Π Desktop Application

Pi-App

Simply Works

TABLE OF CONTENTS

1. **OVERVIEW AND USER MANUAL**
    1) Introduction to our Application
    2) Why π
    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

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.

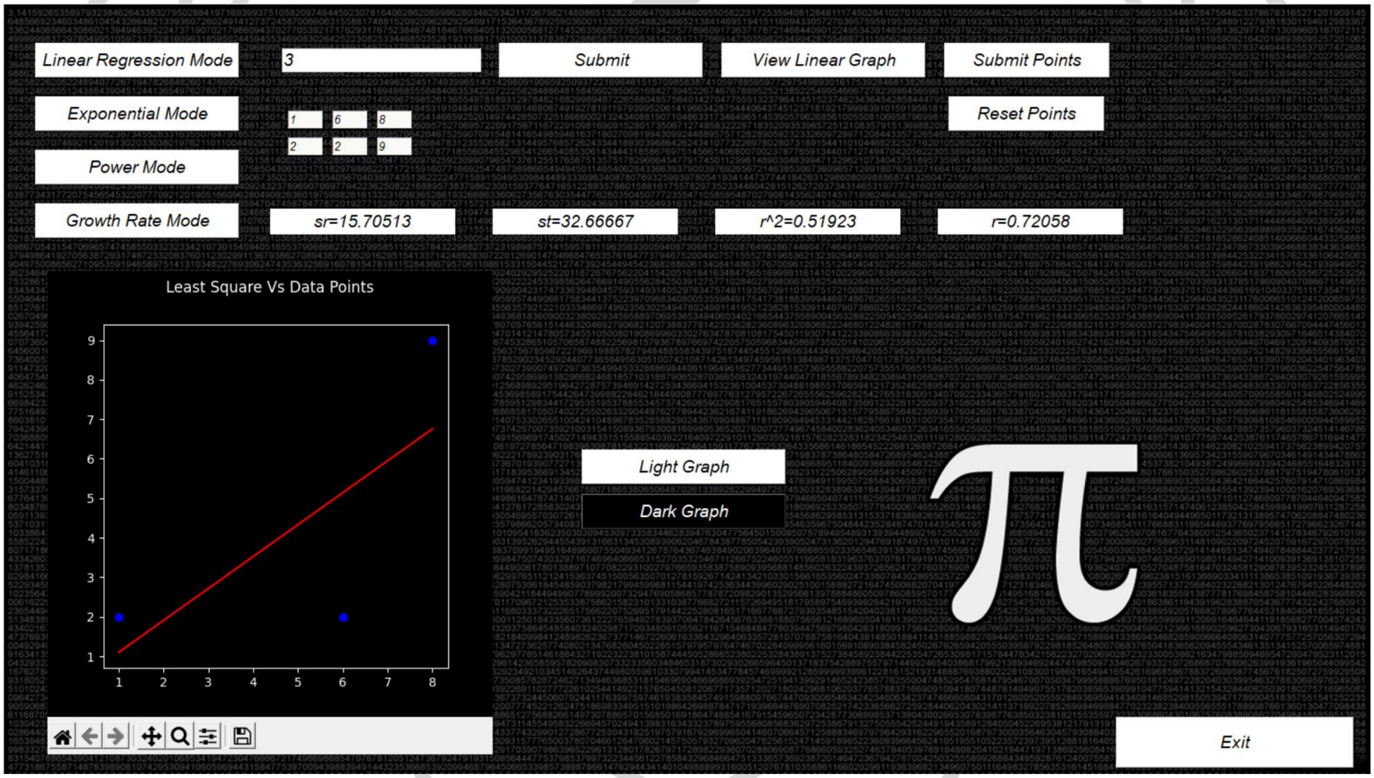
1.2. Why π

The name π 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

```
import math
import tkinter as tk
from sympy import symbols, Eq, solve
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import *
from PIL import *
from matplotlib.figure import Figure
import tkinter.font as tkfont
import os


class math_project():
    global x_points_global
    x_points_global=[]
    global y_points_global
    y_points_global=[]
    def __init__(self):
        self.st=0
        self.sr=0
        self.r_square=0
        self.r=0
        self.x_fitted_weighted=[]
        self.y_fitted_weighted=[]

```

- Linear Mode Calculations

```python
    def linear_regression(self):
        x_points=x_points_global.copy()
        y_points=y_points_global.copy()
        y_points_plot=y_points_global.copy()
        print(x_points,y_points)
        sum_of_x=sum(x_points)
        sum_of_y=sum(y_points)
        sum_of_x_square=0
        sum_of_x_y=0
        n=len(x_points)
        for i in range(len(x_points)):
            sum_of_x_square=sum_of_x_square+pow(x_points[i],2)
            sum_of_x_y=sum_of_x_y+x_points[i]*y_points[i]
        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}")
        a0, a1 = symbols('a0 a1')
        eq1 = Eq(n * a0 + sum_of_x * a1 - sum_of_y,0)
        eq2 = Eq(sum_of_x * a0 + sum_of_x_square * a1 -sum_of_x_y,0)
        sol_dict = solve((eq1,eq2), (a0, a1))
        a0=sol_dict[a0]
        a1=sol_dict[a1]
        print(f'a0 = {a0}')
        print(f'a1 = {a1}')

        self.sr=0
        self.st=0
        mean_of_y=sum_of_y/len(y_points)
        self.x_fitted_weighted=[]
        self.y_fitted_weighted=[]
        for i in range(len(x_points)):
            self.x_fitted_weighted.append(x_points_global[i])
            self.y_fitted_weighted.append(a0+a1*x_points[i])
            self.sr=self.sr+pow((y_points[i]-a0-a1*x_points[i]),2)
            self.st=self.st+pow((y_points[i]-mean_of_y),2)
        self.r_square=(self.st-self.sr)/self.st
        self.r=math.sqrt(self.r_square)
        self.sr_var.set(f"sr={round(self.sr,5)}")
        self.st_var.set(f"st={round(self.st,5)}")
        self.r_square_var.set(f"r^2={round(self.r_square,5)}")
        self.r_var.set(f"r={round(self.r,5)}")
        print(f"sr={self.sr}\nstst={self.st}\nr^2={self.r_square}\nr={self.r}")
```

- Exponential Mode Calculations

```python
    def exponential_model(self):
        x_points=x_points_global.copy()
        y_points=y_points_global.copy()
        y_points_plot=y_points_global.copy()
        for i in range(len(y_points)):
            y_points[i]=math.log(y_points[i])
        sum_of_x=sum(x_points)
        sum_of_y=sum(y_points)
        sum_of_x_square=0
        sum_of_x_y=0
        n=len(x_points)
        for i in range(len(x_points)):
            sum_of_x_square=sum_of_x_square+pow(x_points[i],2)
            sum_of_x_y=sum_of_x_y+x_points[i]*y_points[i]
        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}")
        a0, a1 = symbols('a0 a1')
        eq1 = Eq(n * a0 + sum_of_x * a1 - sum_of_y,0)
        eq2 = Eq(sum_of_x * a0 + sum_of_x_square * a1 -sum_of_x_y,0)
        sol_dict = solve((eq1,eq2), (a0, a1))
        a0=sol_dict[a0]
        a1=sol_dict[a1]
        print(f'a0 = {a0}')
        print(f'a1 = {a1}')

        self.sr=0
        self.st=0
        mean_of_y=sum_of_y/len(y_points)
        a=pow(math.e,a0)
        b=a1
        for i in range(len(x_points)):
            self.sr=self.sr+pow((y_points[i]-a0-a1*x_points[i]),2)
            self.st=self.st+pow((y_points[i]-mean_of_y),2)
        i=x_points_global[0]
        self.x_fitted_weighted=[]
        self.y_fitted_weighted=[]
        while i<x_points_global[len(x_points_global)-1]:
            self.x_fitted_weighted.append(i)
            self.y_fitted_weighted.append((a*pow(math.e,b*i)))
            i+=0.01
        self.r_square=(self.st-self.sr)/self.st
        self.r=math.sqrt(self.r_square)
        self.sr_var.set(f"sr={round(self.sr,5)}")
        self.st_var.set(f"st={round(self.st,5)}")
        self.r_square_var.set(f"r^2={round(self.r_square,5)}")
        self.r_var.set(f"r={round(self.r,5)}")
```

- ➢ Power Mode Calculations

```python
def power_model(self):
    x_points=x_points_global.copy()
    y_points=y_points_global.copy()
    y_points_plot=y_points_global.copy()
    for i in range(len(y_points)):
        y_points[i]=math.log10(y_points[i])
        x_points[i]=math.log10(x_points[i])

    sum_of_x=sum(x_points)
    sum_of_y=sum(y_points)
    sum_of_x_square=0
    sum_of_x_y=0
    n=len(x_points)
    for i in range(len(x_points)):
        sum_of_x_square=sum_of_x_square+pow(x_points[i],2)
        sum_of_x_y=sum_of_x_y+x_points[i]*y_points[i]
    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}")
    a0, a1 = symbols('a0 a1')
    eq1 = Eq(n * a0 + sum_of_x * a1 - sum_of_y,0)
    eq2 = Eq(sum_of_x * a0 + sum_of_x_square * a1 -sum_of_x_y,0)
    sol_dict = solve((eq1,eq2), (a0, a1))
    a0=sol_dict[a0]
    a1=sol_dict[a1]
    print(f'a0 = {pow(10,a0)}')
    print(f'a1 = {a1}')
    self.sr=0
    self.st=0
    mean_of_y=sum_of_y/len(y_points)
    a=pow(10,a0)
    b=a1
    for i in range(len(x_points)):
        self.sr=self.sr+pow((y_points[i]-a0-a1*x_points[i]),2)
        self.st=self.st+pow((y_points[i]-mean_of_y),2)

    i=x_points_global[0]
    self.x_fitted_weighted=[]
    self.y_fitted_weighted=[]
    while i<x_points_global[len(x_points_global)-1]:
        self.x_fitted_weighted.append(i)
        self.y_fitted_weighted.append((a*pow(i,b)))
        i+=0.01
    self.r_square=(self.st-self.sr)/self.st
    self.r=math.sqrt(self.r_square)
    self.sr_var.set(f"sr={round(self.sr,5)}")
    self.st_var.set(f"st={round(self.st,5)}")
```

- ➢ Growth-Rate Model Calculations

```python
def growth_model(self):
    x_points=x_points_global.copy()
    y_points=y_points_global.copy()
    for i in range(len(y_points)):
        y_points[i]=1/(y_points[i])
        x_points[i]=1/(x_points[i])
    sum_of_x=sum(x_points)
    sum_of_y=sum(y_points)
    sum_of_x_square=0
    sum_of_x_y=0
    n=len(x_points)
    for i in range(len(x_points)):
        sum_of_x_square=sum_of_x_square+pow(x_points[i],2)
        sum_of_x_y=sum_of_x_y+x_points[i]*y_points[i]
    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}")
    a0, a1 = symbols('a0 a1')
    eq1 = Eq(n * a0 + sum_of_x * a1 - sum_of_y,0)
    eq2 = Eq(sum_of_x * a0 + sum_of_x_square * a1 -sum_of_x_y,0)
    sol_dict = solve((eq1,eq2), (a0, a1))
    a0=sol_dict[a0]
    a1=sol_dict[a1]
    print(f'a0 = {a0}')
    print(f'a1 = {a1}')
    self.sr=0
    self.st=0
    mean_of_y=sum_of_y/len(y_points)
    a=1/a0
    b=a1*a
    self.x_fitted_weighted=[]
    self.y_fitted_weighted=[]
    i=x_points_global[0]
    while i<x_points_global[len(x_points_global)-1]:
        self.x_fitted_weighted.append(i)
        self.y_fitted_weighted.append((a*i)/(b+i))
        i+=0.01
    for i in range(len(x_points)):
        self.sr=self.sr+pow((y_points[i]-a0-a1*x_points[i]),2)
        self.st=self.st+pow((y_points[i]-mean_of_y),2)
    self.r_square=(self.st-self.sr)/self.st
    self.r=math.sqrt(self.r_square)
    self.sr_var.set(f"sr={round(self.sr,5)}")
    self.st_var.set(f"st={round(self.st,5)}")
    self.r_square_var.set(f"r^2={round(self.r_square,5)}")
    self.r_var.set(f"r={round(self.r,5)}")
    print(f"sr={self.sr}\nst={self.st}\nr^2={self.r_square}\nr={self.r}")
```

- Points Entry

```python
def print_points_entry(self,reset=False):
    myfont=tkfont.Font(family="ubuntu",size=10,weight="normal",slant="italic")
    myfont_large=tkfont.Font(family="ubuntu",size=14,weight="normal",slant="italic")
    x_entry=[]
    y_entry=[]
    def clear_points():
        for i in range(len(x_entry)):
            x_entry[i].destroy()
            y_entry[i].destroy()
        points_submit.destroy()
        points_reset.destroy()
    def submit_points():
        x_points_global.clear()
        y_points_global.clear()
        for i in range(len(x_entry)):
            x_points_global.append(float(x_entry[i].get()))
            y_points_global.append(float(y_entry[i].get()))
    if reset:clear_points()
    number_of_points=int(how_many_entry.get())
    for i in range(number_of_points):
        x_point=tk.Entry(root,width=5,font=myfont,bg="#FAF9F6")
        x_point.insert(0,f"x{i+1}")
        x_point.place(x=320+50*i,y=120)
        y_point=tk.Entry(root,width=5,font=myfont,bg="#FAF9F6")
        y_point.insert(0,f"y{i+1}")
        y_point.place(x=320+50*i,y=150)
        x_entry.append(x_point)
        y_entry.append(y_point)
    points_submit=tk.Button(root,text="Submit Points",command=lambda: [submit_points()],padx=25,font=myfont_large,background="white",fg="#000000")
    points_submit.grid(row=1,column=5,padx=10,pady=10)
    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")
    points_reset.grid(row=2,column=5,padx=10,pady=10)
```

- Modes Widgets

```python
def linear_window(self,exponential=False,power=False,growth=False):
    myfont=tkfont.Font(family="ubuntu",size=14,weight="normal",slant="italic")
    self.sr_var=tk.StringVar()
    self.sr_var.set(f"sr={round(self.sr,5)}")
    sr_label=tk.Label(root,textvariable=self.sr_var,padx=25,font=myfont,width=14,background="white",fg="#000000").place(x=300,y=230)
    self.st_var=tk.StringVar()
    self.st_var.set(f"st={round(self.st,5)}")
    st_label=tk.Label(root,textvariable=self.st_var,padx=25,font=myfont,width=14,background="white",fg="#000000").place(x=550,y=230)
    self.r_square_var=tk.StringVar()
    self.r_square_var.set(f"r^2={round(self.r_square,5)}")
    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)
    self.r_var=tk.StringVar()
    self.r_var.set(f"r={round(self.r,5)}")
    r_label=tk.Label(root,textvariable=self.r_var,padx=25,font=myfont,width=14,background="white",fg="#000000").place(x=1050,y=230)
    global how_many_entry
    how_many_entry=tk.Entry(root,font=myfont,background="white",fg="#000000")
    how_many_entry.grid(row=1,column=2,padx=10,pady=10)
    how_many_entry.insert(0,"Enter Number of Points")
    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)
    if exponential==True:
        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)
    elif power==True:
        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)
    elif growth==True:
        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)
    else:
        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)
```

- Graph and its theme

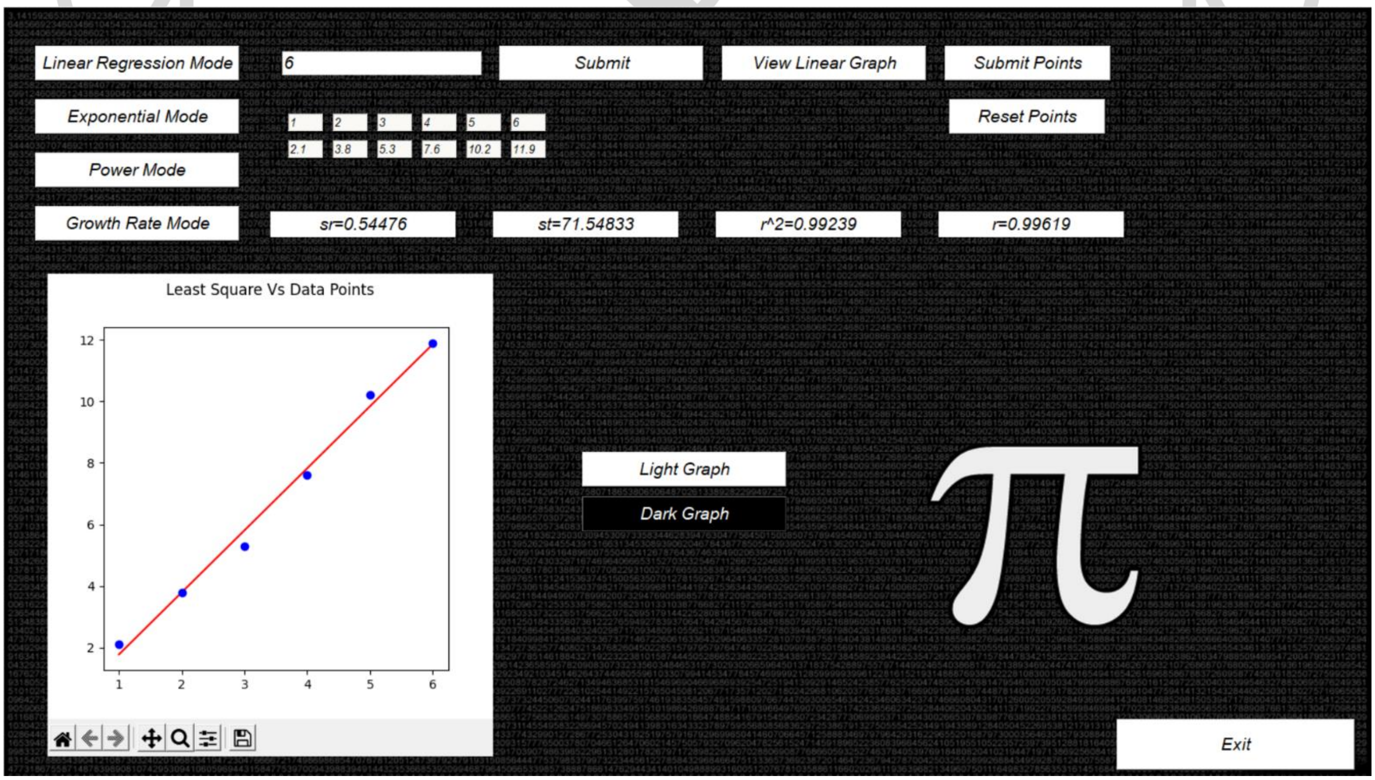
```python
def plot (self,mode=""):
    myfont=tkfont.Font(family="ubuntu",size=14,weight="normal",slant="italic")
    frame=tk.Frame(root)
    frame.place(x=50,y=300)
    if mode=="black":
        plt.style.use('dark_background')
    elif mode=="light":
        plt.style.use('default')
    fig, ax1 = plt.subplots(1,figsize=(5,5))
    fig.suptitle('Least Square Vs Data Points')
    ax1.plot(self.x_fitted_weighted, self.y_fitted_weighted,"r")
    ax1.plot(x_points_global, y_points_global,"b",linestyle="",marker="o")
    canvas = FigureCanvasTkAgg(fig, master=frame)  # A tk.DrawingArea.
    canvas.draw()
    canvas.get_tk_widget().pack()

    toolbar = NavigationToolbar2Tk(canvas, frame)
    toolbar.update()
    canvas.get_tk_widget().pack()
    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)
    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

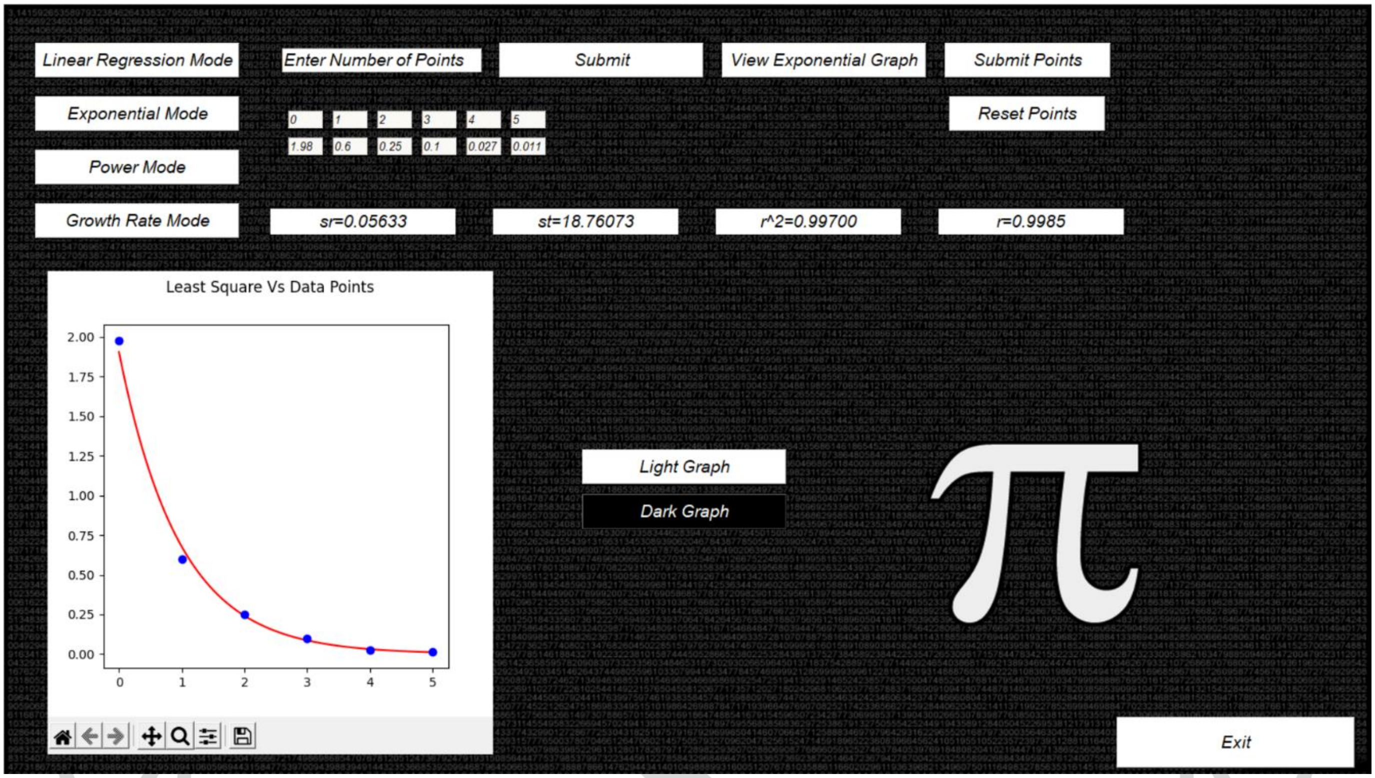
```python
project=math_project()
root = tk.Tk()
def main_window():
    root.attributes('-fullscreen', True)
    myfont=tkfont.Font(family="ubuntu",size=14,weight="normal",slant="italic")
    myfont_xlarge=tkfont.Font(family="koulen",size=18,weight="normal",slant="italic")
    margin = tk.Label( root, text ="Computational Mathematics",background="white",fg="#000000",font=myfont_xlarge)
    margin.grid(row=0,column=0)
    script_dir = os.path.dirname(os.path.abspath(__file__))
    bg = Image.open(os.path.join(script_dir, 'flower.jpg'))
    bg=bg.resize((root.winfo_screenwidth(),root.winfo_screenheight()), Image.ANTIALIAS)
    bg= ImageTk.PhotoImage(bg)
    # Show image using label
    label1 = tk.Label( root, image = bg)
    label1.place(x = -1,y = -1)


    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)
    regression_mode.grid(row=1,column=0,padx=10,pady=10)
    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)
    exponential_mode.grid(row=2,column=0,padx=10,pady=10)
    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)
    power_mode.grid(row=3,column=0,padx=10,pady=10)
    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)
    growth_mode.grid(row=4,column=0,padx=10,pady=10)

    exit_button=tk.Button(root,text="Exit",font=myfont,command=root.destroy,padx=20,pady=10,background="white",fg="#000000",height=1,width=20)
    exit_button.place(x=1250,y=800)



    root.mainloop()
main_window()
```

2.3. Test Cases

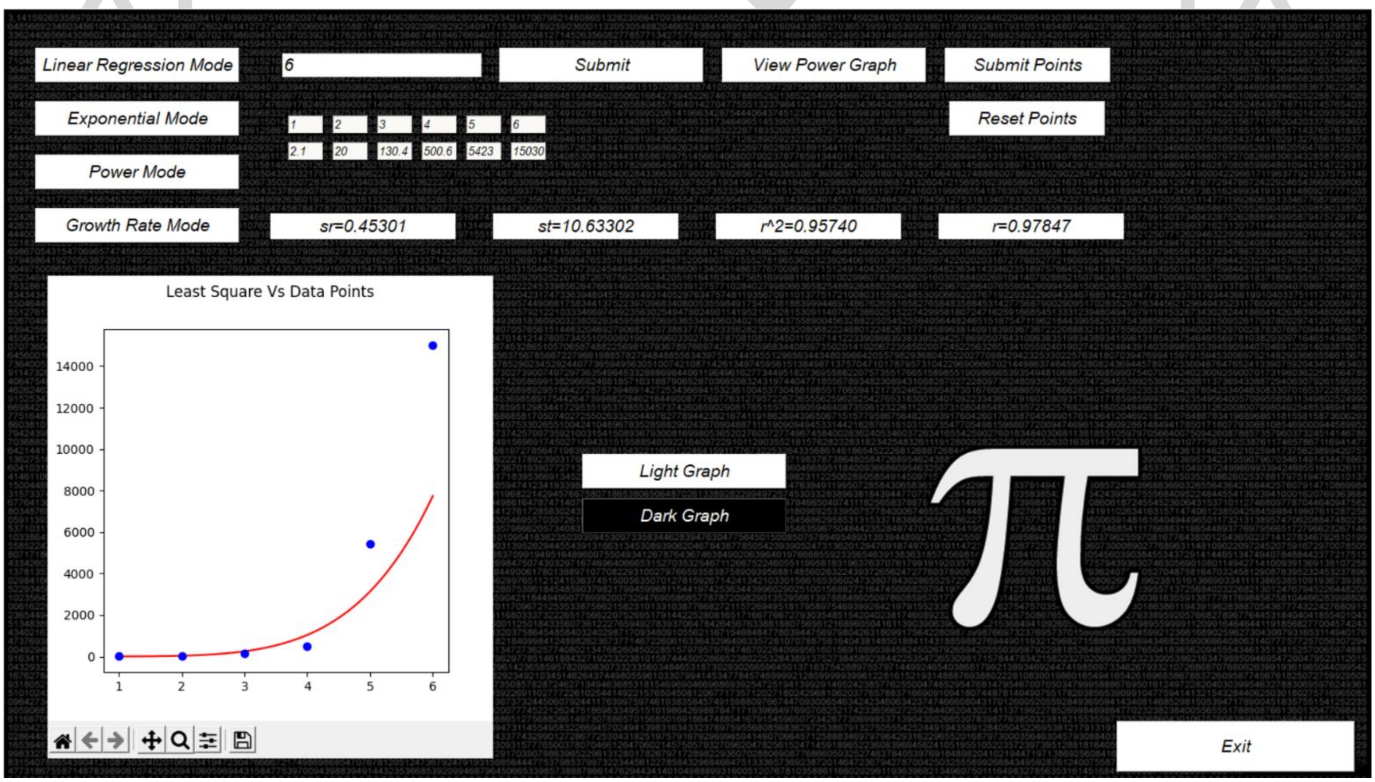
- Linear Mode



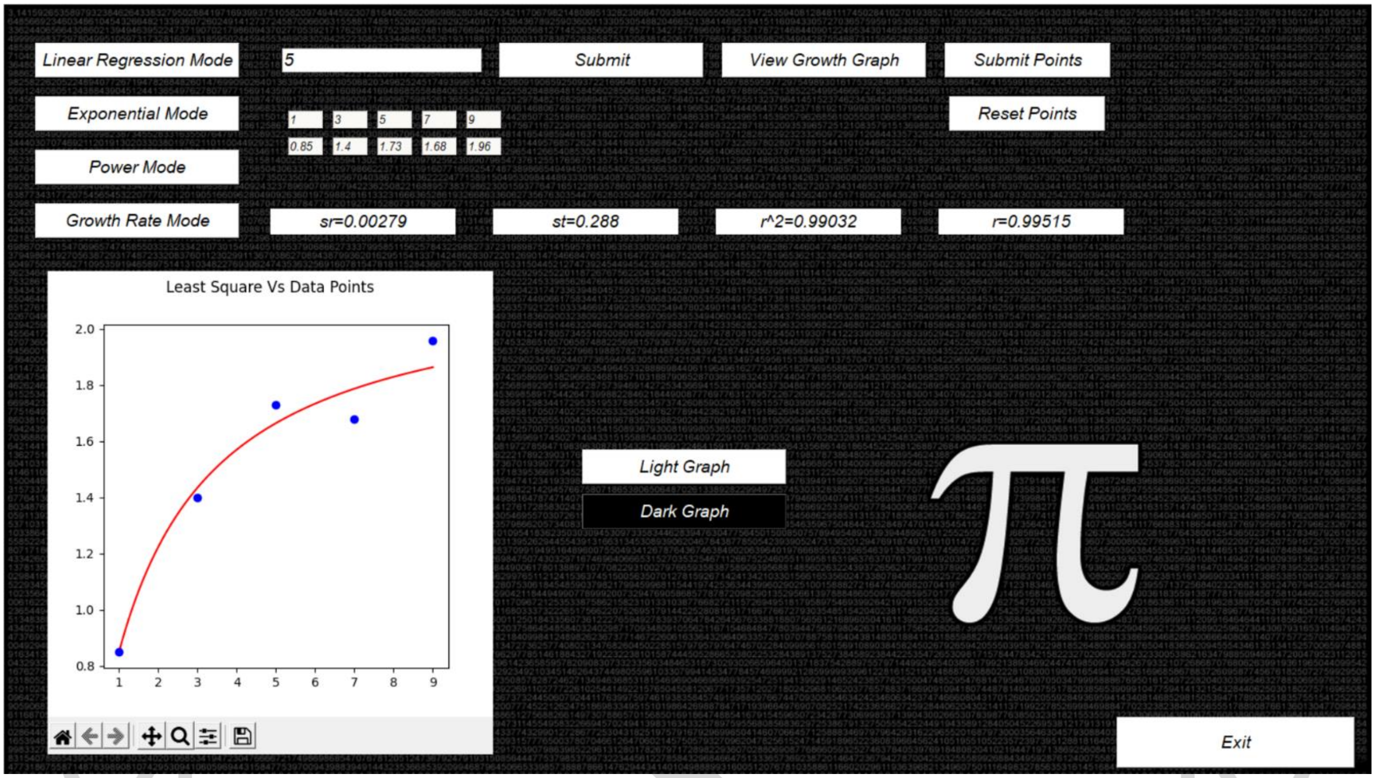
➢ Exponential Mode



➢ Power Model



- Growth-Rate Mode



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.

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.

```
from sympy import *
import math

def differentiation(expression):
    x, y = symbols('x y')

    #print("Before Differentiation : {}".format(gfg_exp))

    # Use sympy.diff() method
    dif = diff(expression, x)
    return str(dif)

```

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

```
function_input=input("Enter Function Expression:")

check_initial=(input("Is there initial value for x (y/n):"))
while check_initial != "y" and check_initial != "n":
    check_initial=(input("Is there initial value for x (y/n):"))
if check_initial=="y":
    x=float(input("Enter the Initial X:"))
elif check_initial=="n":
    x=0

check_iterations=(input("Is there number of iterations(y/n):"))
while check_iterations != "y" and check_iterations != "n":
    check_iterations=(input("Is there number of iterations(y/n):"))
if check_iterations=="y":
    number_of_iterations=int(input("Enter the Number of Iterations:"))
elif check_iterations=="n":
    number_of_iterations=0

check_error=(input("Is there error value(y/n):"))
while check_error != "y" and check_error != "n":
    check_error=(input("Is there error value(y/n):"))
if check_error=="y":
    error=float(input("Enter the Error Value:"))
elif check_error=="n":
    pass
```

➢ In Case of a number of iterations is given.

```python
if number_of_iterations!=0:
    for i in range(number_of_iterations):
        if i>0:
            x=next_x
        lower_term=eval(differentiation(function_input))
        while lower_term==0:
            x+=1
            lower_term=eval(differentiation(function_input))
        upper_term=eval(function_input)
        next_x=x-(upper_term/lower_term)
        #print(next_x)

        absolute_error=abs(next_x-x)
        try:
            if absolute_error<error:
                break
            #print(i)
        except:
            pass
```

➢ Otherwise

```python
else:
    absolute_error=1+error
    i=0
    while absolute_error>error:
        if i>0:
            x=next_x
        lower_term=eval(differentiation(function_input))
        while lower_term==0:
            x+=1
            lower_term=eval(differentiation(function_input))
        upper_term=eval(function_input)
        next_x=x-(upper_term/lower_term)
        try:
            absolute_error=abs(next_x-x)
            if absolute_error<error:
                break
        except:
            pass
        i+=1
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>
```

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

```
import math
expression=input("Enter the expression:")
start=float(input("Enter the start point:"))
end=float(input("Enter the end point"))
number_of_segments=int(input("Enter number of segments"))
unit_segment=(end-start)/number_of_segments
areas=[]
while start<=end:
    x=start
    y_start=eval(expression)
    x=start+unit_segment
    areas.append(round(y_start,4))
    start+=unit_segment

#print(sum(areas[1:len(areas)-1]))
#print(areas)
area=(unit_segment/2)*(areas[0]+areas[len(areas)-1]+2*(sum(areas[1:len(areas)-1])))
print(round(area,5))
```

5.3. Test Cases

```
PS C:\Users\himah> & C:/Users/himah/AppData/Local/Programs/Python/Python310/python.exe "d:/Term6/Computational Mathmetics/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
```