

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 in addition to Stirling central difference formula 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**
- **Stirling central difference formula**
- **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

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

$$\text{where } S = \frac{x - x_0}{h}, h = x_1 - x_0$$

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

i	x_i	f_i	Δf	$\Delta^2 f$	$\Delta^3 f$	Δ^4
0	x_0	f_0				
1	x_1	f_1	Δf_0	$\Delta^2 f_0$	$\Delta^3 f_0$	
2	x_2	f_2	Δf_1	$\Delta^2 f_1$	$\Delta^3 f_1$	$\Delta^4 f_0$
3	x_3	f_3	Δf_2	$\Delta^2 f_2$		
4	x_4	f_4	Δf_3			

- 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 $a = x_0 < x_1 < \dots < x_n = b$ 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

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

$$\text{Where } s = \frac{x-x_n}{h}, h = x_1 - x_0$$

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

i	x_i	f_i	∇f_i	$\nabla^2 f_i$	$\nabla^3 f_i$	$\nabla^4 f_i$
0	x_0	f_0				
1	x_1	f_1	∇f_1	$\nabla^2 f_2$	$\nabla^3 f_3$	
2	x_2	f_2	∇f_2	$\nabla^2 f_3$	$\nabla^3 f_4$	$\nabla^4 f_4$
3	x_3	f_3	∇f_3	$\nabla^2 f_4$		
4	x_4	f_4				

Stirling central difference formula

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

$$\begin{aligned}
 p_n(x) &= P_{2m+1}(x) \\
 &= f[x_0] + \frac{sh}{2}(f[x_{-1}, x_0] + f[x_0, x_1]) + s^2 h^2 f[x_{-1}, x_0, x_1] \\
 &\quad + \frac{s(s^2-1)h^3}{2}(f[x_{-2}, x_{-1}, x_0, x_1] + f[x_{-1}, x_0, x_1, x_2]) \\
 &\quad + \dots + s^2(s^2-1)(s^2-4)\dots(s^2-(m-1)^2)h^{2m}f[x_{-m}, \dots, x_m] \\
 &\quad + \frac{s^2(s^2-1)\dots(s^2-m^2)h^{2m+1}}{2}(f[x_{-m-1}, \dots, x_m] \\
 &\quad + f[x_{-m}, \dots, x_{m+1}]).
 \end{aligned}$$

Divided difference table: Consider the function value (x_i, f_i) $i = \dots, -2, -1, 0, 1, 2, \dots$ then the Divided difference table is

x	$f(x)$	1 st Divided	2 nd Divided	3 rd Divided	4 th Divided
x_{-2}	$f[x_{-2}]$	$f[x_{-2}, x_{-1}]$			
x_{-1}	$f[x_{-1}]$	$f[x_{-1}, x_0]$	$f[x_{-2}, x_{-1}, x_0]$	$f[x_{-2}, x_{-1}, x_0, x_1]$	
x_0	$f[x_0]$	$f[x_0, x_1]$	$f[x_{-1}, x_0, x_1]$	$f[x_{-1}, x_0, x_1, x_2]$	$f[x_{-2}, x_{-1}, x_0, x_1, x_2]$
x_1	$f[x_1]$	$f[x_1, x_2]$	$f[x_0, x_1, x_2]$		
x_2	$f[x_2]$				

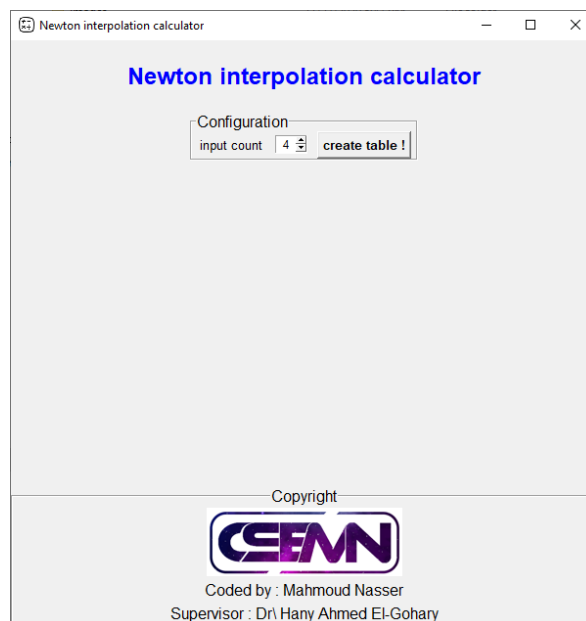
NIC usage demonstration

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...

The screenshot shows a software interface with two main sections. The top section, titled 'Configuration', contains a spin box for 'input count' set to 4 and a 'create table !' button. The bottom section, titled 'Input table', contains two rows of input fields. The first row is labeled 'x' and the second row is labeled 'f(x)'. Below these rows is a 'Find f ()' label with an input field and a 'Calc !' button.

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 ...

The screenshot shows the 'Input table' section with the following data:

x	1	2	3	4
f(x)	2	4	6	8

Below the table, the 'Find f (1.5)' field is filled with 1.5, and the 'Calc !' button is visible.

The 'Solution' section displays the following difference table:

X	f(x)	1st diff.	2nd diff.	3rd diff.
1	2			
2	4	2.0		
3	6	2.0	0.0	
4	8	2.0	0.0	0.0

Below the table, it states 'Using Newton forward difference formula' and displays the result: **f(1.5) = 3.0**

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 for newton forward, backward and Sterling problems...

Newton interpolation calculator

Configuration
input count 4 create table !

Input table

x	4	6	8	10
f(x)	1	3	8	20


Find f (4.5) Calc !

Solution

X	f(x)	1st diff.	2nd diff.	3rd diff.
4	1	2.0	3.0	4.0
6	3	5.0	7.0	
8	8	12.0		
10	20			

Using Newton forward difference formula
f(4.5) = 1.4375

Copyright



Coded by : Mahmoud Nasser
Supervisor : Dr/ Hany Ahmed El-Gohary

Newton interpolation calculator

Configuration
input count 4 create table !

Input table

x	4	6	8	10
f(x)	1	3	8	20


Find f (9) Calc !

Solution

X	f(x)	1st diff.	2nd diff.	3rd diff.
4	1	2.0	3.0	4.0
6	3	5.0	7.0	
8	8	12.0		
10	20			

Using Newton backward difference formula
f(9.0) = 12.875

Copyright



Coded by : Mahmoud Nasser
Supervisor : Dr/ Hany Ahmed El-Gohary

Newton interpolation calculator

Configuration
input count 5 create table !

Input table

x	1	1.3	1.6	1.9	2.2
f(x)	0.7651977	0.620086	0.4554022	0.2818186	0.1103623


Find f (1.5) Calc !

Solution

X	f(x)	1st div.	2nd div.	3rd div.	4th div.
1	0.7651977	-0.4837056666666664	-0.10873388888888935	0.06587839506172917	0.0018251028806576597
1.3	0.620086	-0.548946	-0.04944333333333309	0.06806851851851836	
1.6	0.4554022	-0.5786119999999999	0.011818333333333448		
1.9	0.2818186	-0.5715209999999998			
2.2	0.1103623				

Using Sterling central difference formula
f(1.5) = 0.5118199942386832

Copyright



Coded by : Mahmoud Nasser
Supervisor : Dr/ Hany Ahmed El-Gohary

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
    xLabel = Label(inputTableFrame,text="x",font=("Arial",10,"bold"))
    xLabel.grid(row=0,column=0)
    yLabel = Label(inputTableFrame,text="f (x)",font=("Arial",10,"bold"))
    yLabel.grid(row=1,column=0)

    for i in range(int(inputcountEntry.get())):
        xEntry = Entry(inputTableFrame,width=8,justify=CENTER, validate = 'focusout', validatecommand = numValidate)
        xEntry.grid(row=0,column=i+1,padx=1,pady=1)
```

```

        yEntry = Entry(inputTableFrame,width=8,justify=CENTER, validate = 'focusout', validatecommand = 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,"bold"), 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,font=("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

    xValues = []
    for x in inputTableValues :
        xValues.append(float(x[0].get()))

```

```

xVal = float(calcFrameElements["entry"].get()) # value of X to obtain f(x) for...
x0 = float(inputTableValues[0][0].get()) #first value of X
xn = float(inputTableValues[len(inputTableValues)-1][0].get()) #last value of X
Xc = float(inputTableValues[int(len(inputTableValues)/2)][0].get()) # value of centered X
Xbc = float(inputTableValues[int(len(inputTableValues)/2)-
1][0].get()) # value before centered X
Xac = float(inputTableValues[int(len(inputTableValues)/2)+1][0].get()) # value after centered X

if xVal in xValues : # if found in input table
    diffTableContainer["method"] = Label(outputTableFrame,text="Found in input table !",font=("Arial",10))
    result = float(inputTableValues[xValues.index(xVal)][1].get())
    diffTableContainer["result"] = Label(outputTableFrame,text=f"f ({xVal}) = {result}",font=("Arial",12,"bold"))
    colSpan=len(differenceTableValues)+2
    diffTableContainer["method"].grid(column=0,row=1,columnspan=colSpan)
    diffTableContainer["result"].grid(column=0,row=2,columnspan=colSpan)
    outputTableFrame.pack()
    return
elif (len(xValues)%2 != 0 and (( xVal < Xc and (xVal > Xbc ) or ( xVal > Xc and (xVal < Xac ) ) ) ): # if xVal near center of table
    diffTableContainer["method"] = Label(outputTableFrame,text="Using Sterling central difference formula",font=("Arial",10))
    #use sterling
    calcDivTable()
    tableType="div."
    result = calcSterling()
elif ( abs( x0- xVal ) < abs( xn - xVal ) ) :
    calcDiffTable()
    result = calcNewtonForward()
    diffTableContainer["method"] = Label(outputTableFrame,text="Using Newton forward difference formula",font=("Arial",10))
    tableType="diff."
elif ( abs( x0- xVal ) > abs( xn - xVal ) ):
    calcDiffTable()
    result = calcNewtonBackward()
    diffTableContainer["method"] = Label(outputTableFrame,text="Using Newton backward difference formula",font=("Arial",10))
    tableType="diff."
else :
    diffTableContainer["method"] = Label(outputTableFrame,text="Odd input count needed for central difference",font=("Arial",10))
    result="Unknown !"

```

```

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

    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]} {tableType}",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()

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

#calculate the divided table
def calcDivTable():

    differenceTableValues.clear()

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

    x0 = float(inputTableValues[0][0].get())
    x1 = float(inputTableValues[1][0].get())
    h = x1-x0

    #calculate divided y values
    divYValues=[]
    for j in range(len(yValues)-1) :
        divYValues.append((yValues[j+1]-yValues[j])/h)

    differenceTableValues.append(divYValues)

```

```

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

    differenceTableValues.append(divYValues)

#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 specialFactS(rep): #  $S(S^2 - 1^2)(s^2 - 2^2)\dots$ 

    if rep==0 :
        return "S"
    elif rep > 0 :
        return (f"(S^(2)-{rep}^2))"+" * "+specialFactS(rep-1))

def calcSterling():

    xVal=float(calcFrameElements["entry"].get()) # desired value to obtain f(x) at
    x0 = float(inputTableValues[0][0].get())
    x1 = float(inputTableValues[1][0].get())
    centerIndex = int(len(inputTableValues)/2)
    xc = float(inputTableValues[centerIndex][0].get()) # x value at center of table
    h = x1-x0
    sVal = (xVal-xc)/h
    f_xc = float(inputTableValues[centerIndex][1].get()) # f(x) at center of the table
    p_x = f_xc
    evenSIitiration = 0
    oddSIitiration = 0
    times= len(differenceTableValues)
    for i in range(times) :
        current_H = h**(i+1)

        if (i%2 == 0) :
            divY_1 = float(differenceTableValues[i][((int(len(differenceTableValues[i])/2) -
1))])
            divY_2 = float(differenceTableValues[i][((int(len(differenceTableValues[i])/2) )
))]
            current_div = (divY_1 + divY_2) / 2
            termOfSval = Expression(specialFactS(evenSIitiration))(sVal)
            evenSIitiration += 1

        else :
            current_div = float(differenceTableValues[i][((int(len(differenceTableValues[i])/
2)))]
            termOfSval = Expression(specialFactS(oddSIitiration) + " * S")(sVal)
            oddSIitiration += 1

        finalValue = current_H * current_div * termOfSval
        p_x += finalValue
    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.minsize(600,600)
main_window.iconbitmap(".\\images\\NIC.ico")

numValidate = (main_window.register(validate), '%d', '%i', '%P', '%S', '%S', '%v', '%V', '%W')
    #number validation
bigLabel = Label(main_window, text="Newton interpolation calculator", fg="Blue", font=('Arial', 18, "bold"), anchor="center", pady=20)
bigLabel.pack()

#Configuration frame
#contains input count and creat input table
configFrame= LabelFrame(main_window, text="Configuration", font=("Arial", 12))
inputcountLabel = Label(configFrame, text="input count", font=("Arial", 10))
inputcountLabel.grid(row=0, column=0, padx=5)
inputcountEntry = Spinbox(configFrame, from_= 2, to = 10, width=3, justify=CENTER, validate = 'focusout', validatecommand = numValidate)
inputcountEntry.delete(0)
inputcountEntry.insert(END, "4")
inputcountEntry.grid(row=0, column=1, padx=5)
inputcountButton = Button(configFrame, text="create table !", command=createInputTable, font=('Arial', 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 ordinal 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()

```