Faculty of electronic engineering

Menoufia university

Egypt

Newton interpolation calculator (NIC)

Mahmoud Nasser

2nd year student

General department

Section: 12

+201062099261

Mahmoud Nasser | LinkedIn

Newton interpolation calculator or NIC for short is a simple calculator developed using python programming language. With a simple GUI created using TKinter and cx_Freeze modules. Thanks to the developers of Equation and math modules, it wasn't hard at all to develop this project.

Newton interpolation calculator is implementing Newton's forward interpolation and Newton's backward interpolation formulas to find the function values of f at any non-tabulated value of x in any interval. Demonstration of usage, source code and test values are briefly demonstrated in up coming sections.

NIC project is for educational purpose only.

This project is supervised by

Dr\ Hany Ahmed El-Gohary

• Index

- Newton’s forward and backward formulas
- NIC usage demonstration
- Examples
- Source code

Newton’s forward and backward formulas

- Newton’s forward formula

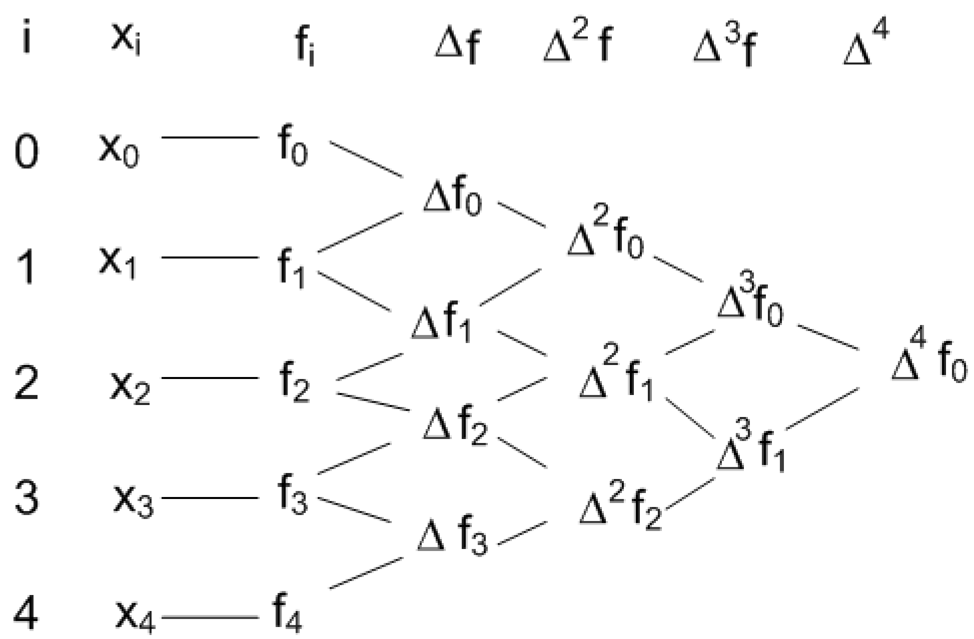
This formula is used when the required value of f(x) is near the beginning of the table

Let the function **f** is known at **n+1** equally spaced data points $a = x_0 < x_1 < \dots < \ = x_n = b$ in the interval [a,b] as $f_0, f_1, \dots f_n$. Then the **n** the degree polynomial approximation of **f(x)** can be given as

$$f(x) \cong P_n(S) = f_0 + S\Delta f_0 + \frac{S(S-1)}{2!}\Delta^2 f_0 + \cdots + \frac{S(S-1)\cdots(S-n+1)}{n!}\Delta^n f_0$$

$$where \ \ s = \frac{x - x_o}{h} \ , h = \ x_1 - x_o$$

Forward difference table: Consider the function value (x_i, f_i) $i = 0,1,2,\ldots,5$ then the forward difference table is



- ## Newton’s backward formula

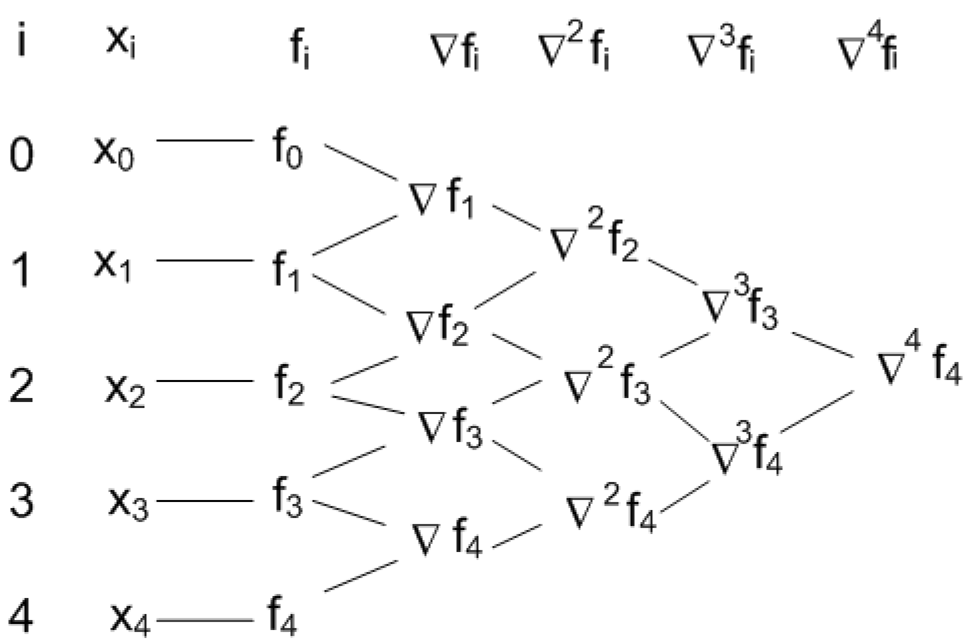
This formula is used when the required value of f(x) is near the ending of the table

Let the function **f** is known at **n+1** equally spaced data points $a = x_0 < x_1 < \ldots < \; = x_n = b$ in the interval **[a,b]** as $f_0, f_1, \ldots f_n$. Then the **n** the degree polynomial approximation of **f(x)** can be given as

$$f(x) \cong P_n(S) = f_n + s\nabla f_n + \frac{s(s+1)}{2!}\nabla^2 f_n + \cdots \ldots + \frac{s(s+1)\ldots(s+n-1)}{n!}\nabla^n f_n$$

$$\text{Where } s = \frac{x-x_n}{h}, h = x_1 - x_o$$

Backward difference table: Consider the function value (x_i, f_i) $i = 0,1,2,\ldots,5$ then the backward difference table is



NIC usage demonstration

The NIC files and directories is as following…

images	11/27/2020 5:47 PM	File folder	
lib	11/27/2020 5:47 PM	File folder	
api-ms-win-crt-heap-l1-1-0.dll	12/6/2019 10:09 PM	Application exten...	13 KB
api-ms-win-crt-locale-l1-1-0.dll	12/6/2019 10:09 PM	Application exten...	12 KB
api-ms-win-crt-math-l1-1-0.dll	12/6/2019 10:09 PM	Application exten...	21 KB
api-ms-win-crt-runtime-l1-1-0.dll	12/6/2019 10:09 PM	Application exten...	16 KB
api-ms-win-crt-stdio-l1-1-0.dll	12/6/2019 10:09 PM	Application exten...	18 KB
NIC	11/27/2020 5:47 PM	Application	31 KB
python3.dll	7/20/2020 4:03 PM	Application exten...	58 KB
python38.dll	7/20/2020 4:03 PM	Application exten...	3,957 KB
tcl86t.dll	7/20/2020 4:03 PM	Application exten...	1,304 KB
tk86t.dll	7/20/2020 4:03 PM	Application exten...	1,174 KB
vcruntime140.dll	7/20/2020 4:03 PM	Application exten...	82 KB

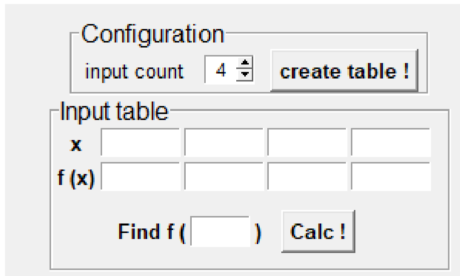
For executing the application simply double click on the **NIC.exe** file

A GUI window will start and show up as following…

In the configuration panel there is a spin box labeled **input count**

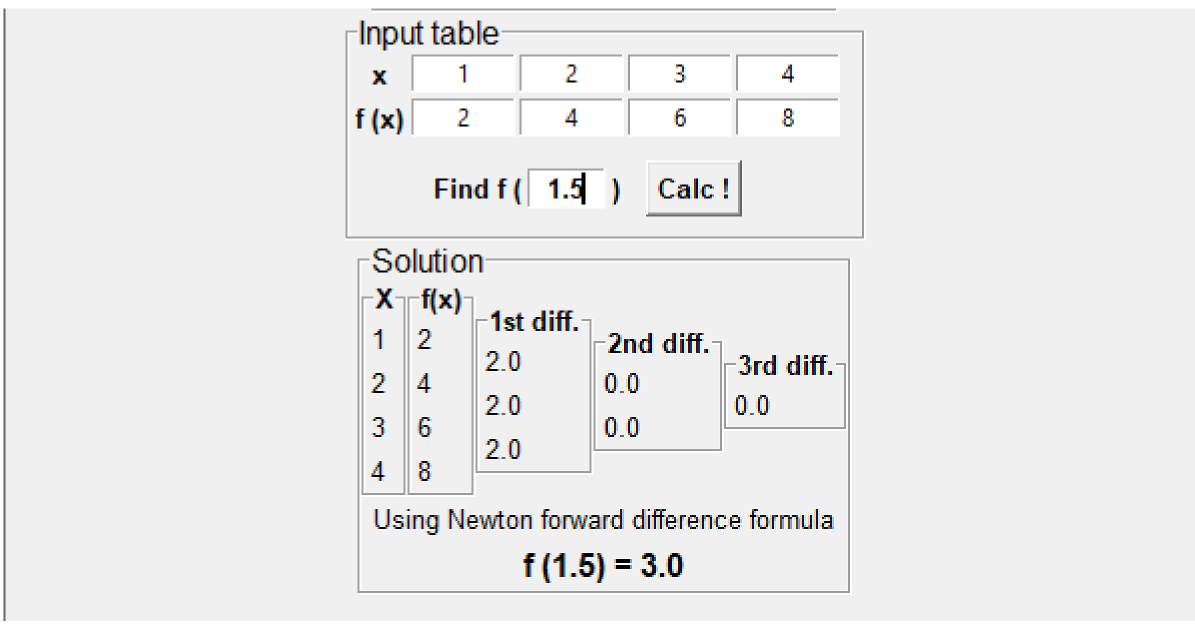
This section is to be filled with the count of x known values in the table

Afterwards, click on **create table** button which will create a table as following…



First row is to be filled with **x** values and second row is to be filled with the **f (x)** values. At the third row one input field to be filled with the **x** value which need to calculate **f(x)** value for…

After filling previous fields, click **Calc !** button to process input values and show the required **f (x)** value and the difference table used during the process as following …

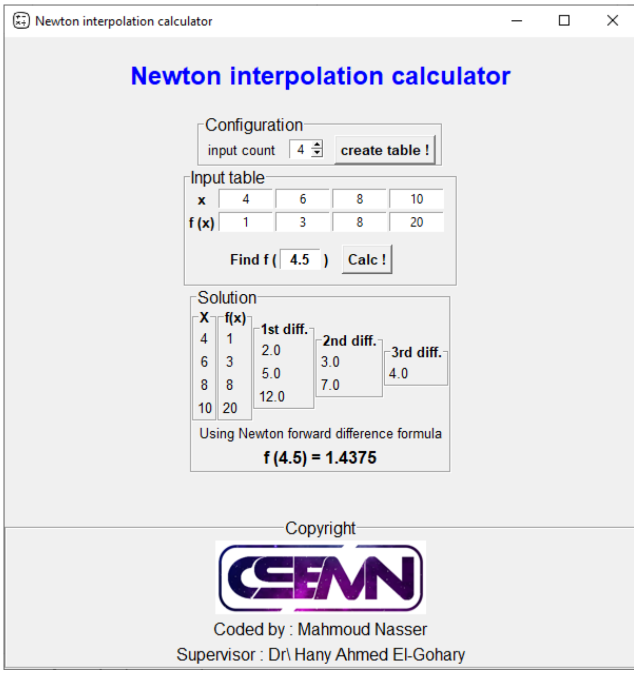


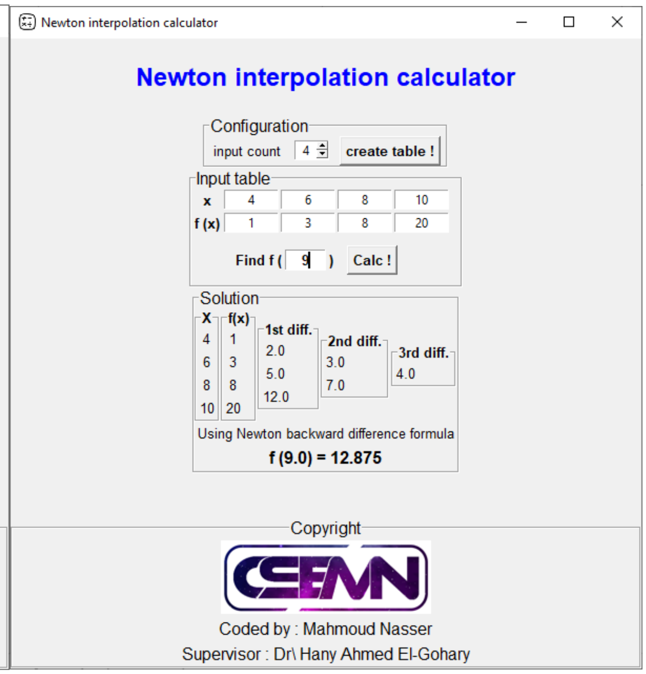
Note that the NIC show out which formula was used during the process…

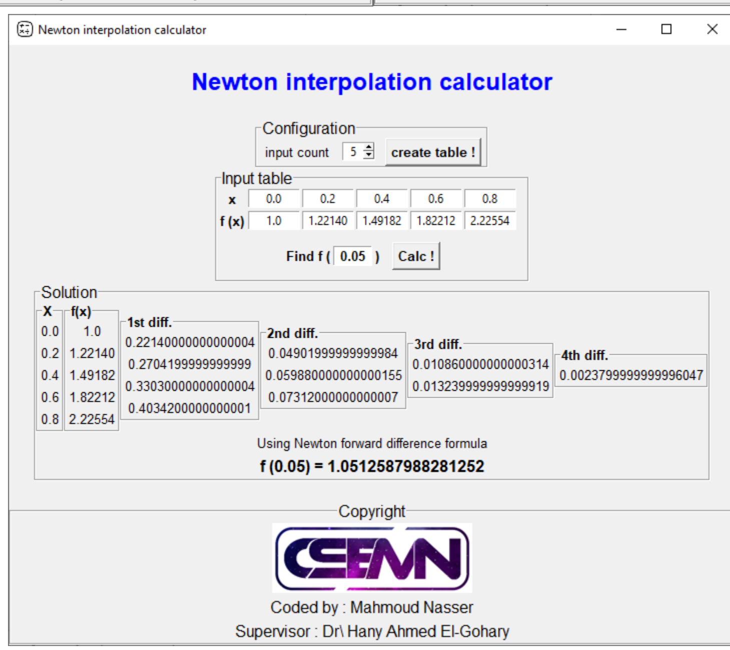
To input new problem consider clicking on **create table** again

Examples

Following 3 examples mixed between newton forward and backward formulas…







Source code

Following is the source code written in python for the NIC application

```
import math
from tkinter import Label,LabelFrame,Entry,Button,Tk,END,CENTER,Frame,messagebox,Spinbox
from PIL import ImageTk, Image
from Equation import Expression


#create a table for input -> by "create table" button
def createInputTable():

    #remove any previous results
    for child in outputTableFrame.winfo_children():
        child.destroy()
    outputTableFrame.pack_forget()
    diffTableContainer.clear()

    #check if there were previous tables and remove it
    if len(inputTableValues)>0:
        for i in inputTableValues :
            i[0].grid_remove()
            i[1].grid_remove()
        inputTableValues.clear()
    inputTableFrame.pack_forget()
    if len(calcFrameElements)>0 :
        for i in calcFrameElements:
            calcFrameElements[i].grid_forget()
    calcFrameElements.clear()

    # create new input table
    xLable = Label(inputTableFrame,text="x",font=("Arial",10,"bold"))
    xLable.grid(row=0,column=0)
    yLable = Label(inputTableFrame,text="f (x)",font=("Arial",10,"bold"))
    yLable.grid(row=1,column=0)

    for i in range(int(inputcountEnrty.get())):
        xEntry = Entry(inputTableFrame,width=8,justify=CENTER, validate = 'focusout', validat
ecommand = numValidate)
        xEntry.grid(row=0,column=i+1,padx=1,pady=1)
        yEntry = Entry(inputTableFrame,width=8,justify=CENTER, validate = 'focusout', validat
ecommand = numValidate)
        yEntry.grid(row=1,column=i+1,padx=1,pady=1)
        inputTableValues.append([xEntry,yEntry])
```

```
    calcFrame = Frame(inputTableFrame)
    calcFrameElements["lable"] = Label(calcFrame,text="Find f (",font=("Arial",10,"bold"))
    calcFrameElements["lable"].grid(row=0,column=0)
    calcFrameElements["entry"] = Entry(calcFrame,justify=CENTER,width=5,font=("Arial",10,"bol
d"), validate = 'focusout', validatecommand = numValidate)
    calcFrameElements["entry"].grid(row=0,column=1)
    calcFrameElements["lable2"]= Label(calcFrame,text=")",font=("Arial",10,"bold"))
    calcFrameElements["lable2"].grid(row=0,column=2)
    calcFrameElements["button"] = Button(calcFrame,text="Calc !",command=createOutputTable,fo
nt=("Arial",10,"bold"))
    calcFrameElements["button"].grid(row=0,column=3,padx=10)

    colspn=len(inputTableValues)+1
    calcFrame.grid(row=2,pady=10,columnspan=colspn)


    inputTableFrame.pack(ipadx=5)

def createOutputTable():

    #remove any previous results
    for child in outputTableFrame.winfo_children():
        child.destroy()
    outputTableFrame.pack_forget()
    diffTableContainer.clear()

    #validate if any empty entries

    for i in inputTableValues :
        if (i[0].get() == "" or i[1].get()==""):
            messagebox.showerror("Empty !","please, fill all input values !")
            return
    if (calcFrameElements["entry"].get()==""):
        messagebox.showerror("Empty !","please, fill f(x) value !")
        return

    calcDiffTable()

    for i in range(len(differenceTableValues)+2) : # +2 for x and y columns
        #create X and Y columns
        if i < 2 :
            if i == 0 :
                diffTableContainer[f"col{i}"] = LabelFrame(outputTableFrame, text="X",font=("
Arial",10,"bold"))
```

```
            elif i==1 :
                diffTableContainer[f"col{i}"] = LabelFrame(outputTableFrame, text="f(x)",font
=("Arial",10,"bold"))
            inputValues=[]
            for k in inputTableValues :
                inputValues.append(k[i].get())
                for j in range(0,len(inputValues)) :
                    diffTableContainer[f"col{i}-
row{j}"]= Label(diffTableContainer[f"col{i}"],text= inputValues[j],font=("Arial",10))
                    diffTableContainer[f"col{i}-row{j}"].grid(column=i,row=j)
        else:
            diffTableContainer[f"col{i}"] = LabelFrame(outputTableFrame, text=f"{ordinal[i-
1]} diff.",font=("Arial",10,"bold"))
            tmp=0
            for l in differenceTableValues[i-2] :
                diffTableContainer[f"col{i}-
row{j}"]= Label(diffTableContainer[f"col{i}"],text=l,font=("Arial",10))
                diffTableContainer[f"col{i}-row{j}"].grid(column=i,row=tmp)
                tmp+=1


        diffTableContainer[f"col{i}"].grid(row=0,column=i)
    outputTableFrame.pack()

    xVal = float(calcFrameElements["entry"].get())
    x0 = float(inputTableValues[0][0].get())
    xn = float(inputTableValues[len(inputTableValues)-1][0].get())

    if ( abs( x0- xVal ) < abs( xn - xVal) ) :
        result = calcNewtonForward()
        diffTableContainer["method"] = Label(outputTableFrame,text="Using Newton forward diff
erence formula",font=("Arial",10))
    elif ( abs( x0- xVal ) > abs( xn - xVal) ):
        result = calcNewtonBackward()
        diffTableContainer["method"] = Label(outputTableFrame,text="Using Newton backward dif
ference formula",font=("Arial",10))
    else :
        diffTableContainer["method"] = Label(outputTableFrame,text="Can't calculate values in
 the middle of the table",font=("Arial",10))
        result="Unknown !"


    diffTableContainer["result"] = Label(outputTableFrame,text=(f"f ({xVal}) = {result}"),fon
t=("Arial",12,"bold"))
    colSpan=len(differenceTableValues)+2
    diffTableContainer["method"].grid(column=0,row=1,columnspan=colSpan)
```

```python
    diffTableContainer["result"].grid(column=0,row=2,columnspan=colSpan)



#create and calculate the difference table
def calcDiffTable():

    differenceTableValues.clear()

    #obtains y values from the input table
    yValues=[]
    for i in inputTableValues :
        yValues.append(float(i[1].get()))

    #calculate delta y values
    deltaYValues=[]
    for j in range(len(yValues)-1) :
        deltaYValues.append(yValues[j+1]-yValues[j])

    differenceTableValues.append(deltaYValues)

    for i in range(len(yValues)-2) :
        deltaYValues=[]
        for k in range(len(differenceTableValues[i])-1) :
            deltaYValues.append(differenceTableValues[i][k+1] - differenceTableValues[i][k])

        differenceTableValues.append(deltaYValues)

#to return s(s-1)(s-2)....
def factOfS(rep) :
    if rep==0 :
        return "S"
    elif rep > 0 :
        return (f"(S-{rep})"+" * "+factOfS(rep-1))

def calcNewtonForward():

    xVal=float(calcFrameElements["entry"].get()) # desired value to obtain f(x) at
    x0 = float(inputTableValues[0][0].get())
    x1 = float(inputTableValues[1][0].get())
    h  = x1-x0
    sVal = (xVal-x0) / h
    f_x0 = float(inputTableValues[0][1].get())
    p_x  = f_x0
    times= len(differenceTableValues)
    for i in range(times) :
```

```
        deltaY  = float(differenceTableValues[i][0])
        factOfSVal = Expression(factOfS(i))(sVal)
        p_x += ((factOfSVal / math.factorial(i+1) ) * deltaY)
    return p_x

def calcNewtonBackward():

    xVal=float(calcFrameElements["entry"].get()) # desired value to obtain f(x) at
    x0 = float(inputTableValues[0][0].get())
    x1 = float(inputTableValues[1][0].get())
    xn = float(inputTableValues[len(inputTableValues)-1][0].get())
    h  = x1-x0
    sVal = (xVal-xn) / h
    f_xn = float(inputTableValues[len(inputTableValues)-1][1].get())
    p_x  = f_xn
    times= len(differenceTableValues)
    for i in range(times) :
        deltaY  = float(differenceTableValues[i][(len(differenceTableValues[i])-1)])
        factOfSVal = Expression(factOfS(i).replace("-","+"))(sVal)  #replace s(s-1)(s-
2)... by s(s+1)(s+2)...
        p_x += ((factOfSVal / math.factorial(i+1) ) * deltaY)
    return p_x

def validate(action, index, value_if_allowed,
                       prior_value, text, validation_type, trigger_type, widget_name):
        if value_if_allowed:
            try:
                float(value_if_allowed)
                return True
            except ValueError:
                messagebox.showerror("Invalid input","please, Enter only numbers")
                return False
        else:
            return False

main_window = Tk()
main_window.title("Newton interpolation calculator")
main_window.geometry("600x600")
main_window.iconbitmap(".\\images\\NIC.ico")

numValidate = (main_window.register(validate),'%d', '%i', '%P', '%s', '%S', '%v', '%V', '%W')
 #number validation
bigLable = Label(main_window,text="Newton interpolation calculator",fg="Blue",font=('Arial',1
8,"bold"),anchor="center",pady=20)
bigLable.pack()
```

```
#Configuration frame
#contains input count and creat input table
configFrame= LabelFrame(main_window, text="Configuration",font=("Arial",12))
inputcountLable = Label(configFrame,text="input count",font=("Arial",10))
inputcountLable.grid(row=0,column=0,padx=5)
inputcountEnrty = Spinbox(configFrame,from_= 2, to = 10,width=3,justify=CENTER, validate = 'f
ocusout', validatecommand = numValidate)
inputcountEnrty.delete(0)
inputcountEnrty.insert(END,"4")
inputcountEnrty.grid(row=0,column=1,padx=5)
inputcountButton = Button(configFrame,text="create table !",command=createInputTable,font=('A
rial',10,"bold"))
inputcountButton.grid(row=0,column=2,padx=5)
configFrame.pack()
#End of configuration frame

#input table frame
#contains values of X and f(x) entered by user
inputTableFrame= LabelFrame(main_window, text="Input table",font=("Arial",12))
inputTableValues= []
calcFrameElements={}

#output table frame
#contains values of X and f(x) and delta(n) of x columns
outputTableFrame= LabelFrame(main_window, text="Solution",font=("Arial",12))
differenceTableValues= []
diffTableContainer={}

ordinal=("0th","1st","2nd","3rd","4th","5th","6th","7th","8th","9th","10th") #tuple of ordina
l numbers

#copyright section
copyrightFrame = LabelFrame(main_window,text="Copyright",labelanchor='n',font=("Arial",12))
img = ImageTk.PhotoImage(Image.open(".\\images\\logo.png").resize((200,70)))
Label(copyrightFrame,image=img).pack()
Label(copyrightFrame,font=("Arial",12),text="Coded by : Mahmoud Nasser").pack()
Label(copyrightFrame,font=("Arial",12),text="Supervisor : Dr\\ Hany Ahmed El-Gohary").pack()

copyrightFrame.pack(fill="x",side="bottom")
#Main loop
main_window.mainloop()
```