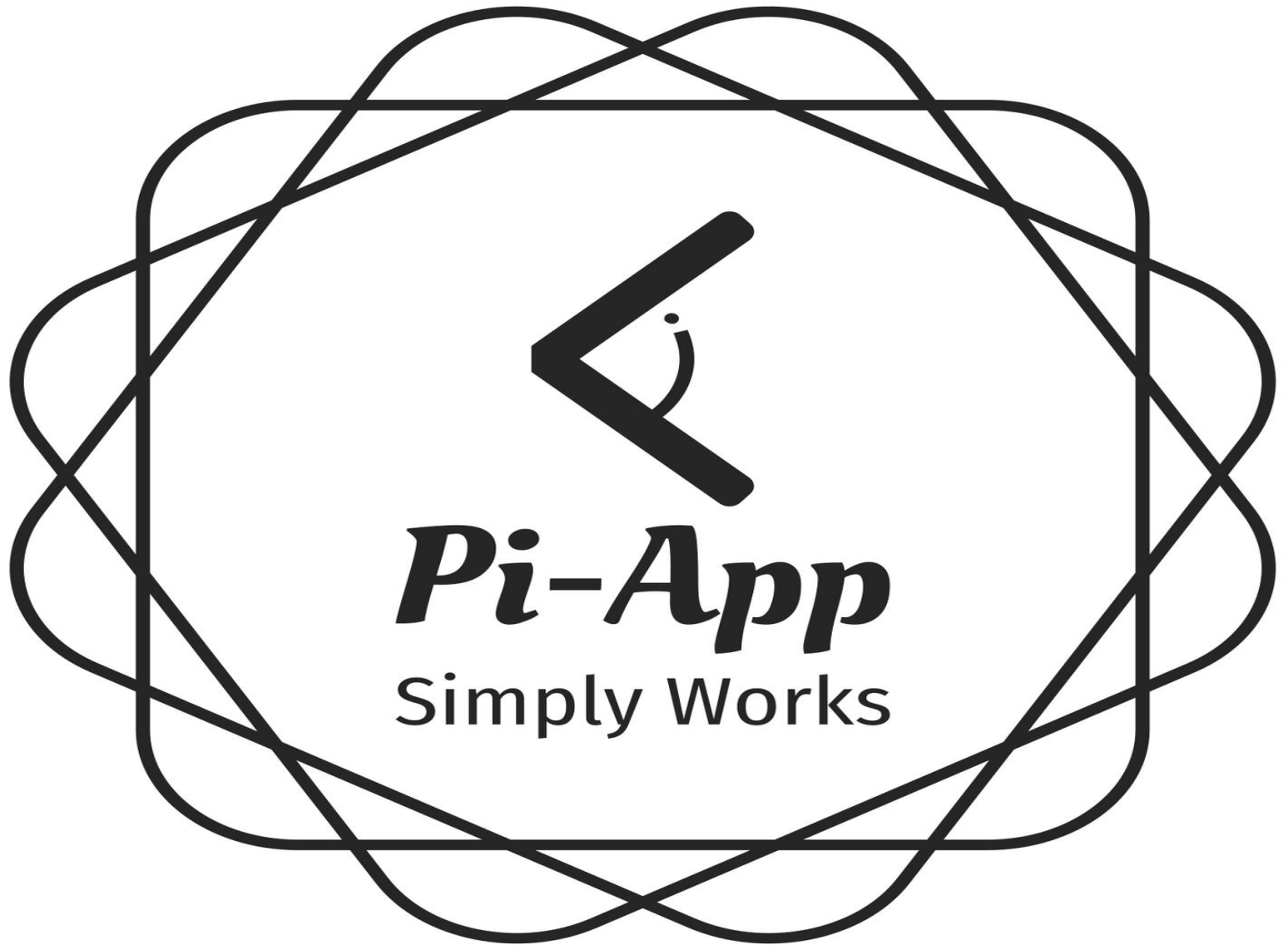


# Π Desktop Application



# TABLE OF CONTENTS

## 1. OVERVIEW AND USER MANUAL

- 1) Introduction to our Application
- 2) Why  $\pi$
- 3) Features
- 4) How to use

## 2. CODE

- 1) Structure of the code
- 2) Code screenshots
- 3) Test cases

## 3. WHAT'S NEXT

- 1) Our vision
- 2) Features to be added
- 3) Mobile Application?

## 4. NEWTON'S METHOD

- 1) Aim of Newton's method
- 2) Code Screenshots
- 3) Test Cases

## 5. Trapezoidal Method

- 1) Aim of Trapezoidal method
- 2) Code Screenshots
- 3) Test Cases

*Pi-App*  
Simply Works



# 1. OVERVIEW AND USER MANUAL

## 1.1. Introduction to our Application

Our app is a simple mathematic tool helps the students to have fast, simple and powerful tool to have a great understand of the Linear regression concepts and the linearized models.

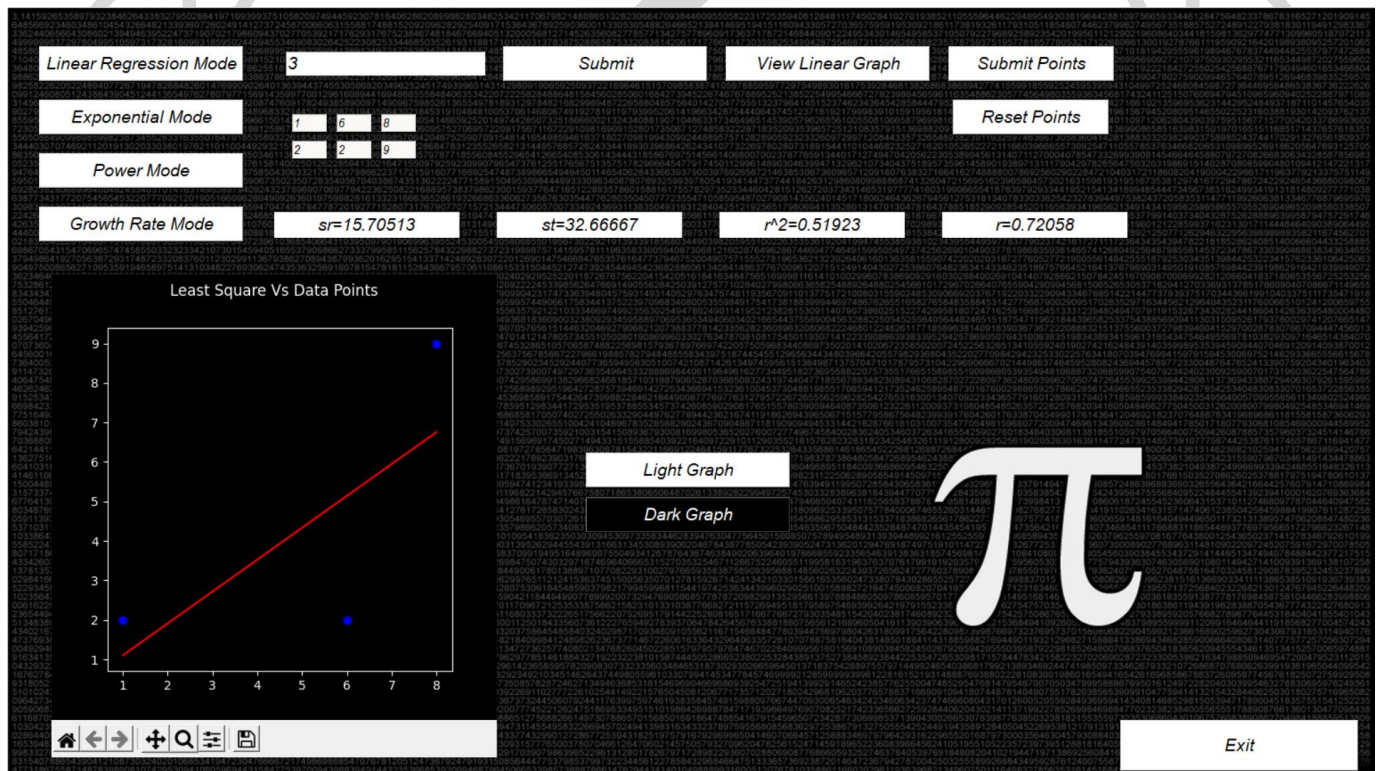
## 1.2. Why $\pi$

The name  $\pi$  gives our app the historical depth and importance of mathematics in our lives as it firstly known by the Egyptians 2000 B.C as approximated constant which define the relation between the diameter and circumference of a circle.

## 1.3. Features

Our application comes with simple user interface, Dark and light modes for the plot and simple transition between modes where on the technical side our app provides 4 modes, the main mode “Linear regression” and 3 linearized models Exponential, Power and Growth-Rate.

## 1.4. How to use



- Choose the mode from the left menu tabs.
- To open the points values entry, enter how many points you will add, then submit.
- After filling each point value of (x, y) as a single column, click on submit points.
- To see the graph data just click on view graph and choose which theme you want from the light and dark options.
- A reset button added if you want to change the number of points where the exit to quit the app.



## 2. CODE

### 2.1. Structure of the code

Our code is OOP based programming to ease the functionality and use the main concepts of OOP as the inheritance which is most important property in our class.

The code basically consists of a class with 8 functions, the initialization function, 4 function for each mode calculations, a function for the each mode buttons, a function for showing the number of entry widgets for the points value and the last one for plotting the graph and change its theme.

### 2.2. Code screenshots

- Imports and class initialization

```
1      import math
2      import tkinter as tk
3      from sympy import symbols, Eq, solve
4      import matplotlib.pyplot as plt
5      from matplotlib.backends.backend_tkagg import *
6      from PIL import *
7      from matplotlib.figure import Figure
8      import tkinter.font as tkfont
9      import os
10
11
12      class math_project():
13          global x_points_global
14          x_points_global=[]
15          global y_points_global
16          y_points_global=[]
17          def __init__(self):
18              self.st=0
19              self.sr=0
20              self.r_square=0
21              self.r=0
22              self.x_fitted_weighted=[]
23              self.y_fitted_weighted=[]
24
```

## ➤ Linear Mode Calculations

```

26 def linear_regression(self):
27     x_points=x_points_global.copy()
28     y_points=y_points_global.copy()
29     y_points_plot=y_points_global.copy()
30     print(x_points,y_points)
31     sum_of_x=sum(x_points)
32     sum_of_y=sum(y_points)
33     sum_of_x_square=0
34     sum_of_x_y=0
35     n=len(x_points)
36     for i in range(len(x_points)):
37         sum_of_x_square=sum_of_x_square+pow(x_points[i],2)
38         sum_of_x_y=sum_of_x_y+x_points[i]*y_points[i]
39     print(f"sum of x={sum_of_x}\nsum of y={sum_of_y}\nsum of x^2={sum_of_x_square}\nsum of xy={sum_of_x_y}")
40     a0, a1 = symbols('a0 a1')
41     eq1 = Eq(n * a0 + sum_of_x * a1 - sum_of_y,0)
42     eq2 = Eq(sum_of_x * a0 + sum_of_x_square * a1 -sum_of_x_y,0)
43     sol_dict = solve((eq1,eq2), (a0, a1))
44     a0=sol_dict[a0]
45     a1=sol_dict[a1]
46     print(f'a0 = {a0}.')
47     print(f'a1 = {a1}.')
48
49     self.sr=0
50     self.st=0
51     mean_of_y=sum_of_y/len(y_points)
52     self.x_fitted_weighted=[]
53     self.y_fitted_weighted=[]
54     for i in range(len(x_points)):
55         self.x_fitted_weighted.append(x_points_global[i])
56         self.y_fitted_weighted.append(a0+a1*x_points[i])
57         self.sr=self.sr+pow((y_points[i]-a0-a1*x_points[i]),2)
58         self.st=self.st+pow((y_points[i]-mean_of_y),2)
59     self.r_square=(self.st-self.sr)/self.st
60     self.r=math.sqrt(self.r_square)
61     self.sr_var.set(f"sr={round(self.sr,5)}")
62     self.st_var.set(f"st={round(self.st,5)}")
63     self.r_square_var.set(f"r^2={round(self.r_square,5)}")
64     self.r_var.set(f"r={round(self.r,5)}")
65     print(f"sr={self.sr}\nstst={self.st}\nr^2={self.r_square}\nr={self.r}")

```

## ➤ Exponential Mode Calculations

```

70 def exponential_model(self):
71     x_points=x_points_global.copy()
72     y_points=y_points_global.copy()
73     y_points_plot=y_points_global.copy()
74     for i in range(len(y_points)):
75         y_points[i]=math.log(y_points[i])
76     sum_of_x=sum(x_points)
77     sum_of_y=sum(y_points)
78     sum_of_x_square=0
79     sum_of_x_y=0
80     n=len(x_points)
81     for i in range(len(x_points)):
82         sum_of_x_square=sum_of_x_square+pow(x_points[i],2)
83         sum_of_x_y=sum_of_x_y+x_points[i]*y_points[i]
84     print(f"sum of x={sum_of_x}\nsum of y={sum_of_y}\nsum of x^2={sum_of_x_square}\nsum of xy={sum_of_x_y}")
85     a0, a1 = symbols('a0 a1')
86     eq1 = Eq(n * a0 + sum_of_x * a1 - sum_of_y,0)
87     eq2 = Eq(sum_of_x * a0 + sum_of_x_square * a1 -sum_of_x_y,0)
88     sol_dict = solve((eq1,eq2), (a0, a1))
89     a0=sol_dict[a0]
90     a1=sol_dict[a1]
91     print(f'a0 = {a0}.')
92     print(f'a1 = {a1}.')
93
94     self.sr=0
95     self.st=0
96     mean_of_y=sum_of_y/len(y_points)
97     a=pow(math.e,a0)
98     b=a1
99     for i in range(len(x_points)):
100         self.sr=self.sr+pow((y_points[i]-a0-a1*x_points[i]),2)
101         self.st=self.st+pow((y_points[i]-mean_of_y),2)
102     i=x_points_global[0]
103     self.x_fitted_weighted=[]
104     self.y_fitted_weighted=[]
105     while i<x_points_global[len(x_points_global)-1]:
106         self.x_fitted_weighted.append(i)
107         self.y_fitted_weighted.append((a*pow(math.e,b*i)))
108         i+=0.01
109     self.r_square=(self.st-self.sr)/self.st
110     self.r=math.sqrt(self.r_square)
111     self.sr_var.set(f"sr={round(self.sr,5)}")
112     self.st_var.set(f"st={round(self.st,5)}")
113     self.r_square_var.set(f"r^2={round(self.r_square,5)}")
114     self.r_var.set(f"r={round(self.r,5)}")

```

## ➤ Power Mode Calculations

```

122 def power_model(self):
123     x_points=x_points_global.copy()
124     y_points=y_points_global.copy()
125     y_points_plot=y_points_global.copy()
126     for i in range(len(y_points)):
127         y_points[i]=math.log10(y_points[i])
128         x_points[i]=math.log10(x_points[i])
129
130     sum_of_x=sum(x_points)
131     sum_of_y=sum(y_points)
132     sum_of_x_square=0
133     sum_of_x_y=0
134     n=len(x_points)
135     for i in range(len(x_points)):
136         sum_of_x_square=sum_of_x_square+pow(x_points[i],2)
137         sum_of_x_y=sum_of_x_y+x_points[i]*y_points[i]
138     print(f"sum of x={sum_of_x}\nsum of y={sum_of_y}\nsum of x^2={sum_of_x_square}\nsum of xy={sum_of_x_y}")
139     a0, a1 = symbols('a0 a1')
140     eq1 = Eq(n * a0 + sum_of_x * a1 - sum_of_y,0)
141     eq2 = Eq(sum_of_x * a0 + sum_of_x_square * a1 -sum_of_x_y,0)
142     sol_dict = solve((eq1,eq2), (a0, a1))
143     a0=sol_dict[a0]
144     a1=sol_dict[a1]
145     print(f'a0 = {pow(10,a0)}')
146     print(f'a1 = {a1}')
147     self.sr=0
148     self.st=0
149     mean_of_y=sum_of_y/len(y_points)
150     a=pow(10,a0)
151     b=a1
152     for i in range(len(x_points)):
153         self.sr=self.sr+pow((y_points[i]-a0-a1*x_points[i]),2)
154         self.st=self.st+pow((y_points[i]-mean_of_y),2)
155
156     i=x_points_global[0]
157     self.x_fitted_weighted=[]
158     self.y_fitted_weighted=[]
159     while i<x_points_global[len(x_points_global)-1]:
160         self.x_fitted_weighted.append(i)
161         self.y_fitted_weighted.append((a*pow(i,b)))
162         i+=0.01
163     self.r_square=(self.st-self.sr)/self.st
164     self.r=math.sqrt(self.r_square)
165     self.sr_var.set(f"sr={round(self.sr,5)}")
166     self.st_var.set(f"st={round(self.st,5)}")

```

## ➤ Growth-Rate Model Calculations

```

175 def growth_model(self):
176     x_points=x_points_global.copy()
177     y_points=y_points_global.copy()
178     for i in range(len(y_points)):
179         y_points[i]=1/(y_points[i])
180         x_points[i]=1/(x_points[i])
181     sum_of_x=sum(x_points)
182     sum_of_y=sum(y_points)
183     sum_of_x_square=0
184     sum_of_x_y=0
185     n=len(x_points)
186     for i in range(len(x_points)):
187         sum_of_x_square=sum_of_x_square+pow(x_points[i],2)
188         sum_of_x_y=sum_of_x_y+x_points[i]*y_points[i]
189     print(f"sum of x={sum_of_x}\nsum of y={sum_of_y}\nsum of x^2={sum_of_x_square}\nsum of xy={sum_of_x_y}")
190     a0, a1 = symbols('a0 a1')
191     eq1 = Eq(n * a0 + sum_of_x * a1 - sum_of_y,0)
192     eq2 = Eq(sum_of_x * a0 + sum_of_x_square * a1 -sum_of_x_y,0)
193     sol_dict = solve((eq1,eq2), (a0, a1))
194     a0=sol_dict[a0]
195     a1=sol_dict[a1]
196     print(f'a0 = {a0}')
197     print(f'a1 = {a1}')
198     self.sr=0
199     self.st=0
200     mean_of_y=sum_of_y/len(y_points)
201     a=1/a0
202     b=a1*a
203     self.x_fitted_weighted=[]
204     self.y_fitted_weighted=[]
205     i=x_points_global[0]
206     while i<x_points_global[len(x_points_global)-1]:
207         self.x_fitted_weighted.append(i)
208         self.y_fitted_weighted.append((a*i)/(b+i))
209         i+=0.01
210     for i in range(len(x_points)):
211         self.sr=self.sr+pow((y_points[i]-a0-a1*x_points[i]),2)
212         self.st=self.st+pow((y_points[i]-mean_of_y),2)
213     self.r_square=(self.st-self.sr)/self.st
214     self.r=math.sqrt(self.r_square)
215     self.sr_var.set(f"sr={round(self.sr,5)}")
216     self.st_var.set(f"st={round(self.st,5)}")
217     self.r_square_var.set(f"r^2={round(self.r_square,5)}")
218     self.r_var.set(f"r={round(self.r,5)}")
219     print(f"sr={self.sr}\nst={self.st}\nr^2={self.r_square}\nr={self.r}")

```



## ➤ Points Entry

```

221
222 def print_points_entry(self, reset=False):
223     myfont=tkfont.Font(family="ubuntu", size=10, weight="normal", slant="italic")
224     myfont_large=tkfont.Font(family="ubuntu", size=14, weight="normal", slant="italic")
225     x_entry=[]
226     y_entry=[]
227     def clear_points():
228         for i in range(len(x_entry)):
229             x_entry[i].destroy()
230             y_entry[i].destroy()
231         points_submit.destroy()
232         points_reset.destroy()
233     def submit_points():
234         x_points_global.clear()
235         y_points_global.clear()
236         for i in range(len(x_entry)):
237             x_points_global.append(float(x_entry[i].get()))
238             y_points_global.append(float(y_entry[i].get()))
239     if reset: clear_points()
240     number_of_points=int(how_many_entry.get())
241     for i in range(number_of_points):
242         x_point=tk.Entry(root, width=5, font=myfont, bg="#FAF9F6")
243         x_point.insert(0, f"x{i+1}")
244         x_point.place(x=320+50*i, y=120)
245         y_point=tk.Entry(root, width=5, font=myfont, bg="#FAF9F6")
246         y_point.insert(0, f"y{i+1}")
247         y_point.place(x=320+50*i, y=150)
248         x_entry.append(x_point)
249         y_entry.append(y_point)
250     points_submit=tk.Button(root, text="Submit Points", command=lambda: [submit_points()], padx=25, font=myfont_large, background="white", fg="#000000")
251     points_submit.grid(row=1, column=5, padx=10, pady=10)
252     points_reset=tk.Button(root, text="Reset Points", command=lambda: [clear_points(), points_submit.destroy(), points_reset.destroy()], padx=25, font=myfont_large, background="white", fg="#000000")
253     points_reset.grid(row=2, column=5, padx=10, pady=10)

```

## ➤ Modes Widgets

```

259
260 def linear_window(self, exponential=False, power=False, growth=False):
261     myfont=tkfont.Font(family="ubuntu", size=14, weight="normal", slant="italic")
262     self.sr_var=tk.StringVar()
263     self.sr_var.set(f"sr=(round(self.sr,5))")
264     sr_label=tk.Label(root, textvariable=self.sr_var, padx=25, font=myfont, width=14, background="white", fg="#000000").place(x=300, y=230)
265     self.st_var=tk.StringVar()
266     self.st_var.set(f"st=(round(self.st,5))")
267     st_label=tk.Label(root, textvariable=self.st_var, padx=25, font=myfont, width=14, background="white", fg="#000000").place(x=550, y=230)
268     self.r_square_var=tk.StringVar()
269     self.r_square_var.set(f"r^2=(round(self.r_square,5))")
270     r_square_label=tk.Label(root, textvariable=self.r_square_var, padx=25, font=myfont, width=14, background="white", fg="#000000").place(x=800, y=230)
271     self.r_var=tk.StringVar()
272     self.r_var.set(f"r=(round(self.r,5))")
273     r_label=tk.Label(root, textvariable=self.r_var, padx=25, font=myfont, width=14, background="white", fg="#000000").place(x=1050, y=230)
274     global how_many_entry
275     how_many_entry=tk.Entry(root, font=myfont, background="white", fg="#000000")
276     how_many_entry.grid(row=1, column=2, padx=10, pady=10)
277     how_many_entry.insert(0, "Enter Number of Points")
278     how_many_button=tk.Button(root, text="Submit", command=lambda: [self.print_points_entry()], font=myfont, background="white", fg="#000000", height=1, width=20).grid(row=1, column=3, padx=10, pady=10)
279     if exponential==True:
280         exp_reg_button=tk.Button(root, text="View Exponential Graph", command=lambda: [self.exponential_model(), self.plot()], font=myfont, background="white", fg="#000000", height=1, width=20).grid(row=1, column=4, padx=10, pady=10)
281     elif power==True:
282         power_reg_button=tk.Button(root, text="View Power Graph", command=lambda: [self.power_model(), self.plot()], font=myfont, background="white", fg="#000000", height=1, width=20).grid(row=1, column=4, padx=10, pady=10)
283     elif growth==True:
284         growth_reg_button=tk.Button(root, text="View Growth Graph", command=lambda: [self.growth_model(), self.plot()], font=myfont, background="white", fg="#000000", height=1, width=20).grid(row=1, column=4, padx=10, pady=10)
285     else:
286         linear_reg_button=tk.Button(root, text="View Linear Graph", command=lambda: [self.linear_regression(), self.plot()], font=myfont, background="white", fg="#000000", height=1, width=20).grid(row=1, column=4, padx=10, pady=10)
289

```

## ➤ Graph and its theme

```

290
291 def plot(self, mode=""):
292     myfont=tkfont.Font(family="ubuntu", size=14, weight="normal", slant="italic")
293     frame=tk.Frame(root)
294     frame.place(x=50, y=300)
295     if mode=="black":
296         plt.style.use('dark_background')
297     elif mode=="light":
298         plt.style.use('default')
299     fig, ax1 = plt.subplots(1, figsize=(5,5))
300     fig.suptitle("Least Square Vs Data Points")
301     ax1.plot(self.x_fitted_weighted, self.y_fitted_weighted, "r")
302     ax1.plot(x_points_global, y_points_global, "b", linestyle="", marker="o")
303     canvas = FigureCanvasTkAgg(fig, master=frame) # A tk.DrawingArea.
304     canvas.draw()
305     canvas.get_tk_widget().pack()
306
307     toolbar = NavigationToolbar2Tk(canvas, frame)
308     toolbar.update()
309     canvas.get_tk_widget().pack()
310     light=tk.Button(master=root, text="Light Graph", font=myfont, background="white", fg="#000000", height=1, width=20, command=lambda: [self.plot(mode="light")]).place(x=650, y=500)
311     dark=tk.Button(master=root, text="Dark Graph", font=myfont, background="#000000", fg="white", height=1, width=20, command=lambda: [self.plot(mode="black")]).place(x=650, y=550)

```

## ➤ Main window Function

```

319
320 project=math_project()
321 root = tk.Tk()
322 def main_window():
323     root.attributes("-fullscreen", True)
324     myfont=tkfont.Font(family="ubuntu",size=14,weight="normal",slant="italic")
325     myfont_xlarge=tkfont.Font(family="koulen",size=18,weight="normal",slant="italic")
326     margin = tk.Label( root, text ="Computational Mathematics",background="white",fg="#000000",font=myfont_xlarge)
327     margin.grid(row=0,column=0)
328     script_dir = os.path.dirname(os.path.abspath(__file__))
329     bg = Image.open(os.path.join(script_dir, 'flower.jpg'))
330     bg=bg.resize((root.winfo_screenwidth(),root.winfo_screenheight()), Image.ANTIALIAS)
331     bg= ImageTk.PhotoImage(bg)
332     # Show image using label
333     label1 = tk.Label( root, image = bg)
334     label1.place(x = -1,y = -1)
335
336
337     regression_mode=tk.Button(text="Linear Regression Mode",font=myfont,command=lambda:[project.linear_window(),project.plot()],background="white",fg="#000000",height=1,width=20)
338     regression_mode.grid(row=1,column=0,padx=10,pady=10)
339     exponential_mode=tk.Button(text="Exponential Mode",font=myfont,command=lambda:[project.linear_window(exponential=True),project.plot()],background="white",fg="#000000",height=1,width=20)
340     exponential_mode.grid(row=2,column=0,padx=10,pady=10)
341     power_mode=tk.Button(text="Power Mode",font=myfont,command=lambda:[project.linear_window(power=True),project.plot()],background="white",fg="#000000",height=1,width=20)
342     power_mode.grid(row=3,column=0,padx=10,pady=10)
343     growth_mode=tk.Button(text="Growth Rate Mode",font=myfont,command=lambda:[project.linear_window(growth=True),project.plot()],background="white",fg="#000000",height=1,width=20)
344     growth_mode.grid(row=4,column=0,padx=10,pady=10)
345
346     exit_button=tk.Button(root,text="Exit",font=myfont,command=root.destroy,padx=20,pady=10,background="white",fg="#000000",height=1,width=20)
347     exit_button.place(x=1250,y=800)
348
349
350
351 root.mainloop()
352 main_window()
353
354
355

```

## 2.3. Test Cases

### ➤ Linear Mode

Linear Regression Mode: 6 Submit View Linear Graph Submit Points

Exponential Mode: 1 2 3 4 5 6 Reset Points

Power Mode: 2.1 3.8 5.3 7.6 10.2 11.9

Growth Rate Mode: sr=0.54476 st=71.54833  $r^2=0.99239$   $r=0.99619$

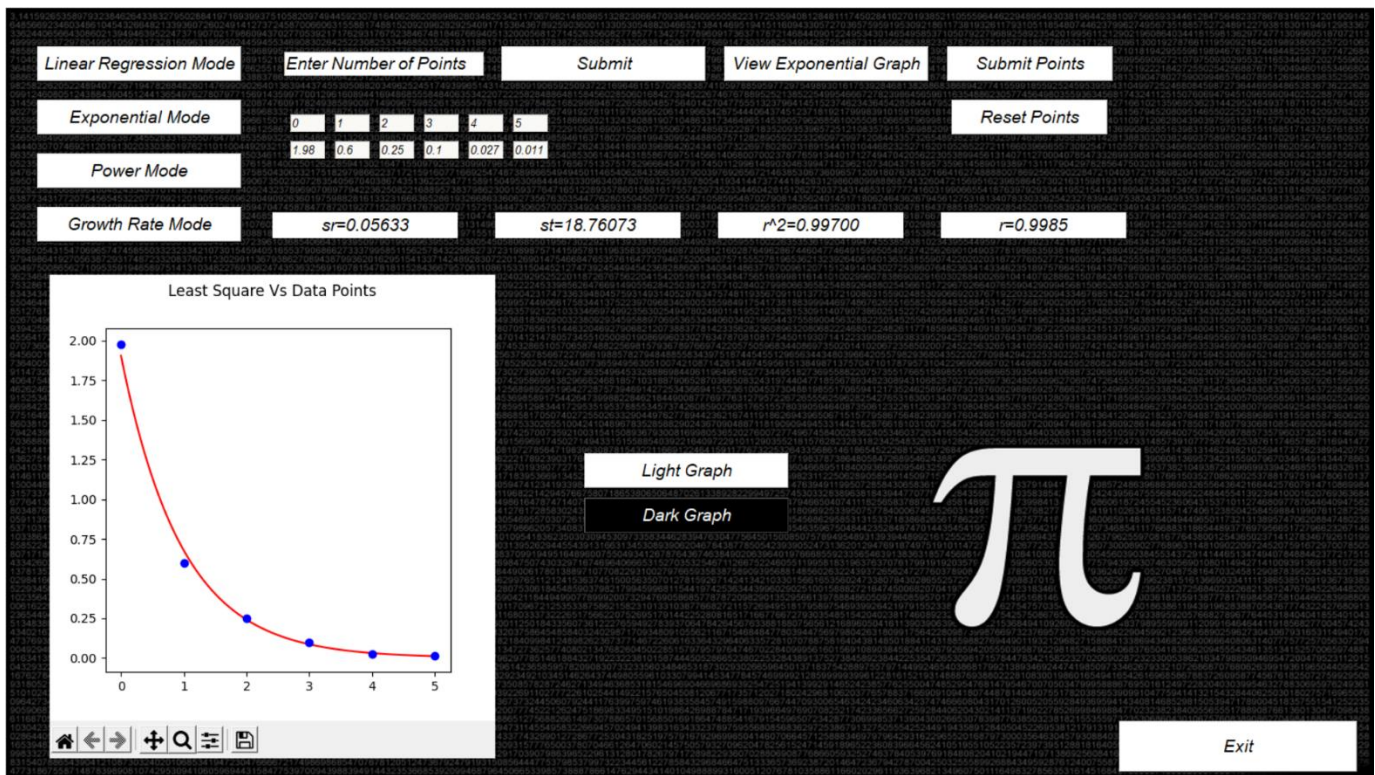
Least Square Vs Data Points

Light Graph Dark Graph

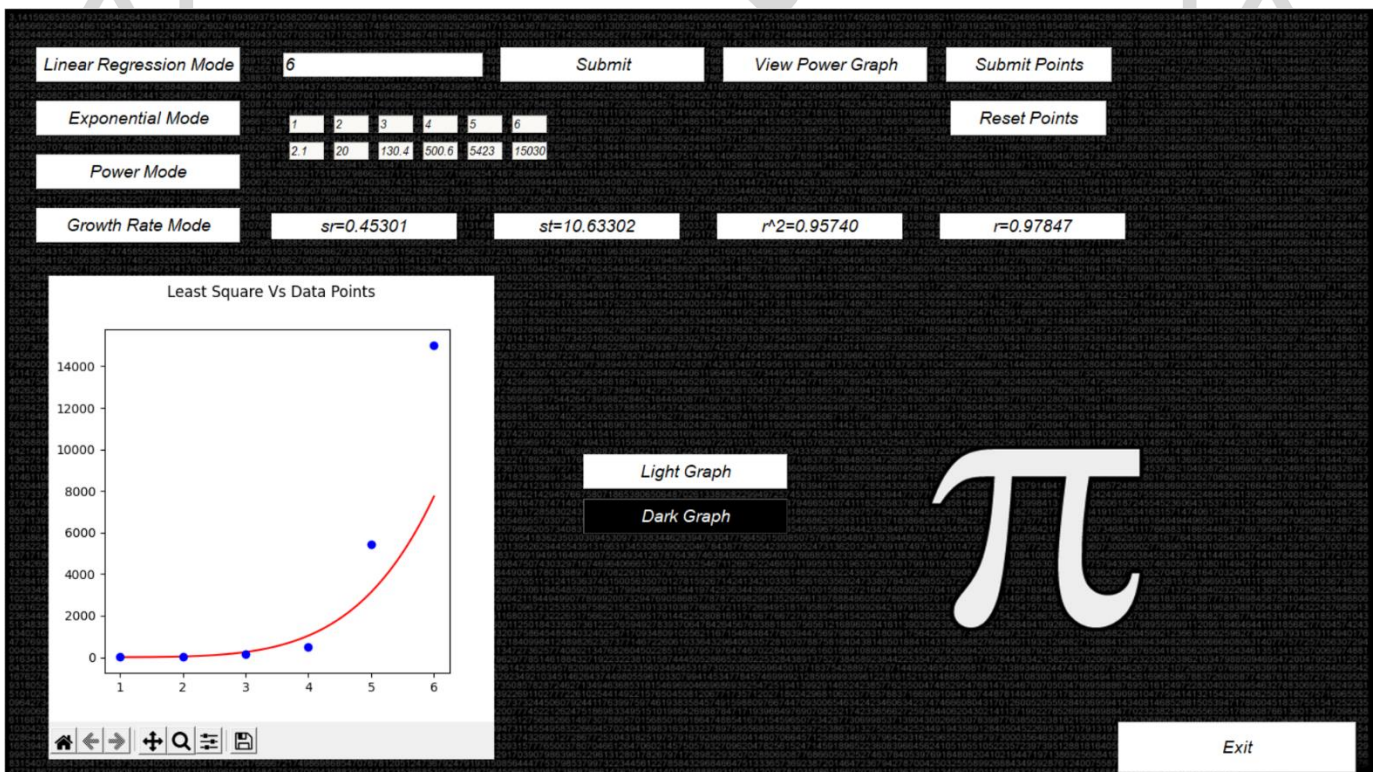
Exit



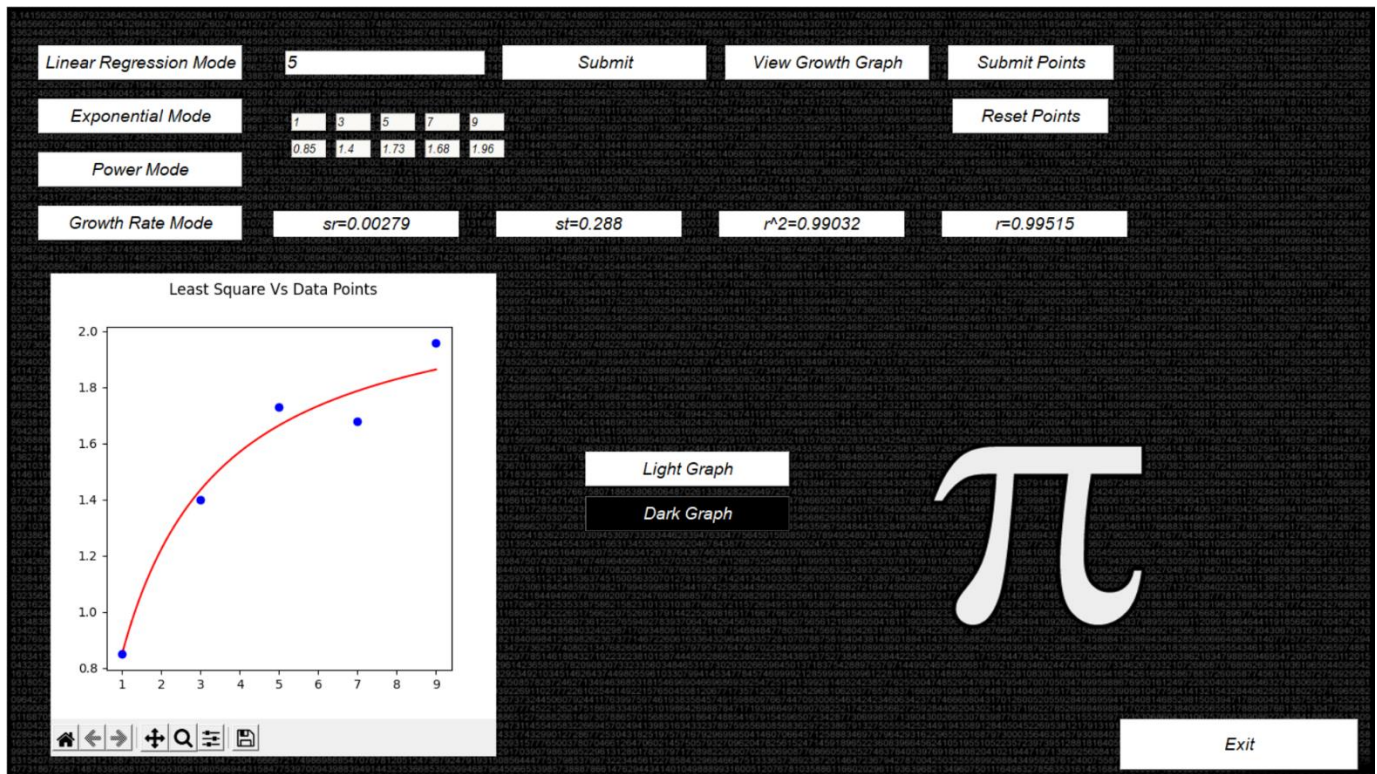
## ➤ Exponential Mode



## ➤ Power Model



## ➤ Growth-Rate Mode



*Pi-App*  
Simply Works



## 3. What`s Next

### 3.1. Our Vision

Our team has an ambition to extend this application from simple tool to be a real worldwide application which helps math students to interact with the concepts easier and save their thoughts and work which needs a good understand for the database concepts.

### 3.2. Features to be added

Our team aims to extend the functionality of our app by adding a comparator for which model is most fit to the given points, also takes the x and y vectors as an excel sheet file and extract their values from it, the gui improvements won't be much except adding a light mode for the whole program, finally a new functions and modes may be added to increase the power of our pi-app as adding a graphical calculator and equations solver.

### 3.3. Mobile App?

Our team has a parallel thought of release a mobile pi-app to ease the access of the students and anyone else using the tool to it, we still do not know which technology will be used for building the app but a great news will be announce soon.

*Pi-App*  
Simply Works





# 4. Newton's Method

## 4.1. Aim of Newton's Method

- Newton's method aims to solve a non-linear system using repeated sequence using a standard formula and initial guess to reach an accepted approximation to the correct answer.

## 4.2. Code Screenshots

- Imports and Differentiation Function.

```
1  from sympy import *
2  import math
3
4  def differentiation(expression):
5      x, y = symbols('x y')
6
7      #print("Before Differentiation : {}".format(gfg_exp))
8
9      # Use sympy.diff() method
10     dif = diff(expression, x)
11     return str(dif)
12
```

- Takes the Arguments from user, taking into account the different scenarios may happen.

```
14  function_input=input("Enter Function Expression:")
15
16  check_initial=(input("Is there initial value for x (y/n):"))
17  while check_initial != "y" and check_initial != "n":
18      check_initial=(input("Is there initial value for x (y/n):"))
19  if check_initial=="y":
20      x=float(input("Enter the Initial X:"))
21  elif check_initial=="n":
22      x=0
23
24  check_iterations=(input("Is there number of iterations(y/n):"))
25  while check_iterations != "y" and check_iterations != "n":
26      check_iterations=(input("Is there number of iterations(y/n):"))
27  if check_iterations=="y":
28      number_of_iterations=int(input("Enter the Number of Iterations:"))
29  elif check_iterations=="n":
30      number_of_iterations=0
31
32  check_error=(input("Is there error value(y/n):"))
33  while check_error != "y" and check_error != "n":
34      check_error=(input("Is there error value(y/n):"))
35  if check_error=="y":
36      error=float(input("Enter the Error Value:"))
37  elif check_error=="n":
38      pass
```

➤ In Case of a number of iterations is given.

```
41     if number_of_iterations!=0:
42         for i in range(number_of_iterations):
43             if i>0:
44                 x=next_x
45                 lower_term=eval(differentiation(function_input))
46                 while lower_term==0:
47                     x+=1
48                     lower_term=eval(differentiation(function_input))
49                 upper_term=eval(function_input)
50                 next_x=x-(upper_term/lower_term)
51                 #print(next_x)
52
53             absolute_error=abs(next_x-x)
54             try:
55                 if absolute_error<error:
56                     break
57                 #print(i)
58             except:
59                 pass
```

➤ Otherwise

```
60     else:
61         absolute_error=1+error
62         i=0
63         while absolute_error>error:
64             if i>0:
65                 x=next_x
66                 lower_term=eval(differentiation(function_input))
67                 while lower_term==0:
68                     x+=1
69                     lower_term=eval(differentiation(function_input))
70                 upper_term=eval(function_input)
71                 next_x=x-(upper_term/lower_term)
72                 try:
73                     absolute_error=abs(next_x-x)
74                     if absolute_error<error:
75                         break
76                 except:
77                     pass
78                 i+=1
79     print(f"Answer Is:{next_x}")
```



### 4.3. Test Case

```
PS C:\Users\himah> & C:/Users/himah/AppData/Local/Programs/Python/Python310/python.exe "d:/Term6/Computation
al Mathmetics/Newton_method.py"
Enter Function Expression:x**2 + log(x)
Is there initial value for x (y/n):y
Enter the Initial X:0.5
Is there number of iterations(y/n):n
Is there error value(y/n):y
Enter the Error Value:0.0001
Answer Is:0.652918640419014
PS C:\Users\himah> █
```

```
al Mathmetics/Newton_method.py"
Enter Function Expression:x**2 - 7
Is there initial value for x (y/n):n
Is there number of iterations(y/n):n
Is there error value(y/n):y
Enter the Error Value:0.001
Answer Is:2.6457513111113693
PS C:\Users\himah> █
```

*Pi-App*  
Simply Works



# 5. Trapezoidal Method

## 5.1. Aim of Trapezoidal Method

- The trapezoidal method aims to calculate a bounded interval integration by the area under the curve of the function where the area is divided through many segments where the curve inside this segment can be linearized.

## 5.2. Code Screenshots

```
1 import math
2 expression=input("Enter the expression:")
3 start=float(input("Enter the start point:"))
4 end=float(input("Enter the end point"))
5 number_of_segments=int(input("Enter number of segments"))
6 unit_segment=(end-start)/number_of_segments
7 areas=[]
8 while start<=end:
9     x=start
10    y_start=eval(expression)
11    x=start+unit_segment
12    areas.append(round(y_start,4))
13    start+=unit_segment
14
15    #print(sum(areas[1:len(areas)-1]))
16    #print(areas)
17    area=(unit_segment/2)*(areas[0]+areas[len(areas)-1]+2*(sum(areas[1:len(areas)-1])))
18    print(round(area,5))
```

## 5.3. Test Cases

```
PS C:\Users\himah> & C:/Users/himah/AppData/Local/Programs/Python/Python310/python.exe "d:/Term6/Computation
al Mathematics/Trapezoidal.py"
Enter the expression:x*math.e**-x
Enter the start point:0
Enter the end point:1
Enter number of segments:5
Answer Is:0.26091
```