МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КУБГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчет**

**по лабораторной работе №2 по курсу**

**«МЕТОДЫ ПОИСКОВОЙ ОПТИМИЗАЦИИ»**

Работу выполнили

Студенты 36 группы

Гаранина Л.В.

Агаджанян А.С.

Преподаватель:

Нигодин Е.А.

Краснодар

2025

**Цель работы:** изучить задачу квадратичного программирования и решить одну из них.

**Ход работы**

Для решения задачи квадратичного программирования используются следующие шаги.

**Шаги алгоритма**

1. Преобразовать ограничения исходной задачи к виду g(x) ≤ 0;
2. Составить функцию Лагранжа L(x, λ) = f(x) +;
3. Найти частные производные и составить условие Куна-Такера:
4. Ввести дополнительные переменные , обращающие неравенства в равенства;
5. Ввести первые 2 уравнения искусственные переменные ;
6. Найти целевую функцию F(z) и составить вспомогательную задачу ЛП;
7. Решить задачу симплекс-методом, взяв в качестве начального базиса .

**Реализация симплекс-метода**

Программа написана на языке Python в VS Code. Для вывода графика используется библиотека matplotlib, для создания графического интерфейса – tkinter.

При запуске программы и переходе на вкладку LR2 сразу в правой области отрисовывается график функции , показанный на рисунке 1.

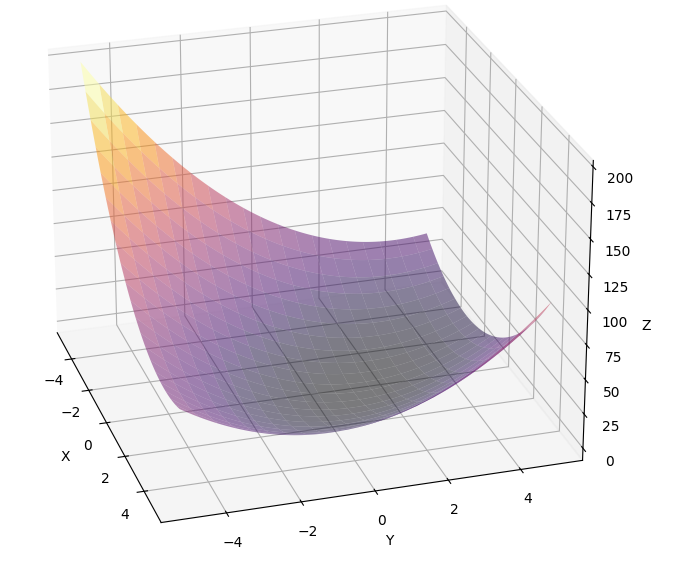


Рисунок 1 – График функции

Рисунок 2 демонстрирует основное меню, расположенное слева и состоящее из поля ввода задержки между интервалами в секундах, поля вывода результатов вычислений и двух кнопок – "Запуск" и "Очистить вывод". При нажатии на "Очистить вывод" полы вывода результатов очищается.

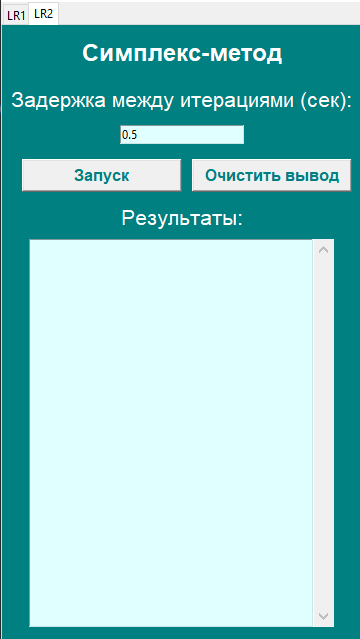


Рисунок 2 – Меню для симплекс-метода

После нажатия на кнопку "Запуск" программа запускает симплекс-метод, выводя результаты в поле вывода, как показано на рисунке 3.

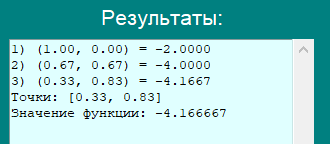


Рисунок 3 – Вывод результатов метода

Параллельно результаты вычислений отображаются точками на графике. Как только метод завершается, результат, последняя точка, отображается красным крестиком. Это показано на рисунке 4.

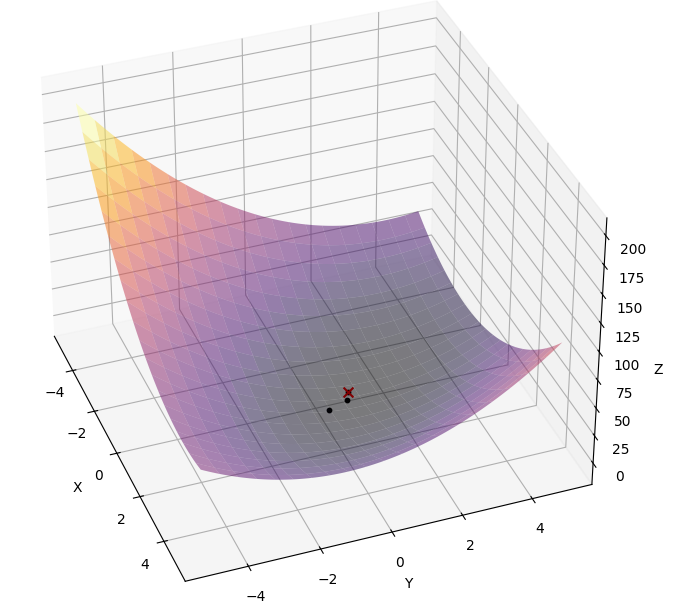


Рисунок 4 – Отображение последовательного решения симплекс-метода через вывод точек на графике

**Вывод:** проделав лабораторную работу, была изучена задача квадратичного программирования и построено ее решение.

**Листинг программы**

**Файл quadratic\_programming.py:**

from functions import \*

def simplex\_method():

c = vect()

a = extract\_and\_modify2()

d = [0, 0, 0]

c1 = c.copy()

def maxVal(a, c):

"""Поиск индексов минимального элемента в последней строке"""

up\_f = max(c[1:])

max\_index = c[1:].index(up\_f) + 1

max\_val = []

for i in range(len(a)):

if a[i][max\_index] != 0:

ratio = a[i][0] / a[i][max\_index]

max\_val.append((ratio, i))

if max\_val:

non\_negative\_values = [(ratio, idx) for ratio, idx in max\_val if ratio >= 0]

if non\_negative\_values:

min\_res, max\_res\_index = min(non\_negative\_values, key=lambda x: x[0])

else:

min\_res, max\_res\_index = None, None

else:

min\_res, max\_res\_index = None, None

return [max\_res\_index, max\_index]

def ch\_ab(a, m, c1):

"""Изменение матрицы и вектора c1 (нормализация строки и обновление)"""

id1, id2 = m

el = a[id1][id2]

if el == 0:

return a, c1

timeC=(-c1[id2])

for i in range(len(a[0])):

a[id1][i] /= el

c1[i] += a[id1][i] \* timeC

for i in range(len(a)):

if i != id1:

time = -a[i][id2]

for j in range(len(a[0])):

a[i][j] += a[id1][j] \* time

return a, c1

iterations = 0

max\_iterations = 10

tt=[7, 8, 6]

h=[]

while True:

m = maxVal(a, c1)

if m[0] is None or m[1] is None or iterations >= max\_iterations:

print("Программа завершена из-за зацикливания или превышения лимита итераций.")

break

id1, id2 = m

tt[id1] = id2

a, c1 = ch\_ab(a, m, c1)

d = [row[0] for row in a]

if all(x <= 0 for x in c1):

break

iterations += 1

result = [0.0] \* 9

for index, value in zip(tt, d):

result[index-1] = value

h.append((round(result[0], 2), round(result[1], 2), round(func1(result[0:2]), 6)))

f=func1(result[0:2])

result = [0.0] \* 9

for index, value in zip(tt, d):

result[index-1] = value

result\_rounded = [round(value, 2) for value in result]

f=func1(result[0:2])

h.append((round(result[0],2), round(result[1], 2), round(func1(result[0:2]), 6)))

return result\_rounded, round(f, 6), h

**Файл function.py:**

def func1(x):

"""Основная функция"""

return 2 \* x[0]\*\*2 + 2 \* x[0] \* x[1] + 2 \* x[1]\*\*2 - 4 \* x[0] - 6 \* x[1]

def dop(x):

"""Ограничение"""

return x[0] + 2\*x[1] - 2

**Файл helper.py:**

from functions import \*

import sympy as sp

import re

x = sp.symbols('x0 x1')

l = sp.symbols('l')

v = [sp.symbols('v0'), sp.symbols('v1')]

z = [sp.symbols('z0'), sp.symbols('z1')]

w = sp.symbols('w')

def lagrange\_function(x, l):

"""Функция Лагранжа"""

return func1(x) + l \* dop(x)

def compute\_derivatives(x, l):

"""Вычисление частных производных из функции Лагранжа"""

L\_func = lagrange\_function(x, l)

dL\_dx0 = sp.diff(L\_func, x[0])

dL\_dx1 = sp.diff(L\_func, x[1])

dL\_dl = sp.diff(L\_func, l)

return dL\_dx0, dL\_dx1, dL\_dl

def modify\_derivatives(dL\_dx0, dL\_dx1, dL\_dl, v, z, w):

"""Модификация частных производных"""

modified\_dL\_dx0 = dL\_dx0 - v[0] + z[0]

modified\_dL\_dx1 = dL\_dx1 - v[1] + z[1]

modified\_dL\_dl = dL\_dl + w

return modified\_dL\_dx0, modified\_dL\_dx1, modified\_dL\_dl

def modify\_and\_sum\_derivatives(modified\_dL\_dx0, modified\_dL\_dx1):

"""Сложение частных производных с изменением знаков и удалением z"""

modified\_dL\_dx0\_with\_sign = -modified\_dL\_dx0.subs({z[0]: 0, z[1]: 0})

modified\_dL\_dx1\_with\_sign = -modified\_dL\_dx1.subs({z[0]: 0, z[1]: 0})

result = modified\_dL\_dx0\_with\_sign + modified\_dL\_dx1\_with\_sign

return result

def calculate\_modified\_sum\_of\_derivatives():

"""Целевая функция для вычисления всех шагов"""

# Вычисление частных производных

dL\_dx0, dL\_dx1, dL\_dl = compute\_derivatives(x, l)

# Модификация частных производных

modified\_dL\_dx0, modified\_dL\_dx1, modified\_dL\_dl0 = modify\_derivatives(dL\_dx0, dL\_dx1, dL\_dl, v, z, w)

# Сложение частных производных с изменением знаков и удалением z

result = modify\_and\_sum\_derivatives(modified\_dL\_dx0, modified\_dL\_dx1)

return result

def reorder\_coefficients(expression):

"""Правильный порядок с изменением знаков, если нужно"""

expression = expression.replace(' ', '')

t=True

terms = re.findall(r'[+-]?[\d]\*\.?[\d]+\\*?[a-zA-Z]+(?:\^?\d\*)?|[+-]?[a-zA-Z]+\d\*|[+-]?[\d]\*\.?[\d]+', expression)

constants = []

x0\_terms = []

x1\_terms = []

l\_terms = []

v0\_terms = []

v1\_terms = []

w\_terms = []

z0\_terms = []

z1\_terms = []

for term in terms:

if re.match(r'^-[\d]+(\.[\d]+)?$', term):

t = False

break

for term in terms:

if re.search(r'[a-zA-Z]', term):

if t:

if term.startswith('+'):

term = '-' + term[1:]

elif term.startswith('-'):

term = '+' + term[1:]

else:

term = '-' + term

else:

if term.startswith('+') or term.startswith('-'):

pass

else:

term = '+' + term

if 'x0' in term:

x0\_terms.append(term)

elif 'x1' in term:

x1\_terms.append(term)

elif 'l' in term:

l\_terms.append(term)

elif 'v0' in term:

v0\_terms.append(term)

elif 'v1' in term:

v1\_terms.append(term)

elif 'w' in term:

w\_terms.append(term)

elif 'z0' in term:

z0\_terms.append(term)

elif 'z1' in term:

z1\_terms.append(term)

else:

if term.startswith('-'):

term = '+' + term[1:]

constants.append(term)

ordered\_coeffs = [0] \* 9

ordered\_coeffs[0] = constants[0] if constants else '+0'

ordered\_coeffs[1] = x0\_terms[0] if x0\_terms else '+0'

ordered\_coeffs[2] = x1\_terms[0] if x1\_terms else '+0'

ordered\_coeffs[3] = l\_terms[0] if l\_terms else '+0'

ordered\_coeffs[4] = v0\_terms[0] if v0\_terms else '+0'

ordered\_coeffs[5] = v1\_terms[0] if v1\_terms else '+0'

ordered\_coeffs[6] = w\_terms[0] if w\_terms else '+0'

ordered\_coeffs[7] = z0\_terms[0] if z0\_terms else '+0'

ordered\_coeffs[8] = z1\_terms[0] if z1\_terms else '+0'

return ordered\_coeffs

def extract\_all\_values(polynomial):

"""Нахождение коэффициентов в полиноме"""

if isinstance(polynomial, list):

polynomial = ''.join(polynomial)

terms = re.findall(r'[+-]?[\d]\*\.?[\d]\*\\*?[a-zA-Z]?\^?\d\*', polynomial)

all\_values = []

for term in terms:

if not term:

continue

if re.search(r'[a-zA-Z]', term):

coef\_match = re.match(r'([+-]?[\d]\*\.?[\d]\*)', term)

if coef\_match:

coef\_value = coef\_match.group(0)

if coef\_value == '' or coef\_value == '+':

all\_values.append(1.0)

elif coef\_value == '-':

all\_values.append(-1.0)

else:

all\_values.append(float(coef\_value))

else:

try:

all\_values.append(float(term))

except ValueError:

print(f"Warning: unable to convert '{term}' to float.")

return all\_values

def vect():

"""Извлечение коэффициентов из целевой функции"""

str\_f = str(calculate\_modified\_sum\_of\_derivatives())

order = reorder\_coefficients(str\_f)

coeff = extract\_all\_values(order)

return coeff

def extract\_and\_modify2():

"""Извлечение коэффициентов из вспомогательной задачи ЛП"""

dL\_dx0, dL\_dx1, dL\_dl = compute\_derivatives(x, l)

modified\_dL\_dx0, modified\_dL\_dx1, modified\_dL\_dl = modify\_derivatives(dL\_dx0, dL\_dx1, dL\_dl, v, z, w)

str\_x0 = str(modified\_dL\_dx0)

str\_x1= str(modified\_dL\_dx1)

str\_l = str(modified\_dL\_dl)

order1 = reorder\_coefficients(str\_x0)

order2 = reorder\_coefficients(str\_x1)

order3 = reorder\_coefficients(str\_l)

coeff1= extract\_all\_values(order1)

coeff2= extract\_all\_values(order2)

coeff3= extract\_all\_values(order3)

matrix = [coeff1, coeff2, coeff3]

return matrix

**Файл main.py:**

import tkinter as tk

from tkinter import scrolledtext, messagebox

from tkinter.ttk import Notebook, Combobox

from matplotlib import pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk

from gradient import \*

from quadratic\_programming import \*

from functions import \*

def plot\_surface(ax, selected\_function):

ax.cla()

x = np.linspace(-5, 5, 100)

y = np.linspace(-5, 5, 100)

x, y = np.meshgrid(x, y)

if selected\_function == "Функция Химмельблау":

z = np.vectorize(lambda x, y: (x\*\*2 + y - 11)\*\*2 + (x + y\*\*2 - 7)\*\*2)(x, y)

elif selected\_function == "Функция Бута":

z = np.vectorize(lambda x, y: (x + 2\*y - 7)\*\*2 + (2\*x + y - 5)\*\*2)(x, y)

elif selected\_function == "Функция Сферы":

z = np.vectorize(lambda x, y: x\*\*2 + y\*\*2)(x, y)

elif selected\_function == "2x^2+2xy+2y^2-4x-6y":

z = np.vectorize(lambda x, y: 2\*x\*\*2 + 2\*x\*y + 2\*y\*\*2 - 4\*x - 6\*y)(x, y)

ax.plot\_surface(x, y, z, rstride=5, cstride=5, alpha=0.5, cmap="inferno")

ax.set\_xlabel('X')

ax.set\_ylabel('Y')

ax.set\_zlabel('Z')

def main():

window = tk.Tk()

width = window.winfo\_screenwidth()

height = window.winfo\_screenheight()

window.geometry("%dx%d" % (width, height))

window.title("Оптимизация")

# Меню

menu\_frame = tk.Frame(window, width=300, bg="#008080")

menu\_frame.pack(side=tk.LEFT, fill=tk.Y)

# Для графика

plot\_frame = tk.Frame(window, bg="#008080")

plot\_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

# График

fig = plt.figure(figsize=(10, 10))

ax = fig.add\_subplot(projection='3d')

canvas = FigureCanvasTkAgg(fig, master=plot\_frame)

canvas.draw()

canvas.get\_tk\_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)

# Панель инструментов для графика

toolbar = NavigationToolbar2Tk(canvas, plot\_frame)

toolbar.update()

canvas.get\_tk\_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)

# Функция для выполнения градиентного спуска

def run\_gradient\_descent():

try:

selected\_function = combobox\_functions.get()

x\_start = float(entry\_x.get())

y\_start = float(entry\_y.get())

step\_size = float(entry\_step.get())

max\_iterations = int(entry\_iterations.get())

delay = float(entry\_delay.get())

history = gradient\_descent(x\_start, y\_start, step\_size, max\_iterations, selected\_function)

plot\_surface(ax, selected\_function)

for i, (x, y, z) in enumerate(history):

ax.scatter(x, y, z, c="black", s=10, marker="o")

if i == len(history) - 1 and len(history) - 1 < max\_iterations:

ax.scatter(x, y, z, c="red", s=50, marker="x")

canvas.draw()

output\_text.insert(tk.END, f"{i + 1}) ({x:.2f}, {y:.2f}) = {z:.4f}\n")

output\_text.yview\_moveto(1)

window.update()

window.after(int(delay \* 1000))

if len(history)-1 == max\_iterations:

messagebox.showwarning("Предупреждение", "Достигнуто максимальное количество итераций, возможно, не найден минимум.")

else: messagebox.showinfo("Уведомление", "Градиентный спуск завершен!")

except ValueError:

messagebox.showerror("Ошибка", "Пожалуйста, введите корректные значения.")

# Функция для выполнения симплекс-метода

def run\_simplex\_method():

try:

output\_text\_simplex.delete("1.0", tk.END)

output\_text.delete("1.0", tk.END)

delay = float(delay\_simplex.get())

points = list(simplex\_method())

if not points:

messagebox.showinfo("Уведомление", "Метод не нашёл решения.")

return

plot\_surface(ax, "2x^2+2xy+2y^2-4x-6y")

# Получаем последний результат

result = points[1]

point = points[0][0:2]

yy = points[2]

for i, (x, y, z) in enumerate(yy):

ax.scatter(x, y, z, c="black", s=10, marker="o")

if i == len(yy) - 1:

ax.scatter(x, y, z, c="red", s=50, marker="x")

canvas.draw()

output\_text\_simplex.insert(tk.END, f"{i + 1}) ({x:.2f}, {y:.2f}) = {z:.4f}\n")

output\_text\_simplex.yview\_moveto(1)

window.update()

window.after(int(delay \* 1000))

output\_text\_simplex.insert(tk.END, f"Точки: {point}\nЗначение функции: {result}\n")

messagebox.showinfo("Уведомление", "Симплекс-метод завершен!")

except Exception as e:

messagebox.showerror("Ошибка", str(e))

def update\_plot\_simplex(event=None):

current\_tab = tab\_control.index(tab\_control.select())

if current\_tab == 0: # LR1 (Градиентный спуск)

plot\_surface(ax, "Функция Химмельблау")

elif current\_tab == 1: # LR2 (Симплекс-метод)

plot\_surface(ax, "2x^2+2xy+2y^2-4x-6y")

canvas.draw()

# Функция для очистки вывода

def clear\_output():

output\_text.delete(1.0, tk.END)

output\_text\_simplex.delete(1.0, tk.END)

# Функция для обновления графика при выборе функции

def update\_plot(event=None):

selected\_function = combobox\_functions.get()

plot\_surface(ax, selected\_function)

canvas.draw()

# Создание вкладок в меню

tab\_control = Notebook(menu\_frame)

# Вкладка для градиентного спуска

tab1 = tk.Frame(tab\_control, bg="#008080")

tab\_control.add(tab1, text="LR1")

label\_title = tk.Label(tab1, text="Градиентный спуск", font=("Arial", 18, "bold"), fg="#FFFFFF", bg="#008080")

label\_title.pack(pady=10)

label\_function = tk.Label(tab1, text="Выберите функцию:", bg="#008080", font=("Arial", 16), fg="#FFFFFF")

label\_function.pack(pady=5)

combobox\_functions = Combobox(tab1, values=["Функция Химмельблау", "Функция Сферы", "Функция Бута"])

combobox\_functions.pack(pady=5)

combobox\_functions.set("Функция Химмельблау")

combobox\_functions.bind("<<ComboboxSelected>>", update\_plot)

label\_x = tk.Label(tab1, text="Начальное значение x:", bg="#008080", font=("Arial", 16), fg="#FFFFFF")

label\_x.pack(pady=5)

entry\_x = tk.Entry(tab1, bg="#E0FFFF")

entry\_x.insert(0, "0")

entry\_x.pack(pady=5)

label\_y = tk.Label(tab1, text="Начальное значение y:", bg="#008080", font=("Arial", 16), fg="#FFFFFF")

label\_y.pack(pady=5)

entry\_y = tk.Entry(tab1, bg="#E0FFFF")

entry\_y.insert(0, "0")

entry\_y.pack(pady=5)

label\_step = tk.Label(tab1, text="Шаг (α):", bg="#008080", font=("Arial", 16), fg="#FFFFFF")

label\_step.pack(pady=5)

entry\_step = tk.Entry(tab1, bg="#E0FFFF")

entry\_step.insert(0, "0.01")

entry\_step.pack(pady=5)

label\_iterations = tk.Label(tab1, text="Макс. кол-во итераций:", bg="#008080", font=("Arial", 16), fg="#FFFFFF")

label\_iterations.pack(pady=5)

entry\_iterations = tk.Entry(tab1, bg="#E0FFFF")

entry\_iterations.insert(0, "1000")

entry\_iterations.pack(pady=5)

label\_delay = tk.Label(tab1, text="Задержка между итерациями (сек):", bg="#008080", font=("Arial", 16), fg="#FFFFFF")

label\_delay.pack(pady=5)

entry\_delay = tk.Entry(tab1, bg="#E0FFFF")

entry\_delay.insert(0, "0.001")

entry\_delay.pack(pady=5)

button\_frame = tk.Frame(tab1, bg="#008080")

button\_frame.pack(side=tk.TOP, padx=10)

# Кнопки для запуска и очистки

run\_button = tk.Button(button\_frame, text="Запуск", font=("Arial", 12, "bold"), fg="#008080", width=15, height=1, command=run\_gradient\_descent)

run\_button.pack(side=tk.LEFT, padx=10)

clear\_button = tk.Button(button\_frame, text="Очистить вывод", font=("Arial", 12, "bold"), fg="#008080", width=15, height=1, command=clear\_output)

clear\_button.pack(side=tk.RIGHT,pady=10)

# Вывод результатов

label\_delay = tk.Label(tab1, text="Результаты:", bg="#008080", font=("Arial", 16), fg="#FFFFFF")

label\_delay.pack(pady=1)

output\_text = scrolledtext.ScrolledText(tab1, width=35, bg="#E0FFFF")

output\_text.pack(fill=tk.Y, pady=5)

tab\_control.bind("<<NotebookTabChanged>>", update\_plot\_simplex)

# Вкладка для симплекс-метода

tab2 = tk.Frame(tab\_control, bg="#008080")

tab\_control.add(tab2, text="LR2")

label\_title\_simplex = tk.Label(tab2, text="Симплекс-метод", font=("Arial", 18, "bold"), fg="#FFFFFF", bg="#008080")

label\_title\_simplex.pack(pady=10)

label\_delay\_simplex = tk.Label(tab2, text="Задержка между итерациями (сек):", bg="#008080", font=("Arial", 16), fg="#FFFFFF")

label\_delay\_simplex.pack(pady=5)

delay\_simplex = tk.Entry(tab2, bg="#E0FFFF")

delay\_simplex.insert(0, "0.5")

delay\_simplex.pack(pady=5)

button\_frame\_simplex = tk.Frame(tab2, bg="#008080")

button\_frame\_simplex.pack(side=tk.TOP, padx=10)

run\_button\_simplex = tk.Button(button\_frame\_simplex, text="Запуск", font=("Arial", 12, "bold"), fg="#008080", width=15, height=1, command=run\_simplex\_method)

run\_button\_simplex.pack(side=tk.LEFT, padx=10)

clear\_button\_simplex = tk.Button(button\_frame\_simplex, text="Очистить вывод", font=("Arial", 12, "bold"), fg="#008080", width=15, height=1, command=clear\_output)

clear\_button\_simplex.pack(side=tk.RIGHT,pady=10)

label\_delay\_simplex = tk.Label(tab2, text="Результаты:", bg="#008080", font=("Arial", 16), fg="#FFFFFF")

label\_delay\_simplex.pack(pady=1)

output\_text\_simplex = scrolledtext.ScrolledText(tab2, width=35, bg="#E0FFFF")

output\_text\_simplex.pack(fill=tk.Y, pady=5)

tab\_control.pack(expand=1, fill="both")

plot\_surface(ax, "Функция Химмельблау")

canvas.draw()

window.mainloop()

if \_\_name\_\_ == "\_\_main\_\_":

main()