*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()