```
import os
os.environ['TK_SILENCE_DEPRECATION'] = '1'
import tkinter as tk
from tkinter import messagebox
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import threading

# Пункт 1. Создание пользовательского интерфейса
class MinimizationApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Метод деформируемого многогранника")

        # Параметры для минимизации
        self.initial_point = tk.StringVar(value="0, 0")
        self.step_size = tk.DoubleVar(value=0.1)
        self.tol = tk.DoubleVar(value=1e-6)
        self.max_iter = tk.IntVar(value=100)
        self.function_str = tk.StringVar(value="x[0]**2 + x[1]**2")

        # Создание интерфейса для ввода начальных параметров
        tk.Label(root, text="Начальная точка (через запятую):").pack()
        tk.Entry(root, textvariable=self.initial_point).pack()

        tk.Label(root, text="Шаг:").pack()
        tk.Entry(root, textvariable=self.step_size).pack()

        tk.Label(root, text="Точность:").pack()
        tk.Entry(root, textvariable=self.tol).pack()

        tk.Label(root, text="Максимальное количество итераций:").pack()
        tk.Entry(root, textvariable=self.max_iter).pack()

        tk.Label(root, text="Функция для минимизации:").pack()
        tk.Entry(root, textvariable=self.function_str).pack()

        # Кнопка для начала минимизации
        tk.Button(root, text="Запустить минимизацию", command=self.start_minimization).pack()

        # Кнопка для остановки минимизации
        self.stop_flag = False
        tk.Button(root, text="Остановить", command=self.stop_minimization).pack()

        # Область для графического вывода
        self.figure, self.ax = plt.subplots()
        self.canvas = FigureCanvasTkAgg(self.figure, master=root)
        self.canvas.get_tk_widget().pack()

    # Пункт 2. Запуск минимизации
    def start_minimization(self):
        try:
```

```
        # Считывание начальных параметров
        initial_point = np.array([float(x) for x in self.initial_point.get().split(",")])
        step_size = self.step_size.get()
        tol = self.tol.get()
        max_iter = self.max_iter.get()
        function_str = self.function_str.get()

        # Определение целевой функции
        def func(x):
            return eval(function_str)

        self.stop_flag = False
        # Запуск алгоритма в отдельном потоке
        threading.Thread(target=self.run_nelder_mead, args=(func, initial_point, step_size, tol, max_iter)).start()
    except Exception as e:
        messagebox.showerror("Ошибка", f"Ошибка при запуске: {e}")

# Пункт 6. Остановка минимизации
def stop_minimization(self):
    self.stop_flag = True

# Пункт 3. Реализация метода Нелдера-Мида
def run_nelder_mead(self, func, x0, step_size, tol, max_iter):
    # Инициализация начального многогранника
    n = len(x0)
    simplex = np.zeros((n + 1, n))
    simplex[0] = x0
    for i in range(n):
        y = np.array(x0, copy=True)
        y[i] += step_size
        simplex[i + 1] = y

    iter_count = 0
    while iter_count < max_iter and not self.stop_flag:
        simplex = sorted(simplex, key=lambda x: func(x))
        centroid = np.mean(simplex[:-1], axis=0)
        reflection = centroid + (centroid - simplex[-1])

        # Отображение текущего состояния
        self.plot_simplex(simplex, func)
        self.ax.set_title(f"Итерация: {iter_count}")
        self.canvas.draw()

        if np.linalg.norm(simplex[-1] - simplex[0]) < tol:
            break

        if func(reflection) < func(simplex[0]):
            simplex[-1] = reflection
        elif func(reflection) < func(simplex[-2]):
            simplex[-1] = reflection
        else:
            contraction = centroid + 0.5 * (simplex[-1] - centroid)
            if func(contraction) < func(simplex[-1]):
```

```
                simplex[-1] = contraction
            else:
                for i in range(1, n + 1):
                    simplex[i] = simplex[0] + 0.5 * (simplex[i] - simplex[0])

        iter_count += 1

    messagebox.showinfo("Результат", f"Минимизация завершена.\nКоличество итераций: {iter_count}\nТочка минимума: {simplex[0]}")

    # Пункт 4 и 7. Отображение итерационного процесса
    def plot_simplex(self, simplex, func):
        self.ax.clear()
        points = np.array(simplex)
        self.ax.plot(points[:, 0], points[:, 1], 'bo-', label='Симплекс')
        self.ax.scatter(points[0, 0], points[0, 1], color='red', label='Текущая точка минимума')
        self.ax.legend()

# Запуск приложения
if __name__ == "__main__":
    root = tk.Tk()
    app = MinimizationApp(root)
    root.mainloop()
```

```
import os
os.environ['TK_SILENCE_DEPRECATION'] = '1'
import tkinter as tk
from tkinter import messagebox
import numpy as np
import matplotlib.pyplot as plt
```

```
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import threading

# Пункт 1. Создание пользовательского интерфейса
class MinimizationApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Метод деформируемого многогранника")

        # Параметры для минимизации
        self.initial_point = tk.StringVar(value="0, 0")
        self.step_size = tk.DoubleVar(value=0.1)
        self.tol = tk.DoubleVar(value=1e-6)
        self.max_iter = tk.IntVar(value=100)
        self.function_str = tk.StringVar(value="x[0]**2 + x[1]**2")

        # Создание интерфейса для ввода начальных параметров
        tk.Label(root, text="Начальная точка (через запятую):").pack()
        tk.Entry(root, textvariable=self.initial_point).pack()

        tk.Label(root, text="Шаг:").pack()
        tk.Entry(root, textvariable=self.step_size).pack()

        tk.Label(root, text="Точность:").pack()
        tk.Entry(root, textvariable=self.tol).pack()

        tk.Label(root, text="Максимальное количество итераций:").pack()
        tk.Entry(root, textvariable=self.max_iter).pack()

        tk.Label(root, text="Функция для минимизации:").pack()
        tk.Entry(root, textvariable=self.function_str).pack()

        # Кнопка для начала минимизации
        tk.Button(root, text="Запустить минимизацию",
command=self.start_minimization).pack()

        # Кнопка для остановки минимизации
        self.stop_flag = False
        tk.Button(root, text="Остановить", command=self.stop_minimization).pack()

        # Область для графического вывода
        self.figure, self.ax = plt.subplots()
        self.canvas = FigureCanvasTkAgg(self.figure, master=root)
        self.canvas.get_tk_widget().pack()

    # Пункт 2. Запуск минимизации
    def start_minimization(self):
        try:
            # Считывание начальных параметров
            initial_point = np.array([float(x) for x in self.initial_point.get().split(",")])
            step_size = self.step_size.get()
            tol = self.tol.get()
```

```
            max_iter = self.max_iter.get()
            function_str = self.function_str.get()

            # Определение целевой функции
            def func(x):
                return eval(function_str)

            self.stop_flag = False
            # Запуск алгоритма в отдельном потоке
            threading.Thread(target=self.run_nelder_mead, args=(func, initial_point,
step_size, tol, max_iter)).start()
        except Exception as e:
            messagebox.showerror("Ошибка", f"Ошибка при запуске: {e}")

    # Пункт 6. Остановка минимизации
    def stop_minimization(self):
        self.stop_flag = True

    # Пункт 3. Реализация метода Нелдера-Мида
    def run_nelder_mead(self, func, x0, step_size, tol, max_iter):
        # Инициализация начального многогранника
        n = len(x0)
        simplex = np.zeros((n + 1, n))
        simplex[0] = x0
        for i in range(n):
            y = np.array(x0, copy=True)
            y[i] += step_size
            simplex[i + 1] = y

        iter_count = 0
        while iter_count < max_iter and not self.stop_flag:
            simplex = sorted(simplex, key=lambda x: func(x))
            centroid = np.mean(simplex[:-1], axis=0)
            reflection = centroid + (centroid - simplex[-1])

            # Отображение текущего состояния
            self.plot_simplex(simplex, func)
            self.ax.set_title(f"Итерация: {iter_count}")
            self.canvas.draw()

            if np.linalg.norm(simplex[-1] - simplex[0]) < tol:
                break

            if func(reflection) < func(simplex[0]):
                simplex[-1] = reflection
            elif func(reflection) < func(simplex[-2]):
                simplex[-1] = reflection
            else:
                contraction = centroid + 0.5 * (simplex[-1] - centroid)
                if func(contraction) < func(simplex[-1]):
                    simplex[-1] = contraction
                else:
```

```
                for i in range(1, n + 1):
                    simplex[i] = simplex[0] + 0.5 * (simplex[i] - simplex[0])

            iter_count += 1

        messagebox.showinfo("Результат", f"Минимизация завершена.\nКоличество итераций: {iter_count}\nТочка минимума: {simplex[0]}")

    # Пункт 4 и 7. Отображение итерационного процесса
    def plot_simplex(self, simplex, func):
        self.ax.clear()
        points = np.array(simplex)
        self.ax.plot(points[:, 0], points[:, 1], 'bo-', label='Симплекс')
        self.ax.scatter(points[0, 0], points[0, 1], color='red', label='Текущая точка минимума')
        self.ax.legend()

# Запуск приложения
if __name__ == "__main__":
    root = tk.Tk()
    app = MinimizationApp(root)
    root.mainloop()
```