```
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()
```