a function enter 1, a table enter 2 : 1
Please enter your function: cosh(x*x*cos(x))
Please enter the h you want to use: 0.05
If it is a minimum h enter 1, maximum h enter 2  : 1
Enter the order of extrapolation : 6
If you want the 1st derivative enter 1, the 2nd enter 2, the 3rd enter 3 : 2
Enter the X coordinate of the point where you want to get the derivative : 1
Final Answer = -1.53630434901906
```

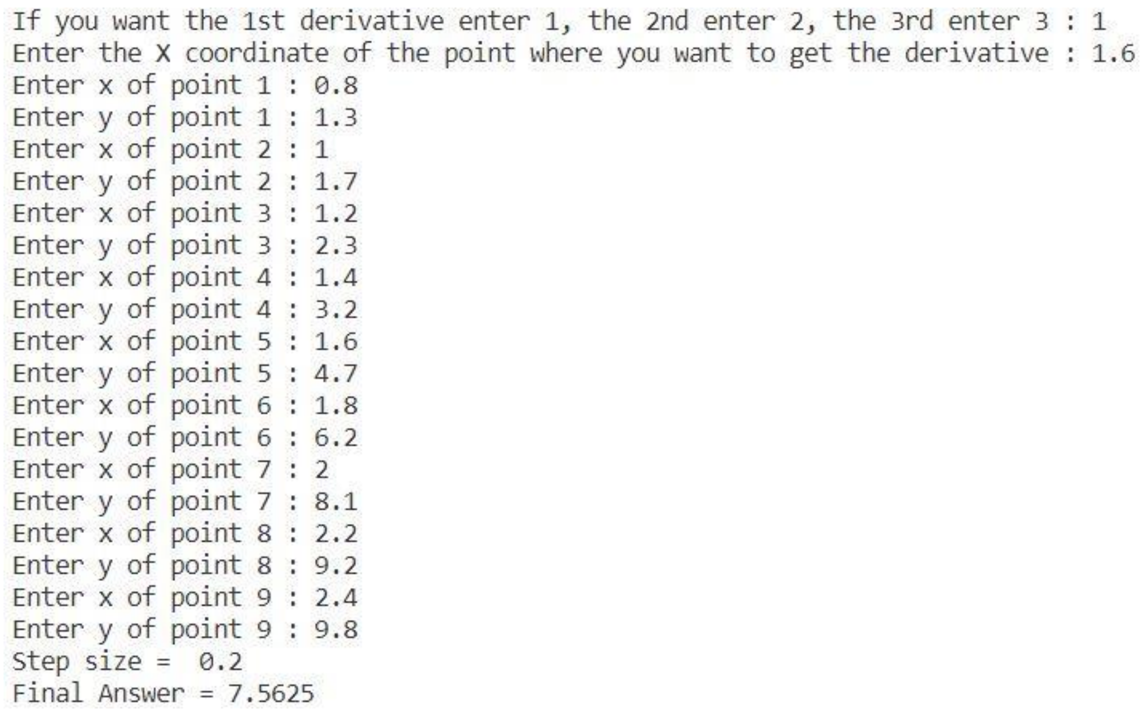
3. Evaluating $y'''(2)$ for $y(x) = xe^x$ up to *O(h^6), h_{min} = 0.1*
   Exact answer = 36.94528



```
If you want to use a function enter 1, a table enter 2 : 1
Please enter your function: x*exp(x)
Please enter the h you want to use: 0.1
If it is a minimum h enter 1, maximum h enter 2  : 1
Enter the order of extrapolation : 6
If you want the 1st derivative enter 1, the 2nd enter 2, the 3rd enter 3 : 3
Enter the X coordinate of the point where you want to get the derivative : 2
Final Answer = 36.9452878720248
```

4. Evaluating $y'(1.6)$ with *O(h^6)* for the following table

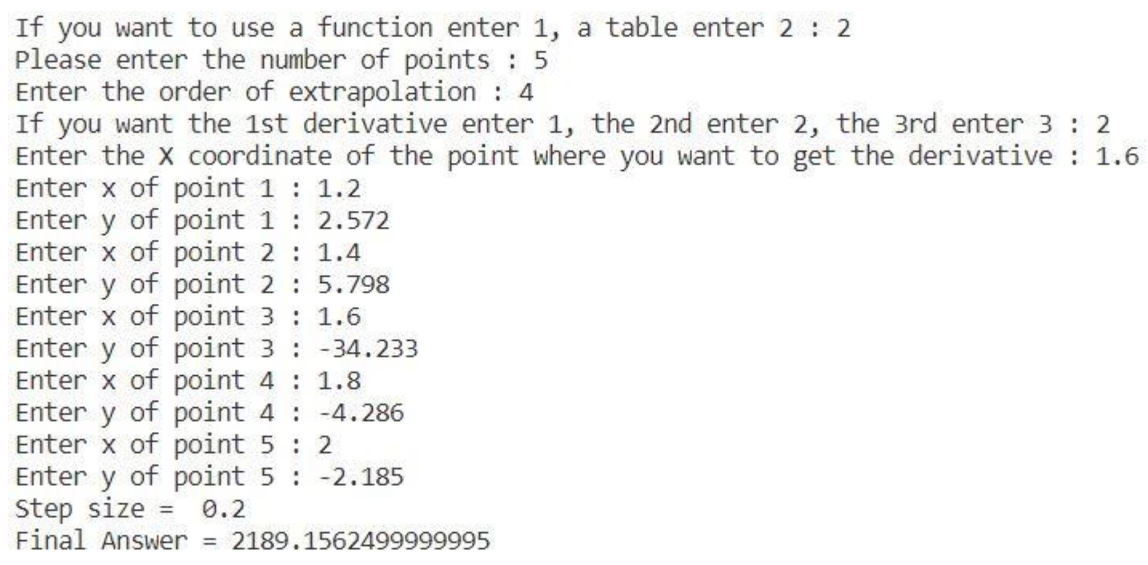
x	0.8	1	1.2	1.4	1.6	1.8	2	2.2	2.4
y	1.3	1.7	2.3	3.2	4.7	6.2	8.1	9.2	9.8



```
If you want the 1st derivative enter 1, the 2nd enter 2, the 3rd enter 3 : 1
Enter the X coordinate of the point where you want to get the derivative : 1.6
Enter x of point 1 : 0.8
Enter y of point 1 : 1.3
Enter x of point 2 : 1
Enter y of point 2 : 1.7
Enter x of point 3 : 1.2
Enter y of point 3 : 2.3
Enter x of point 4 : 1.4
Enter y of point 4 : 3.2
Enter x of point 5 : 1.6
Enter y of point 5 : 4.7
Enter x of point 6 : 1.8
Enter y of point 6 : 6.2
Enter x of point 7 : 2
Enter y of point 7 : 8.1
Enter x of point 8 : 2.2
Enter y of point 8 : 9.2
Enter x of point 9 : 2.4
Enter y of point 9 : 9.8
Step size =  0.2
Final Answer = 7.5625
```

5. Evaluating $y''(1.6)$ with *O(h^4)* for the following table

x	1.2	1.4	1.6	1.8	2
y	2.572	5.798	-34.233	-4.286	-2.185



```
If you want to use a function enter 1, a table enter 2 : 2
Please enter the number of points : 5
Enter the order of extrapolation : 4
If you want the 1st derivative enter 1, the 2nd enter 2, the 3rd enter 3 : 2
Enter the X coordinate of the point where you want to get the derivative : 1.6
Enter x of point 1 : 1.2
Enter y of point 1 : 2.572
Enter x of point 2 : 1.4
Enter y of point 2 : 5.798
Enter x of point 3 : 1.6
Enter y of point 3 : -34.233
Enter x of point 4 : 1.8
Enter y of point 4 : -4.286
Enter x of point 5 : 2
Enter y of point 5 : -2.185
Step size =  0.2
Final Answer = 2189.1562499999995
```

Code

```
from sympy import *
import numpy as np
import math
import sys

class FirstOrSecondDerivError(BaseException):
    pass
class ThirdDerivError(BaseException):
    pass
class WrongFunctionFormat(BaseException):
    pass

#Some notes:
#Exponentials should be in the form of exp, so e^(-x) = exp(-x)
#Multiplication should be done using an asterisk, so xe^x should be entered in th
e form x * exp(x) or else the final answer won't be calculated correctly.

########################################Utility Functions########################
#####################

#Returns the final answer
def Extrapolation(list, ord, table) :
   counter = int(ord / 2 - 1)
   k = 2
   if (table == False):
       ######For Formula#####
    while (k < ord / 2 + 1):
        for v in range(counter):
            list[v] = (pow(4, k - 1) * list[v] - list[v + 1]) / (
                    pow(4, k - 1) - 1)  # Calculates the extrapolation using the
general formula
            # print(RichardList_func) #Reserved for debugging
        k = k + 1
        counter = counter - 1
    return list[0]
    #print('Final Answer =', list[0])
    ######For Table#####
   else :
       while (k < ord / 2 + 1):
           for v in range(counter):
               list[0][v] = (pow(4, k - 1) * list[0][v] - list[0][v + 1]) / (
                       pow(4, k - 1) - 1)# Calculates the extrapolation using the
 general formula
           k = k + 1
       #print('Final Answer =',list[0][0])
       return list[0][0]


def CalcTrueError(ApproxValue, DerivFunc, DerivPoint, DerivOrder):
    #Deriving the exact value
    x = Symbol('x') #specifies a symbol for calculations
    init_printing(use_unicode=True) #No idea
    ExactValue = diff(DerivFunc, 'x', DerivOrder) #Obtains the analytical form of
 the derivative
    DerivEqu = lambdify(x,ExactValue) #No idea 2

    try:
      ExactValue = DerivEqu(DerivPoint)

    except (Exception, TypeError, AttributeError):
     raise WrongFunctionFormat

    if(round(ApproxValue,5) == round(ExactValue,5) == 0):
        return "0/0, Undefined value"
    if (round(ApproxValue,5) == round(ExactValue,5)):
        return 0
    elif (ExactValue == 0): #To avoid oo error
        return str(round(ApproxValue,5))+'/0'
    TrueError = abs( (ExactValue - ApproxValue)/ExactValue ) * 100 #Calculates th
e true error

    return TrueError

#Generously funded by the interface guild.
#Thanks Essam!
def Input_2D():
    xdata = []
    ydata = []
    n = int(input("Type in the no. of points : "))
    print('Insert each point as space seperated x & y')
    for i in range(0, n):
        x,y = map(float, input().split()) # splits a given input 'a b' , maps 'a'
 in x and 'b' in y
        xdata.append(float(x)) #filling the x,y lists
        ydata.append(float(y))
    return np.array(xdata),np.array(ydata) #Returns 2 numpy arrays

# Pretty much the same as "derivativeForFunction" but here we already have the po
ints and their values
def derivativeForTable(h, x, method, mat, order):
    m = mat[0].index(x)  # to get index of X(i)
    try:
        w = mat[0].index(round(x + h, 1))  # to get index of X(i+1)
        z = mat[0].index(round(x - h, 1))  # to get index of X(i-1)

    except ValueError: #Sometimes python likes to add 0.0000000000001 or somethin
g and screw with the numbers so you'd get this exception whenver python feels lik
e a dick.
        raise FirstOrSecondDerivError

        # Failed attempt at making the program more flexible
        # next2_index = mat[0].index(round(x+2*h,2))
        # if method == '1':
        # The other methods have an error of h^2
        # This one has an error of h
        # So we must compensate in the order
        # order = order**2
        # return ((4*mat[1][w] - 3* mat[1][m] - mat[1][next2_index]) / 2*h )
    try:
        if method == '1':
            return (mat[1][w] - mat[1][z]) / (2 * h)
        elif method == '2':
            return (mat[1][w] - 2 * mat[1][m] + mat[1][z]) / (pow(h, 2))
        elif method == '3':
            l = mat[0].index(round(x + (2 * h)), 5)  # to get index of X(i+2)
            g = mat[0].index(round(x - (2 * h)), 5)  # to get index of X(i-2)
            return (-
mat[1][g] + 2 * mat[1][z] - 2 * mat[1][w] + mat[1][l]) / (2 * pow(h, 3))
    except ValueError:
        raise ThirdDerivError

#Some clarification for what "method" is, it is the order of derivation, as in a
first derivative (1), second derivative (2) and so on..
def derivativeForFunction(h, x, method, formula):
    w = formula.subs({'x': float(x + h)}).evalf()  # to get index of X(i+1)
    z = formula.subs({'x': float(x - h)}).evalf()  # to get index of X(i-1)
    m = formula.subs({'x': x}).evalf()  # to get index of X(i)

    try:
     if method == '1':
        return (w - z) / (
                    2 * h)  # Uses the centeral difference formula to obtain the
first derivative (f'(x) = [f(x+h) - f(x-h)]/2h

     elif method == '2':
        return (w - 2 * m + z) / (
            pow(h, 2))  # Uses the formula f''(x) = [f(x+h) - 2f(x) + f(x-
h)]/h^2 to obtain the second derivative

     elif method == '3':  # Uses the formula [f(x-2h) + 2f(x-
h) - 2f(x+h) + f(x+2h)]/2h^3 to obtain the third derivative
        l = formula.subs({'x': float(x + (2 * h))}).evalf()  # to get index of X(
i+2)
        g = formula.subs({'x': float(x - (2 * h))}).evalf()  # to get index of X(
i-2)
        return (-1 * g + 2 * z - 2 * w + l) / (2 * pow(h, 3))
    except OverflowError:
        raise OverflowError



#################################################################################
###########################
def FuncDeriv(func, h, order, x):
    formula = sympify(func)  # converts the input function into a format sympy ca
n work with

    order = 2*(order+1) #So the order of extrapolation is entered as 1,2,3 or any
 other number
    #The extrapolating function takes the order as O(h^order), so for a first ord
er extrapolation O(h^4) the user would enter 1 and it would become 2*2 =4
    #For a second order extrapolation O(h^6), the user would enter 2 which would
become 2*3 =6
    #For a third order O(h^8), the user would enter 3 which would become 2*4 = 8


    FirstDerivList = []
    SecondDerivList = []
    ThirdDerivList = []
    try:
     for a in range(int(order / 2)):  # Loops from 0 to order/2 using the iterato
r "a"
        # Calculates the derivative using the "derivativeForFunction" function an
d add it to the list delcared previously
        # It also multiplies the step size by 2^a each iteration
        # In the end this should give us multiple derivatives using different ste
p size (multiplied by or divided by a power of 2)
        FirstDerivList.append(derivativeForFunction(h * pow(2, a), x, '1', formul
a))
        SecondDerivList.append(derivativeForFunction(h * pow(2, a), x, '2', formu
la))
        ThirdDerivList.append(derivativeForFunction(h * pow(2, a), x, '3', formul
a))
    except OverFlowError:
        return "OverFlow", "OverFlow", "OverFlow", "OverFlow", "OverFlow", "OverF
low"

    FirstDerivApprox = Extrapolation(FirstDerivList, order, False) #Extrapolates
the derivative values
    SecondDerivApprox = Extrapolation(SecondDerivList, order, False)
    ThirdDerivApprox = Extrapolation(ThirdDerivList, order, False)

    try:
      FirstDerivTrueError = CalcTrueError(FirstDerivApprox, formula, x, '1') #Cal
culates the true error
      SecondDerivTrueError = CalcTrueError(SecondDerivApprox, formula, x, '2')
      ThirdDerivTrueError = CalcTrueError(ThirdDerivApprox, formula, x, '3')

    except WrongFunctionFormat:
      # e_index = func.find('e', 0, len(func)-1)
       #for i in range(e_index+1, len(func)+2):
         #func[i+2] = func[i]
      #new_str = func[0:e_index+1]+'xp'+func[e_index+1:len(func)]
       #func[e_index+1 : e_index+2] = 'xp'
      # FuncDeriv( new_str, h, order, x)
       return "Wrong Function Format",None , "Wrong Function Format", None  ,"Wron
g Function Format", None


    return FirstDerivApprox, FirstDerivTrueError, SecondDerivApprox, SecondDerivT
rueError,ThirdDerivApprox, ThirdDerivTrueError


def TableDeriv(x, Xlist, Ylist):


    points = Xlist.size #Gets the length of the Xlist array and therefore the num
ber of points entered. I know I could've made the input function return it but I
didn't want to screw over the interface team.

    try:
        index = np.where(Xlist == x)[0][0] #Finds the index of the input point fo
r the derivative
    #Note that np.where returns a matrix of the indices where the value is found,
 hence the [0][0] to get the first occurance.

    except IndexError:
        return "Point not in table, aborting..."


    #We need to now know which side is shorter to truncate the list

    after = points - (index +1) #Sees how many indices/points are after the wante
d element, this is more compact than after = points - 1 - index
    before = index  #Finds the number of incdices/points are before the wanted el
ement. Excludes the index of the point entered as the indexing starts at 0 anyway
, no need for index - 1
    order = math.floor(math.log2(points)) * 2  #The number of extrapolations depe
nds logarithmically on the number of data points
    #We need at least 4 data points + the middle one to extrapolate once (one CD
with h and another with 2h)
    #We need 8  + the middle one to extrapolate twice (one CD with, another with
2h and another with 4h)
    #And it goes on like this..
    if (after != before): #If the point isn't centered, we need to truncate the p
oint so it's in the middle.
        diff = abs(after-before)
        #print("Aysemmtric table, need to truncate") #Tested with sheet V part I
problem 1 with points 2.2 and 1.2, both truncated fine  #For debugging
        if(after > before): #More points after than before, we need to truncate t
he Xlist and Ylist so they start from 0 to (points - diff)
            Xlist = Xlist[ : points - diff]
            Ylist = Ylist[ : points - diff]
            order = math.floor(math.log2(before*2)) * 2 #Untested, no sheet probl
ems have the required points in a place other than the middle

        else:
            Xlist = Xlist[diff:]
            Ylist = Ylist[diff:]
            order = math.floor(math.log2(after*2)) * 2

    #print(Xlist) #For debugging
    #print(Ylist) #For debugging

    mat = np.row_stack((Xlist,Ylist)) #Requested by the interface team, this take
s 2 lists and stacks them into a matrix
    mat = mat.tolist() #This was required to create the correct input for the fun
ction to work without any modifications, there might be a better way but this wor
ks fine for now.

    #print(mat) #For debugging
    #print(mat2) #For debugging

    step = round(mat[0][1] - mat[0][0], 5)

    #Preparing the lists
    FirstDerivList = []
    SecondDerivList = []
    ThirdDerivList = []

    for i in range(0, 1):
        FirstDerivList.append([])
        SecondDerivList.append([])
        ThirdDerivList.append([])

    for i in range(0, 1):
        for j in range(0, int(order / 2)):
            FirstDerivList[0].append(j)
            SecondDerivList[0].append(j)
            ThirdDerivList[0].append(j)

    for a in range(int(order / 2)):
        try:
        #Calculating the derivatives in each list
            FirstDerivList[0][a] = derivativeForTable(step * pow(2, a), x, '1', m
at, order)
            SecondDerivList[0][a] = derivativeForTable(step * pow(2, a), x, '2',
mat, order)
            ThirdDerivList[0][a] = derivativeForTable(step * pow(2, a), x, '3', m
at, order)

        except FirstOrSecondDerivError: #If the function cannot calculate x+1 or
x-1, then the first, second and third derivatives cannot be calculated
            return "Calculation point doesn't exist on the table","Calculation po
int doesn't exist on the table","Calculation point doesn't exist on the table"
        except ThirdDerivError: #If the function is able to calculate the first a
nd second derivatives but not the third, it starts over and calculates only the f
irst and the second
                                #Could be optimized by not starting over but It's
 not that big of a deal with our usecase.
            for a in range(int(order / 2)):
                FirstDerivList[0][a] = derivativeForTable(step * pow(2, a), x, '1
', mat, order)
                SecondDerivList[0][a] = derivativeForTable(step * pow(2, a), x, '
2', mat, order)
            FirstDerivApprox = Extrapolation(FirstDerivList,order , True) #Extrap
olates the derivative values
            SecondDerivApprox = Extrapolation(SecondDerivList,order , True)
            return FirstDerivApprox,SecondDerivApprox,"Calculation point doesn't
exist on the table"


    FirstDerivApprox = Extrapolation(FirstDerivList,order , True)
    SecondDerivApprox = Extrapolation(SecondDerivList,order , True)
    ThirdDerivApprox = Extrapolation(ThirdDerivList,order , True)

    return FirstDerivApprox,SecondDerivApprox,ThirdDerivApprox

#################################################################################
###########################
def Main():
    answer = input("If you want to use a function enter 1, a table enter 2 : ")
    if answer == '1':
        func = input("please enter your function: ")  # the function is input her
e as a string
        h_func = float( input("please enter the h you want to use: ") )  # step s
ize is entered here and converted from string into float
        order = int( input("Enter the order of extrapolation : ") )  # The order
of extrapolation converted into integer, note
        X = float( input("Enter the X coordinate of the point where you want to g
et the derivative : ") )  # The point we want the derivative at converted from st
ring into float


        print(FuncDeriv(func, h_func, order, X))

    elif answer == '2':
        Xlist,Ylist = Input_2D() #Asks the user to input the number of points, ta
kes the input points and returns 2 numpy arrays
        X = float( input("Enter the X coordinate of the point where you want to g
et the derivative : ") )  # The point we want the derivative at converted from st
ring into float

        print(TableDeriv(X, Xlist, Ylist))


Main()
```