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 №3 по курсу**

**«МЕТОДЫ ПОИСКОВОЙ ОПТИМИЗАЦИИ»**

Работу выполнили

Студенты 46 группы

Стасюк К.В.

Баева Д.Н.

Преподаватель:

Нигодин Е.А.

Краснодар 2023

**Цель работы:** разработать генетический алгоритм оптимизации функции Розенброкка.

**Ход работы:**

Генетический алгоритм (ГА) — это метаэвристический метод оптимизации, инспирированный процессами естественного отбора и генетики. Этот алгоритм используется для решения задачи оптимизации и поиска, особенно в ситуациях, где простое и полный переборное решение невозможно или неэффективно. Генетические алгоритмы часто применяются для нахождения приближенных решений в множестве возможных вариантов.

Рассмотрим ключевые элементы генетического алгоритма и его работы:

1. **Популяция**: Генетический алгоритм начинается с создания начальной популяции. Каждый элемент популяции представляет собой потенциальное решение задачи и называется "особью". Эти особи могут быть представлены в виде строк, векторов или других структур данных, в зависимости от конкретной задачи.
2. **Фитнес-функция**: Для оценки качества каждой особи в популяции определяется фитнес-функция. Фитнес-функция принимает особь в качестве входных данных и возвращает численное значение, которое характеризует "подходит ли" эта особь для решения задачи. Цель состоит в том, чтобы максимизировать или минимизировать значение этой функции, в зависимости от задачи оптимизации.
3. **Селекция**: Особи с более высокими значениями фитнес-функции имеют больший шанс выжить и передать свои гены следующему поколению. Это имитирует естественный отбор, где успешные особи имеют больше потомков. Селекция может быть случайной, но вероятность выбора особей зависит от их фитнес-значения.
4. **Скрещивание (кроссовер)**:Выбранные особи скрещиваются, и их гены комбинируются, чтобы создать потомство. Как именно это происходит, зависит от метода скрещивания, который может варьироваться от простой одноточечной кроссовер до более сложных методов. Цель скрещивания - внести разнообразие в популяцию и, возможно, сочетать положительные черты разных особей.
5. **Мутация**: Некоторые гены в потомстве могут случайным образом мутировать. Мутация вносит случайные изменения в гены особей и помогает сохранить разнообразие в популяции.
6. **Замена поколения**: Новое поколение особей формируется путем комбинирования селекции, скрещивания и мутации. Часть старой популяции заменяется новой на основе их фитнес-значений.
7. **Критерий остановки**: Генетический алгоритм продолжает создавать новые поколения и улучшать популяцию до тех пор, пока не выполнится определенный критерий остановки. Критерии могут включать ограничение по количеству поколений, достижение желаемого значения фитнес-функции или истечение времени.

Теперь рассмотрим подробнее реализацию генетического алгоритма на функции Розенброка:

1. **Инициализация популяции**: В начале работы алгоритма создается начальная популяция из случайных особей. Каждая особь представляется двумя генами (x, y), и ей назначается значение фитнес-функции (z), которое рассчитывается на основе значений x и y, используя вашу оптимизируемую функцию. Наша цель - найти особь (x, y), для которой значение z минимальное.
2. **Эволюция популяции:**

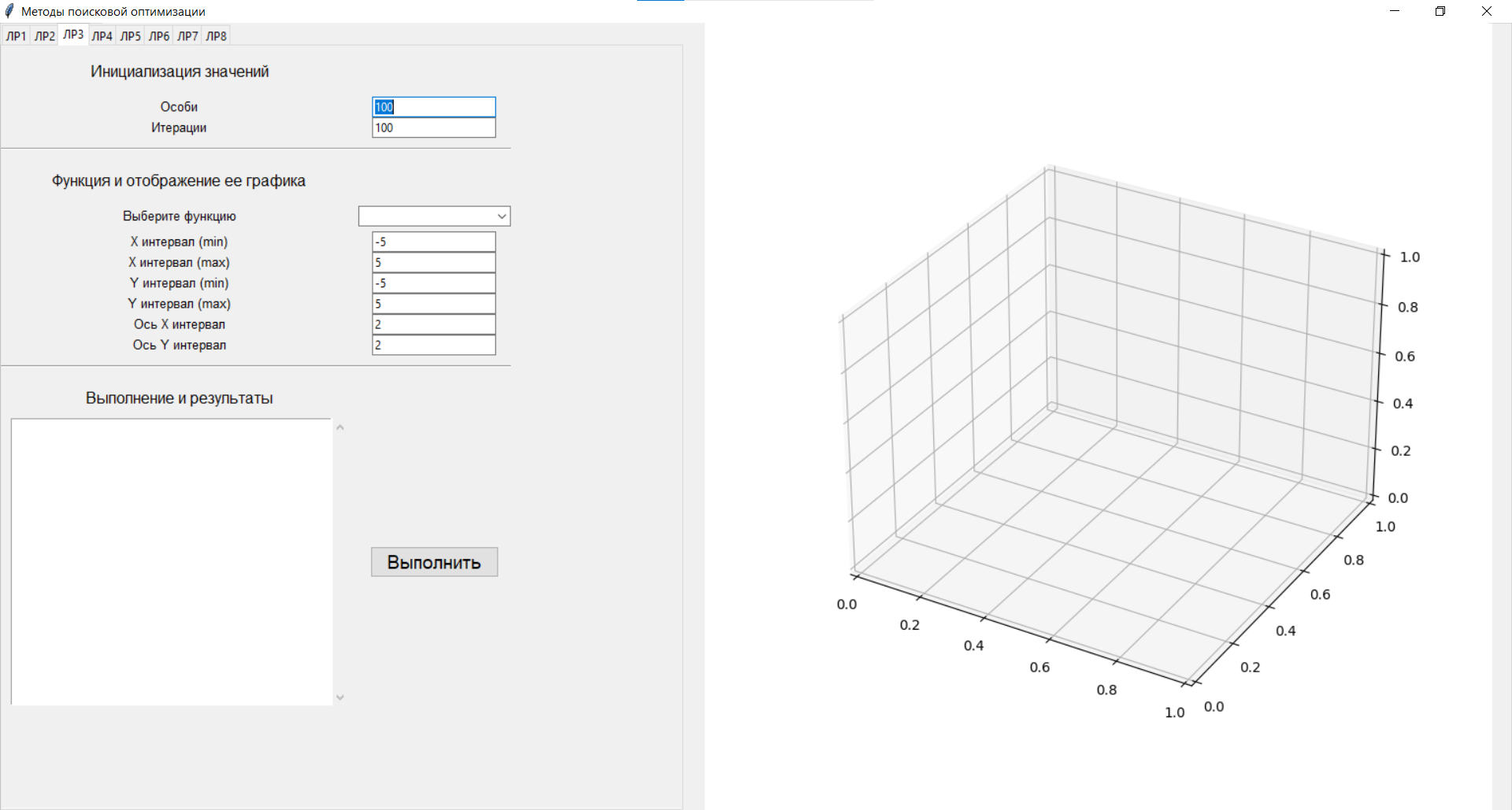
* *Селекция*: Особи в текущей популяции ранжируются в порядке их значений фитнес-функции. Лучшие особи (те, у которых z ближе к оптимуму) имеют большие шансы быть выбранными как родители. Количество особей, которые будут выбраны в качестве родителей, определяется коэффициентом выживаемости (survive\_cof).
* *Репродукция* (скрещивание): Выбранные родители принимают участие в создании потомства. Для каждой новой особи, x и y наследуются от родителей, и значение z пересчитывается на основе новых x и y. Вероятность того, что x или y будут заменены значениями другого родителя, зависит от случайного числа. Это процесс повторяется для нескольких новых особей.
* *Мутация*: Каждая особь в популяции имеет шанс быть подверженной мутации. Это означает, что x и/или y могут быть незначительно изменены случайным образом. Вероятность мутации определяется параметром mut\_chance.

1. **Итерации**: Эти шаги (селекция, скрещивание и мутация) повторяются в течение заданного количества поколений (generations). В каждом поколении лучшие особи сохраняются, а менее успешные могут быть заменены новыми особями.
2. **Завершение и выбор лучшей особи**: После завершения всех итераций, алгоритм возвращает особь с наилучшим значением z, то есть особь с минимальным значением z.

**Особенности реализации генетического алгоритма.**

Для создания программы использовался язык программирования Python 3.10 и среда разработки PyCharm Professional 2023.2. Для графической визуализации были подключены графические фреймворки Tkinter и Matplotlib.

Интерфейс программы имеет вид:

Рисунок 1 – Главное окно программы.

В данном окне можно выбрать необходимую вкладку соответствующей лабораторной работы, в данном случае «ЛР3», ввести количество особей для операций и количество итераций.

В поле «Выполнение и результаты» динамически выводятся результаты работы алгоритма в виде шагов, представленных координатами и значениями оптимизируемой функции в этих координатах. На рисунке 2 показано поле «Выполнение и результаты» до запуска программы.

Изображение выглядит как текст, снимок экрана, Прямоугольник

Автоматически созданное описание

Рисунок 2 – Выполнение и результаты.

Также, в интерфейсе программы есть панель управления «Функция и отображение её графика», в котором можно выбрать необходимую функцию и задать её параметры:

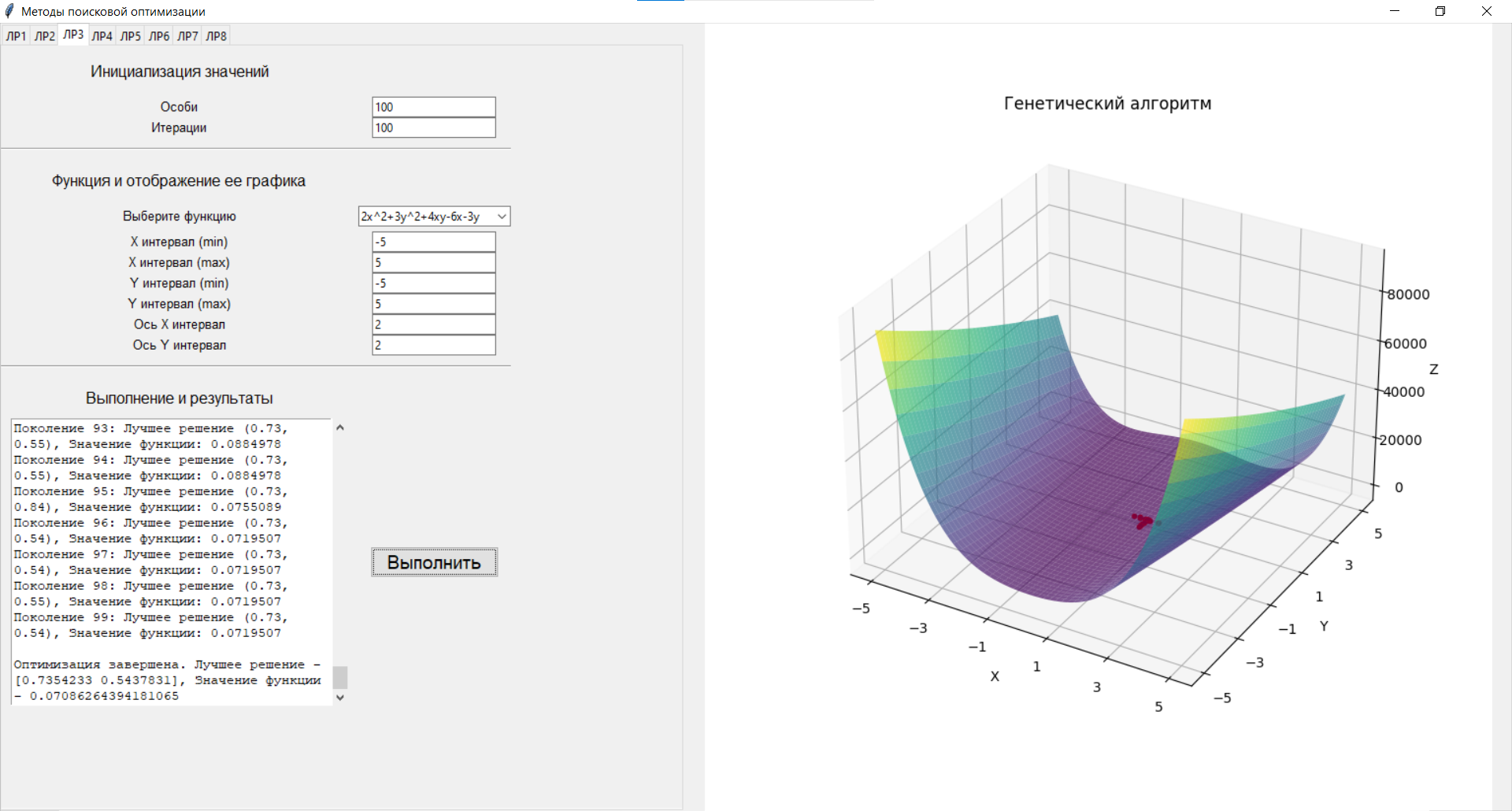
1. X интервал (интервал на котором строится и минимизируется функция);
2. Y интервал (интервал на котором строится и минимизируется функция);
3. Ось X интервал (размер отображаемой сетки по оси X);
4. Ось Y интервал (размер отображаемой сетки по оси Y).

На рисунке 3 показана панель «Функция и отображение её графика»

Изображение выглядит как текст, Шрифт, линия, число

Автоматически созданное описаниеРисунок 3 – Функция и отображение её графика.

Для запуска алгоритма необходимо нажать кнопку «Выполнить», при этом поле «Выполнение и результаты» начнёт динамически заполняться точками, а на отображаемой функции можно наглядно увидеть функционирование алгоритма - искомые точки, которые также отображаются динамически как показано на рисунке 4. Результирующая точка выделена на графике крестом. Значения результирующей функции имеют разный окрас в зависимости от положения: самые высокие точки показаны ярко-жёлтым цветом, а самые низкие – тёмно-синим.

Рисунок 4 – Результат работы программы.

**Вывод:** в ходе работы были изучен, а затем разработан генетический алгоритм оптимизации функции Розенброкка.

**Листинг программы**

Файл LR3.py

import tkinter as tk

from tkinter import ttk

import numpy as np

import time

from tkinter import scrolledtext

from scipy.optimize import minimize

import numpy as np

def GeneticAlgorithm(frame,root,ax,canvas):

def target\_function(x, y):

return (1-x)\*\*2+100\*(y-x\*\*2)\*\*2

# Функция Розенброка для оптимизации

def rosenbrock\_function(x, y):

return (1 - x) \*\* 2 + 100 \* (y - x \*\* 2) \*\* 2

# Оператор селекции (выбор лучших особей)

def selection(population, fitness\_scores):

# Выбираем двух наилучших особей

best\_indices = np.argsort(fitness\_scores)[:2]

return [population[i] for i in best\_indices]

# Оператор кроссовера (одноточечный кроссовер)

def crossover(parent1, parent2):

crossover\_point = np.random.randint(1, len(parent1))

child = np.hstack((parent1[:crossover\_point], parent2[crossover\_point:]))

return child

# Оператор мутации

def mutate(individual, mutation\_rate):

mutation\_indices = np.random.rand(len(individual)) < mutation\_rate

individual[mutation\_indices] += np.random.uniform(-0.5, 0.5)

return individual

def run\_optimization():

# Генерация сетки для графика целевой функции

x\_range = np.linspace(x\_interval\_min.get(), x\_interval\_max.get(), 100)

y\_range = np.linspace(y\_interval\_min.get(), y\_interval\_max.get(), 100)

X, Y = np.meshgrid(x\_range, y\_range)

Z = target\_function(X, Y)

population\_size=int(x\_var.get())

num\_generations=int(y\_var.get())

population = np.random.uniform(low=-5, high=5, size=(population\_size, 2))

results = []

results\_text.config(state=tk.NORMAL)

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

for generation in range(num\_generations):

# Расчет значений функции для текущей популяции

fitness\_scores = np.array([rosenbrock\_function(x, y) for x, y in population])

# Выбор лучших особей

selected\_individuals = selection(population, fitness\_scores)

# Оператор кроссовера и мутации

children = []

for i in range(0, population\_size, 2):

child1 = crossover(selected\_individuals[0], selected\_individuals[1])

child2 = crossover(selected\_individuals[1], selected\_individuals[0])

child1 = mutate(child1, mutation\_rate=0.1) # Пример вероятности мутации

child2 = mutate(child2, mutation\_rate=0.1)

children.extend([child1, child2])

ax.cla()

# Построение поверхности графика целевой функции

ax.plot\_surface(X, Y, Z, cmap='viridis', alpha=0.7)

ax.set\_xlabel('X')

ax.set\_ylabel('Y')

ax.set\_zlabel('Z')

ax.set\_xticks(np.arange(x\_interval\_min.get(), x\_interval\_max.get() + 1, x\_axis\_interval.get()))

ax.set\_yticks(np.arange(y\_interval\_min.get(), y\_interval\_max.get() + 1, y\_axis\_interval.get()))

ax.set\_title("Генетический алгоритм")

for i in range(len(fitness\_scores)):

best\_individual = population[i]

ax.scatter(best\_individual[0], best\_individual[1], fitness\_scores[i], color='red',

s=10)

# Обновление популяции

population = np.array(children)

# Нахождение лучшей особи на текущей итерации

best\_fitness = np.min(fitness\_scores)

best\_individual = population[np.argmin(fitness\_scores)]

# Вывод лучшего решения на текущей итерации

print(f"Поколение {generation}: Лучшее решение - {best\_individual}, Значение функции - {best\_fitness}")

results.append((best\_individual[0], best\_individual[1], generation, best\_fitness))

results\_text.insert(tk.END,

f"Поколение {generation}: Лучшее решение ({best\_individual[0]:.2f}, {best\_individual[1]:.2f}), Значение функции: {best\_fitness:.7f}\n")

results\_text.yview\_moveto(1)

canvas.draw()

root.update()

# Нахождение лучшего решения после всех итераций

final\_fitness\_scores = np.array([rosenbrock\_function(x, y) for x, y in population])

best\_index = np.argmin(final\_fitness\_scores)

best\_solution = population[best\_index]

best\_fitness\_value = final\_fitness\_scores[best\_index]

results\_text.insert(tk.END,

f"\nОптимизация завершена. Лучшее решение - {best\_solution}, Значение функции - {best\_fitness\_value}")

results\_text.yview\_moveto(1)

ax.scatter(best\_solution[0], best\_solution[1], best\_fitness\_value, color='black', marker='x', s=60)

results\_text.config(state=tk.DISABLED)

param\_frame2 = frame

# Параметры задачи

ttk.Label(param\_frame2, text="Инициализация значений", font=("Helvetica", 12)).grid(row=0, column=0, pady=15)

ttk.Label(param\_frame2, text="Особи", font=("Helvetica", 10)).grid(row=1, column=0)

ttk.Label(param\_frame2, text="Итерации", font=("Helvetica", 10)).grid(row=2, column=0)

x\_var = tk.DoubleVar(value=100)

y\_var = tk.DoubleVar(value=100)

x\_entry = ttk.Entry(param\_frame2, textvariable=x\_var)

y\_entry = ttk.Entry(param\_frame2, textvariable=y\_var)

x\_entry.grid(row=1, column=1)

y\_entry.grid(row=2, column=1)

separator = ttk.Separator(param\_frame2, orient="horizontal") # Горизонтальная полоса разделения

separator.grid(row=7, column=0, columnspan=2, sticky="ew", pady=10)

# Параметры функции

ttk.Label(param\_frame2, text="Функция и отображение ее графика", font=("Helvetica", 12)).grid(row=9, column=0, pady=10)

ttk.Label(param\_frame2, text="Выберите функцию", font=("Helvetica", 10)).grid(row=10, column=0)

function\_choices = ["2x^2+3y^2+4xy-6x-3y"]

function\_var = tk.StringVar(value=function\_choices[0])

function\_menu = ttk.Combobox(param\_frame2, textvariable=function\_var, values=function\_choices, width=22)

function\_menu.grid(row=10, column=1, pady=5)

ttk.Label(param\_frame2, text="X интервал (min)", font=("Helvetica", 10)).grid(row=11, column=0)

ttk.Label(param\_frame2, text="X интервал (max)", font=("Helvetica", 10)).grid(row=12, column=0)

ttk.Label(param\_frame2, text="Y интервал (min)", font=("Helvetica", 10)).grid(row=13, column=0)

ttk.Label(param\_frame2, text="Y интервал (max)", font=("Helvetica", 10)).grid(row=14, column=0)

ttk.Label(param\_frame2, text="Ось X интервал", font=("Helvetica", 10)).grid(row=16, column=0)

ttk.Label(param\_frame2, text="Ось Y интервал", font=("Helvetica", 10)).grid(row=17, column=0)

separator = ttk.Separator(param\_frame2, orient="horizontal") # Горизонтальная полоса разделения

separator.grid(row=18, column=0, columnspan=2, sticky="ew", pady=10)

x\_interval\_min = tk.DoubleVar(value=-5)

x\_interval\_max = tk.DoubleVar(value=5)

y\_interval\_min = tk.DoubleVar(value=-5)

y\_interval\_max = tk.DoubleVar(value=5)

x\_axis\_interval = tk.IntVar(value=2)

y\_axis\_interval = tk.IntVar(value=2)

x\_interval\_min\_entry = ttk.Entry(param\_frame2, textvariable=x\_interval\_min)

x\_interval\_max\_entry = ttk.Entry(param\_frame2, textvariable=x\_interval\_max)

y\_interval\_min\_entry = ttk.Entry(param\_frame2, textvariable=y\_interval\_min)

y\_interval\_max\_entry = ttk.Entry(param\_frame2, textvariable=y\_interval\_max)

x\_axis\_interval\_entry = ttk.Entry(param\_frame2, textvariable=x\_axis\_interval)

y\_axis\_interval\_entry = ttk.Entry(param\_frame2, textvariable=y\_axis\_interval)

x\_interval\_min\_entry.grid(row=11, column=1)

x\_interval\_max\_entry.grid(row=12, column=1)

y\_interval\_min\_entry.grid(row=13, column=1)

y\_interval\_max\_entry.grid(row=14, column=1)

x\_axis\_interval\_entry.grid(row=16, column=1)

y\_axis\_interval\_entry.grid(row=17, column=1)

# Создание кнопки Выполнить

button\_style = ttk.Style()

button\_style.configure("My.TButton", font=("Helvetica", 14))

# Создание кнопки Выполнить

apply\_settings\_button = ttk.Button(param\_frame2, text="Выполнить",command=run\_optimization, style="My.TButton")

apply\_settings\_button.grid(row=21, column=1, padx=10, pady=10)

ttk.Label(param\_frame2, text="Выполнение и результаты", font=("Helvetica", 12)).grid(row=20, column=0, pady=10)

results\_text = scrolledtext.ScrolledText(param\_frame2, wrap=tk.WORD, height=18, width=40, padx=2, state=tk.DISABLED)

results\_text.grid(row=21, column=0, padx=10)