import tkinter as tk  
from tkinter import messagebox  
import numpy as np  
import matplotlib.pyplot as plt  
from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk  
import threading  
import time  
import math  
  
  
# Функция для метода наилучшей пробы с отслеживанием неизменяющегося результата  
def best\_trial\_method(func, x0, step, epsilon, max\_iter, update\_graph, stop\_event, max\_no\_change\_iter):  
 x = np.array(x0)  
 history = [x.copy()]  
 no\_change\_count = 0  
 previous\_result = None  
  
 for i in range(max\_iter):  
 if stop\_event.is\_set():  
 return x, history, f"Минимизация была остановлена на {i + 1}-й итерации."  
  
 # Проверяем точки вокруг текущей с шагом step по каждому измерению  
 candidates = [x + np.eye(len(x0))[j] \* step for j in range(len(x0))] + [x - np.eye(len(x0))[j] \* step for j in range(len(x0))]  
  
 if func is None:  
 print("Ошибка: функция не определена.")  
 return None, None, "Ошибка: функция не определена."  
  
 try:  
 f\_values = [func(\*c) for c in candidates]  
 except Exception as e:  
 print(f"Ошибка при вычислении функции: {e}")  
 return None, None, f"Ошибка: {str(e)}"  
  
 # Выбираем наилучшую точку  
 x\_new = candidates[np.argmin(f\_values)]  
  
 # Условие остановки по малым изменениям  
 if np.linalg.norm(x\_new - x) < epsilon:  
 break  
  
 # Проверяем, изменился ли результат   
 if previous\_result is not None and np.linalg.norm(x\_new - previous\_result) < epsilon:  
 no\_change\_count += 1  
 else:  
 no\_change\_count = 0  
  
 # Если результат не изменяется на протяжении нескольких итераций, завершаем  
 if no\_change\_count >= max\_no\_change\_iter:  
 return x\_new, history, f"Результат не менялся с {i + 1 - no\_change\_count}-й итерации."  
  
 x = x\_new  
 history.append(x.copy())  
 previous\_result = x\_new  
  
 update\_graph(history) # Обновляем график  
  
 time.sleep(0.2) # Пауза для визуализации процесса  
  
 return x, history, f"Достигнуто приближение к минимуму за {i + 1} итераций."  
  
# Основное окно программы  
class MinimizationApp(tk.Tk):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.title("Метод наилучшей пробы")  
 self.geometry("800x700")  
  
 self.func = lambda x: x \*\* 2 # По умолчанию минимизируемая функция  
  
 # Интерфейс ввода данных  
 self.create\_widgets()  
  
 self.stop\_event = threading.Event()  
 self.current\_result = None  
  
 def create\_widgets(self):  
 # Поля для ввода параметров  
 tk.Label(self, text="Начальные точки (например, '0,0' для x и x1):").pack()  
 self.x0\_entry = tk.Entry(self)  
 self.x0\_entry.pack()  
  
 tk.Label(self, text="Шаг (step):").pack()  
 self.step\_entry = tk.Entry(self)  
 self.step\_entry.pack()  
  
 tk.Label(self, text="Точность (epsilon):").pack()  
 self.epsilon\_entry = tk.Entry(self)  
 self.epsilon\_entry.pack()  
  
 tk.Label(self, text="Максимум итераций:").pack()  
 self.max\_iter\_entry = tk.Entry(self)  
 self.max\_iter\_entry.pack()  
  
 tk.Label(self, text="Макс. итераций без изменений:").pack()  
 self.no\_change\_iter\_entry = tk.Entry(self)  
 self.no\_change\_iter\_entry.pack()  
  
 # Поле для ввода функции  
 tk.Label(self, text="Функция для минимизации (например, 'x\*\*2 + 2\*x1\*\*2'):").pack()  
 self.func\_entry = tk.Entry(self)  
 self.func\_entry.pack()  
  
 # Поле для ввода переменных (например, 'x, y')  
 tk.Label(self, text="Переменные (например, 'x, y, x1'):").pack()  
 self.vars\_entry = tk.Entry(self)  
 self.vars\_entry.pack()  
  
 # Кнопки управления  
 tk.Button(self, text="Старт", command=self.start\_minimization).pack(pady=10)  
 tk.Button(self, text="Остановить", command=self.stop\_minimization).pack(pady=10)  
  
 # Площадка для графика  
 self.fig, self.ax = plt.subplots()  
 self.canvas = FigureCanvasTkAgg(self.fig, master=self)  
 self.canvas.get\_tk\_widget().pack(fill=tk.BOTH, expand=1)  
  
 # Добавление панели инструментов для масштабирования и перемещения по графику  
 self.toolbar = NavigationToolbar2Tk(self.canvas, self)  
 self.toolbar.update()  
 self.canvas.get\_tk\_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)  
  
 def start\_minimization(self):  
 # Получаем параметры от пользователя  
 try:  
 # Парсим начальные точки как массив  
 x0 = list(map(float, self.x0\_entry.get().split(',')))  
 step = float(self.step\_entry.get())  
 epsilon = float(self.epsilon\_entry.get())  
 max\_iter = int(self.max\_iter\_entry.get())  
 max\_no\_change\_iter = int(self.no\_change\_iter\_entry.get())  
  
 # Переменные и функция для минимизации  
 variables = self.vars\_entry.get().replace(" ", "").split(',')  
 func\_str = self.func\_entry.get()  
  
 # Создаем контекст с математическими функциями  
 context = {  
 "sin": math.sin, "cos": math.cos, "exp": math.exp, "log": math.log,  
 "sqrt": math.sqrt, "pi": math.pi, "e": math.e, "pow": math.pow  
 }  
  
 # Создаем функцию минимизации с использованием eval  
 self.func = eval(f"lambda {','.join(variables)}: {func\_str}", {"\_\_builtins\_\_": None}, context)  
  
 if self.func is None:  
 raise ValueError("Ошибка: функция не определена.")  
  
 print(f"Функция успешно создана: {func\_str}")  
  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Некорректный ввод или ошибка: {str(e)}")  
 print(f"Ошибка при создании функции: {e}")  
 return  
  
 # Очищаем график  
 self.ax.clear()  
  
 # Сбрасываем событие остановки  
 self.stop\_event.clear()  
  
 # Запускаем метод наилучшей пробы в отдельном потоке  
 self.thread = threading.Thread(  
 target=self.run\_minimization,  
 args=(x0, step, epsilon, max\_iter, max\_no\_change\_iter)  
 )  
 self.thread.start()  
  
 def run\_minimization(self, x0, step, epsilon, max\_iter, max\_no\_change\_iter):  
 # Функция обновления графика  
 def update\_graph(history):  
 self.ax.clear()  
  
 # Проверка на наличие значений в history  
 if len(history) == 0:  
 self.ax.set\_title("Нет данных для отображения.")  
 self.canvas.draw()  
 return  
  
 # Определяем, сколько переменных у нас  
 num\_vars = len(history[0])  
  
 # Устанавливаем границы осей с небольшим отступом  
 if num\_vars == 1:  
 history\_x = [h[0] for h in history] # Значения переменной  
 history\_y = [self.func(x) for x in history\_x] # Вычисляем значение функции для этих значений  
 min\_x, max\_x = min(history\_x), max(history\_x)  
 min\_y, max\_y = min(history\_y), max(history\_y)  
  
 # Обновляем оси  
 self.ax.set\_xlim(min\_x - 5, max\_x + 5)  
 self.ax.set\_ylim(min\_y - 5, max\_y + 5)  
  
 # График для одномерной функции  
 x\_vals = np.linspace(min\_x - 1, max\_x + 1, 400)  
 y\_vals = [self.func(x) for x in x\_vals]  
 self.ax.plot(x\_vals, y\_vals, label='Функция')  
 self.ax.scatter(history\_x, history\_y, c='red', label='Шаги')  
  
 elif num\_vars == 2:  
 history\_x, history\_y = zip(\*history)  
  
 # Устанавливаем границы осей  
 min\_x, max\_x = min(history\_x), max(history\_x)  
 min\_y, max\_y = min(history\_y), max(history\_y)  
 self.ax.set\_xlim(min\_x - 5, max\_x + 5)  
 self.ax.set\_ylim(min\_y - 5, max\_y + 5)  
  
 # Создаем сетку  
 x\_vals = np.linspace(-100, 100, 400)  
 y\_vals = np.linspace(-100, 100, 400)  
 X, Y = np.meshgrid(x\_vals, y\_vals)  
  
 try:  
 # Вычисляем значения функции для сетки точек  
 Z = np.array([[self.func(x, y) for x in x\_vals] for y in y\_vals])  
  
 # Настройка отображения контуров  
 self.ax.contour(X, Y, Z, levels=50, cmap="viridis")  
 self.ax.plot(history\_x, history\_y, 'ro-', label='Минимизация')  
  
 except Exception as e:  
 print(f"Ошибка при построении графика: {e}")  
  
 else:  
 # Для трех и четырех переменных просто выводим текущее состояние  
 history\_x = [h[0] for h in history]  
 history\_y = [self.func(\*h) for h in history] # вычисляем значения функции для текущего состояния  
  
 # Отображаем график для многомерной функции  
 # Здесь можно добавить подходящий график, например, проекции или график значений  
 self.ax.plot(history\_x, history\_y, label='Функция')  
 self.ax.scatter(history\_x, history\_y, c='red', label='Шаги')  
  
 self.ax.legend()  
 self.canvas.draw()  
  
 result, history, message = best\_trial\_method(self.func, x0, step, epsilon, max\_iter, update\_graph,  
 self.stop\_event, max\_no\_change\_iter)  
 # Вычисляем значение функции в найденной точке  
 func\_value\_at\_result = self.func(\*result)  
  
 # Отображаем результат  
 self.current\_result = result  
 messagebox.showinfo("Результат", f"Найденная точка: {result}\n"  
 f"Значение функции в точке: {func\_value\_at\_result}\n"  
 f"{message}")  
  
 def stop\_minimization(self):  
 if not self.stop\_event.is\_set(): # Проверяем, если событие остановки еще не установлено  
 self.stop\_event.set()  
 # if self.current\_result is not None:  
 # messagebox.showinfo("Остановлено", f"Минимизация остановлена. Текущий результат: {self.current\_result}")  
  
  
# Запуск программы  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = MinimizationApp()  
 app.mainloop()