REGRESIÓN CUADRÁTICA.py

C: > Users > puent > OneDrive > Documentos > FRANK > Mau > Pincle_Puente_Francisco_de_Jesus_Programa_6 > REGRESIÓN CUADRÁTICA.py

```python
# Importamos los módulos necesarios
import tkinter as tk  # Para crear la interfaz gráfica
from tkinter import ttk, messagebox  # ttk mejora el aspecto de los widgets, messagebox para mostrar alertas
import numpy as np  # Para cálculos numéricos eficientes
import matplotlib.pyplot as plt  # Para graficar
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg  # Para integrar la gráfica de matplotlib dentro de Tkinter

# Definimos la clase principal de la aplicación
class RegresionCuadraticaApp:
    def __init__(self, root):
        self.root = root  # Ventana principal
        self.root.title("Regresión Cuadrática")  # Título de la ventana
        self.root.geometry("1000x700")  # Tamaño de la ventana
        self.root.config(bg="#1e1e2f")  # Color de fondo oscuro
        self.puntos = []  # Lista donde almacenaremos los puntos (x, y) ingresados

        self.configurar_estilos()  # Llamamos al método que configura los estilos visuales
        self.crear_widgets()  # Llamamos al método que crea los elementos de la interfaz

    # Método para configurar estilos visuales usando ttk
    def configurar_estilos(self):
        estilo = ttk.Style()
        estilo.theme_use("clam")  # Usamos un tema visual amigable
        estilo.configure("TLabel", background="#1e1e2f", foreground="#00ffea", font=("Consolas", 11))
        estilo.configure("TButton", background="#00ffea", foreground="#1e1e2f",
                         font=("Consolas", 10, "bold"), padding=6, relief="flat")
        estilo.map("TButton",  # Estilo cuando el botón está activo (hover)
                   background=[("active", "#00d4ff")],
                   foreground=[("active", "#000000")])

    # Método que crea todos los widgets (entradas, botones, marcos, etiquetas, etc.)
    def crear_widgets(self):
        # Marco para entradas de datos
        frame_entrada = tk.Frame(self.root, bg="#1e1e2f")
        frame_entrada.pack(pady=10)

        # Etiquetas y entradas para X e Y
        ttk.Label(frame_entrada, text="X:").grid(row=0, column=0, padx=5)
        self.entry_x = ttk.Entry(frame_entrada, width=10)
        self.entry_x.grid(row=0, column=1, padx=5)

        ttk.Label(frame_entrada, text="Y:").grid(row=0, column=2, padx=5)
        self.entry_y = ttk.Entry(frame_entrada, width=10)
        self.entry_y.grid(row=0, column=3, padx=5)

        # Botones de acción
        ttk.Button(frame_entrada, text="Agregar Punto", command=self.agregar_punto).grid(row=0, column=4, padx=10)
        ttk.Button(frame_entrada, text="Calcular", command=self.calcular).grid(row=0, column=5, padx=10)
        ttk.Button(frame_entrada, text="Limpiar", command=self.limpiar).grid(row=0, column=6, padx=10)

        # Etiqueta para mostrar la ecuación resultante
        self.label_ecuacion = ttk.Label(self.root, text="", font=("Consolas", 12, "bold"))
        self.label_ecuacion.pack(pady=10)

        # Marcos para mostrar tabla, sistema de ecuaciones y gráfica
        self.frame_tabla = tk.Frame(self.root, bg="#2e2e3f")
        self.frame_tabla.pack(fill="x", padx=10, pady=5)

        self.frame_matriz = tk.Frame(self.root, bg="#1e1e2f")
        self.frame_matriz.pack(pady=10)

        self.frame_grafica = tk.Frame(self.root, bg="#000000", bd=2, relief="sunken")
        self.frame_grafica.pack(expand=True, fill="both", padx=10, pady=10)
```

```
    # Método para agregar un punto a la lista
    def agregar_punto(self):
        try:
            x = float(self.entry_x.get())
            y = float(self.entry_y.get())
            self.puntos.append((x, y))  # Agregamos la tupla (x, y)
            self.entry_x.delete(0, tk.END)
            self.entry_y.delete(0, tk.END)
            self.label_ecuacion.config(text=f"Puntos actuales: {len(self.puntos)}")  # Mostramos cuántos puntos hay
        except ValueError:
            messagebox.showerror("Error", "Por favor ingresa valores numéricos válidos.")

    # Método principal para calcular la regresión cuadrática
    def calcular(self):
        if len(self.puntos) < 3:
            messagebox.showwarning("Advertencia", "Se necesitan al menos 3 puntos.")
            return

        # Convertimos los puntos a arreglos numpy para operaciones vectorizadas
        x = np.array([p[0] for p in self.puntos])
        y = np.array([p[1] for p in self.puntos])
        n = len(x)

        # Calculamos los términos necesarios para el sistema
        x2 = x**2
        x3 = x**3
        x4 = x**4
        xy = x * y
        x2y = x2 * y

```

```
            # Calculamos las sumas necesarias
            sumas = {
                'Σx': np.sum(x), 'Σy': np.sum(y),
                'Σx²': np.sum(x2), 'Σx³': np.sum(x3),
                'Σx⁴': np.sum(x4), 'Σxy': np.sum(xy),
                'Σx²y': np.sum(x2y)
            }

            # Mostramos la tabla de valores
            self.mostrar_tabla(x, y, x2, x3, x4, xy, x2y)

            # Construimos el sistema de ecuaciones A·[c, b, a] = B
            A = np.array([
                [n, sumas['Σx'], sumas['Σx²']],
                [sumas['Σx'], sumas['Σx²'], sumas['Σx³']],
                [sumas['Σx²'], sumas['Σx³'], sumas['Σx⁴']]
            ])
            B = np.array([sumas['Σy'], sumas['Σxy'], sumas['Σx²y']])

            # Resolvemos el sistema con álgebra lineal
            coef = np.linalg.solve(A, B)
            a, b, c = coef[2], coef[1], coef[0]  # Recordamos que en numpy el orden es inverso

            # Mostramos la ecuación en pantalla
            self.label_ecuacion.config(text=f"Ecuación: y = {a:.4f}x² + {b:.4f}x + {c:.4f}")
            self.mostrar_matriz(A, B, coef)
            self.mostrar_grafica(a, b, c)

        # Método para mostrar la tabla con los valores intermedios
        def mostrar_tabla(self, x, y, x2, x3, x4, xy, x2y):
            # Limpiamos el contenido anterior
            for widget in self.frame_tabla.winfo_children():
                widget.destroy()

```

```
        columnas = ["x", "y", "x²", "x³", "x⁴", "xy", "x²y"]
        for col, nombre in enumerate(columnas):
            tk.Label(self.frame_tabla, text=nombre, font=("Consolas", 10, "bold"), fg="#00ffea", bg="#2e2e3f").grid(row=0, column=col)

        # Rellenamos la tabla con valores redondeados
        for i in range(len(x)):
            valores = [x[i], y[i], x2[i], x3[i], x4[i], xy[i], x2y[i]]
            for j, valor in enumerate(valores):
                tk.Label(self.frame_tabla, text=f"{valor:.2f}", font=("Consolas", 10), fg="white", bg="#2e2e3f").grid(row=i+1, column=j)

    # Método para mostrar el sistema de ecuaciones y su solución
    def mostrar_matriz(self, A, B, sol):
        for widget in self.frame_matriz.winfo_children():
            widget.destroy()

        tk.Label(self.frame_matriz, text="Sistema de ecuaciones:", font=("Consolas", 12, "bold"), fg="#00ffea", bg="#1e1e2f").pack(anchor="w")

        for i in range(3):
            fila = f"{A[i][0]:8.2f}a + {A[i][1]:8.2f}b + {A[i][2]:8.2f}c = {B[i]:8.2f}"
            tk.Label(self.frame_matriz, text=fila, font=("Consolas", 11), fg="#ffffff", bg="#1e1e2f").pack(anchor="w")

        sol_texto = f"Solución: a = {sol[2]:.4f}, b = {sol[1]:.4f}, c = {sol[0]:.4f}"
        tk.Label(self.frame_matriz, text=sol_texto, font=("Consolas", 11, "bold"), fg="#00ffea", bg="#1e1e2f").pack(anchor="w")

    # Método para mostrar la gráfica de la regresión
    def mostrar_grafica(self, a, b, c):
        for widget in self.frame_grafica.winfo_children():
            widget.destroy()

        x_vals = np.array([p[0] for p in self.puntos])
        y_vals = np.array([p[1] for p in self.puntos])
        x_line = np.linspace(min(x_vals) - 1, max(x_vals) + 1, 200)
        y_line = a * x_line**2 + b * x_line + c
```

```
            # Creamos la figura con matplotlib
            fig, ax = plt.subplots(figsize=(6, 4), dpi=100)
            ax.scatter(x_vals, y_vals, color='red', label='Puntos')
            ax.plot(x_line, y_line, color='lime', label='Regresión Cuadrática')
            ax.set_title("Regresión Cuadrática", color='white')
            ax.set_xlabel("x")
            ax.set_ylabel("y")
            ax.legend()
            ax.grid(True)
            ax.set_facecolor("#1e1e2f")
            fig.patch.set_facecolor('#1e1e2f')

            # Colores de los ejes
            ax.tick_params(axis='x', colors='white')
            ax.tick_params(axis='y', colors='white')
            ax.title.set_color('white')
            ax.xaxis.label.set_color('white')
            ax.yaxis.label.set_color('white')

            # Mostramos la gráfica dentro del frame de Tkinter
            canvas = FigureCanvasTkAgg(fig, master=self.frame_grafica)
            canvas.draw()
            canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)

        # Método para limpiar toda la interfaz
        def limpiar(self):
            self.puntos.clear()
            self.entry_x.delete(0, tk.END)
            self.entry_y.delete(0, tk.END)
            self.label_ecuacion.config(text="")
            for frame in [self.frame_tabla, self.frame_grafica, self.frame_matriz]:
                for widget in frame.winfo_children():
                    widget.destroy()
```

```
196
197 # Código que se ejecuta al iniciar el programa
198 if __name__ == "__main__":
199     root = tk.Tk()
200     app = RegresionCuadraticaApp(root)
201     root.mainloop()
```