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 №6 по курсу**

**«МЕТОДЫ ПОИСКОВОЙ ОПТИМИЗАЦИИ»**

Работу выполнили

Студенты 46 группы

Нагалевский А.М.

Прозоров М.С.

Преподаватель:

Нигодин Е.А.

Краснодар 2023

**Цель работы:** Разработать алгоритм оптимизации функции Розенброка искусственной иммунной сетью.

**Ход работы:**

Искусственные иммунные сети (ИИС) представляют собой вычислительные модели, вдохновленные биологической иммунной системой, и используемые для решения задач машинного обучения, оптимизации и классификации. Оптимизация функции в контексте искусственных иммунных сетей означает поиск таких параметров модели, при которых достигается наилучшая производительность или минимальная ошибка.

Рассмотрим общий алгоритм оптимизации функции искусственной иммунной сети:

1. **Инициализация популяции**: Начнем с создания начальной популяции антител (часто представляющих собой потенциальные решения задачи) в пространстве параметров. Это может быть случайная инициализация или использование более сложных методов, таких как генетические алгоритмы.
2. **Определение аффинности**: Для каждого антитела в популяции вычисляется его аффинность или "подходящесть" к решению задачи. Это может быть мера ошибки для задачи регрессии или мера качества для задачи классификации.
3. **Выбор антител для клонации и мутации**: Антитела с более высокой аффинностью имеют больше шансов быть выбранными для процесса клонации и мутации. Это позволяет сохранять лучшие решения и вносить в них изменения.
4. **Клонирование**: Выбранные антитела клонируются, создавая популяцию клонов. Это может быть выполнено путем увеличения числа копий антитела пропорционально его аффинности.
5. **Мутация**: Происходит случайное изменение клонов для разнообразия популяции. Мутации могут включать в себя изменение параметров антитела, добавление новых элементов и другие трансформации.
6. **Определение аффинности клонов**: Вычисление аффинности для клонов в новой популяции после мутации.
7. **Выбор лучших решений**: Сравнение аффинности и выбор лучших антител из изначальной популяции и их клонов.
8. **Обновление популяции**: На основе выбранных лучших антител создается новая популяция для следующей итерации.
9. **Повторение**: Шаги с 3 по 8 повторяются до достижения условия завершения, такого как достижение определенного уровня аффинности или количество итераций.

Плюсы алгоритма оптимизации функции искусственной иммунной сетью:

* **Глобальная оптимизация**: Алгоритм искусственной иммунной сети обладает способностью выполнять глобальную оптимизацию, что позволяет находить решения в широком пространстве параметров функции.
* **Адаптивность к изменениям**: Вдохновленный биологической иммунной системой, этот алгоритм обладает способностью адаптироваться к изменяющимся условиям. Он может эффективно реагировать на новые требования или изменения в структуре задачи.
* **Работа с неполными данными**: Искусственные иммунные сети могут обрабатывать данные с пропущенными значениями или шумом, что делает их устойчивыми к некоторым видам неопределенности.
* **Возможность параллельной обработки**: Алгоритм может быть эффективно распараллелен, что позволяет ускорить процесс обучения и повысить производительность в сравнении с некоторыми другими методами оптимизации.

Минусы алгоритма оптимизации функции искусственной иммунной сетью:

* **Сложность настройки параметров**: Настройка параметров искусственной иммунной сети может быть сложной задачей. Определение подходящих значений для различных параметров, таких как размер популяции, вероятность мутации и т.д., может потребовать экспериментов и опыта.
* **Сложность интерпретации**: Иногда сложно интерпретировать результаты и понять, какие конкретные изменения в параметрах привели к улучшению или ухудшению производительности модели.
* **Возможность застревания в локальных оптимумах**: Как и многие эволюционные алгоритмы, искусственные иммунные сети могут сталкиваться с проблемой застревания в локальных оптимумах, что затрудняет достижение глобальной оптимизации.
* **Вычислительная сложность**: Некоторые реализации алгоритма могут потреблять значительные вычислительные ресурсы, особенно при увеличении размера популяции и сложности задачи.
* **Неэффективность на некоторых задачах**: В зависимости от природы задачи, алгоритм может оказаться менее эффективным или несостоятельным по сравнению с другими методами оптимизации.

В целом, эффективность алгоритма искусственной иммунной сети зависит от конкретной задачи, наличия опыта в настройке параметров и характеристик самой задачи.

Для создания программы используется язык программирования Python 3.11 и среда разработки PyCharm. Для графической визуализации используется графический фреймворк Tkinter и Matplotlib.

В созданной программе одно главное активное окно.

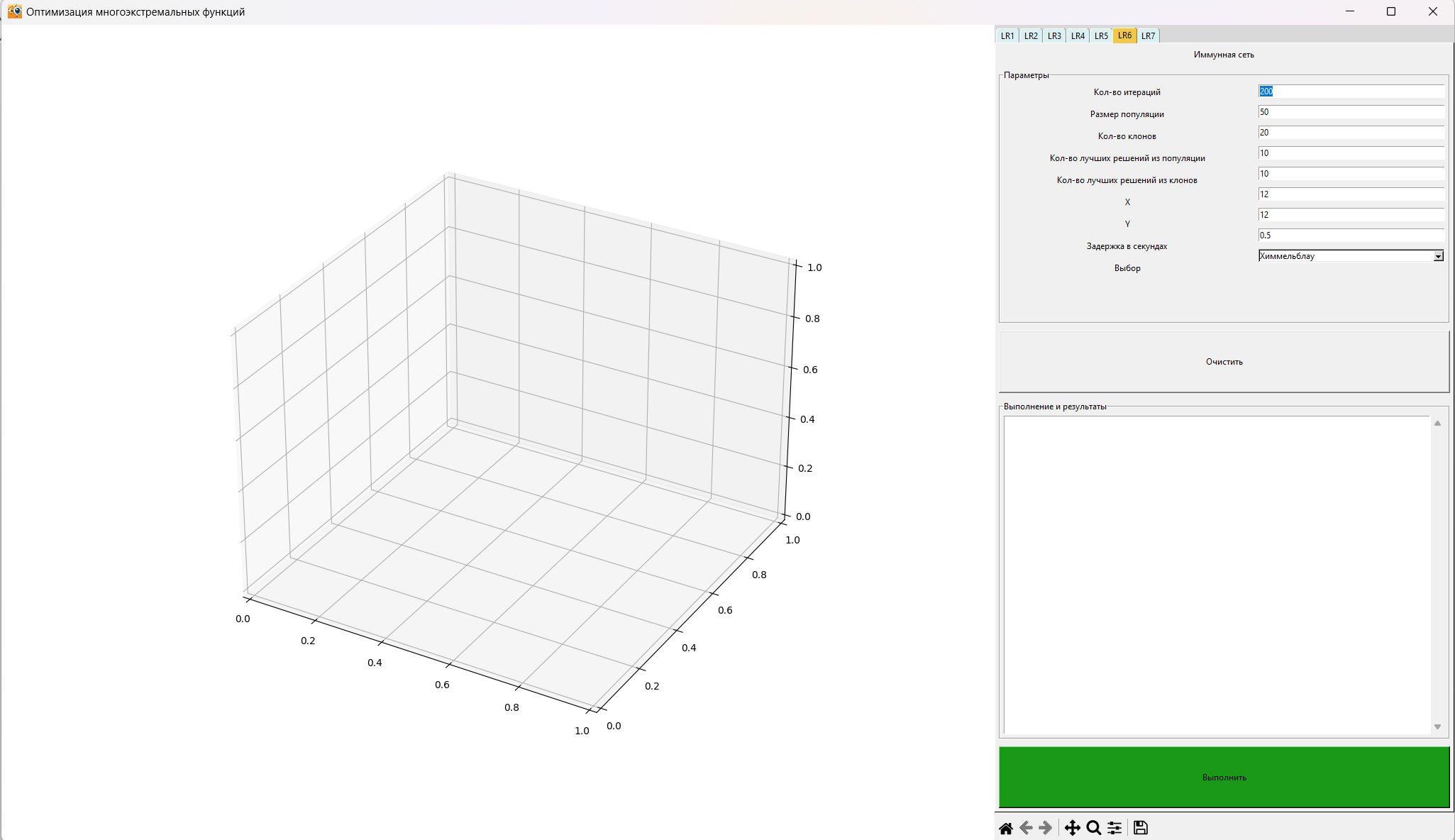


Рисунок 1 – Главное окно программы.

В данном окне можно выбрать необходимую лабораторную работу, в данном случае «6», ввести количество итераций алгоритма, размер популяции, количество клонов, количество лучших решений из популяции, количество лучших решений из клонов и задержку в секундах.

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.

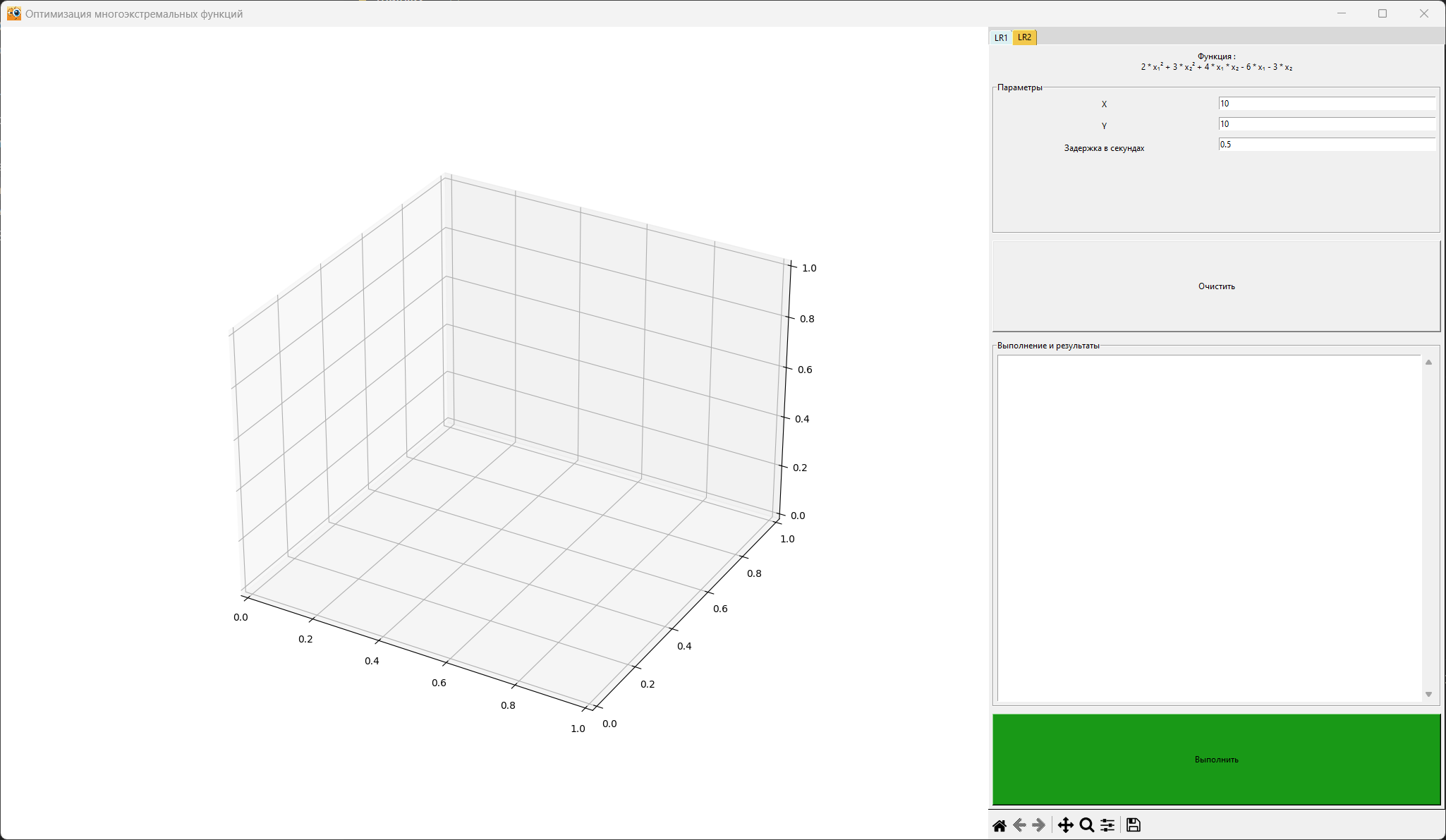


Рисунок 2 – Выполнение и результаты.

При нажатии на кнопку “Выполнить” на главном окне программы отображается необходимая нам функция, как показано на рисунке 4. Причем, более высокие значения функции показана ярко желтым цветом, а самые низкие темно серым цветом.

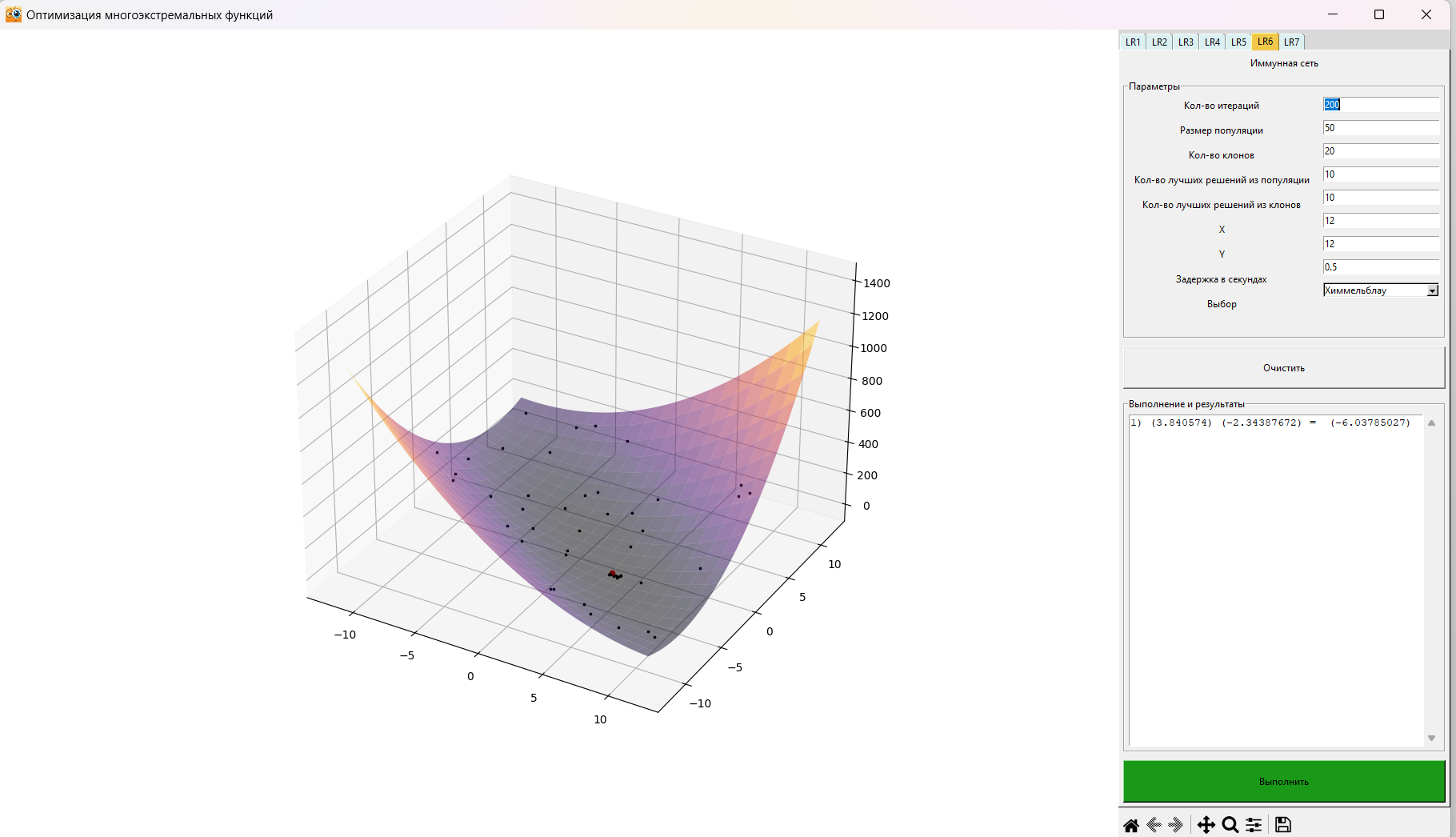


Рисунок 4 – Функция Химмельблау.

Для запуска алгоритма необходимо нажать кнопку «Выполнить», при этом поле «Выполнение и результаты» начнёт динамически заполнятся, а на отображаемой функции можно наглядно увидеть функционирование алгоритма - искомые точки, которые также отображаются динамически как показано на рисунке 5. При полном выполнении программа выводит окно, уведомляющие нас об окончании работы алгоритма.

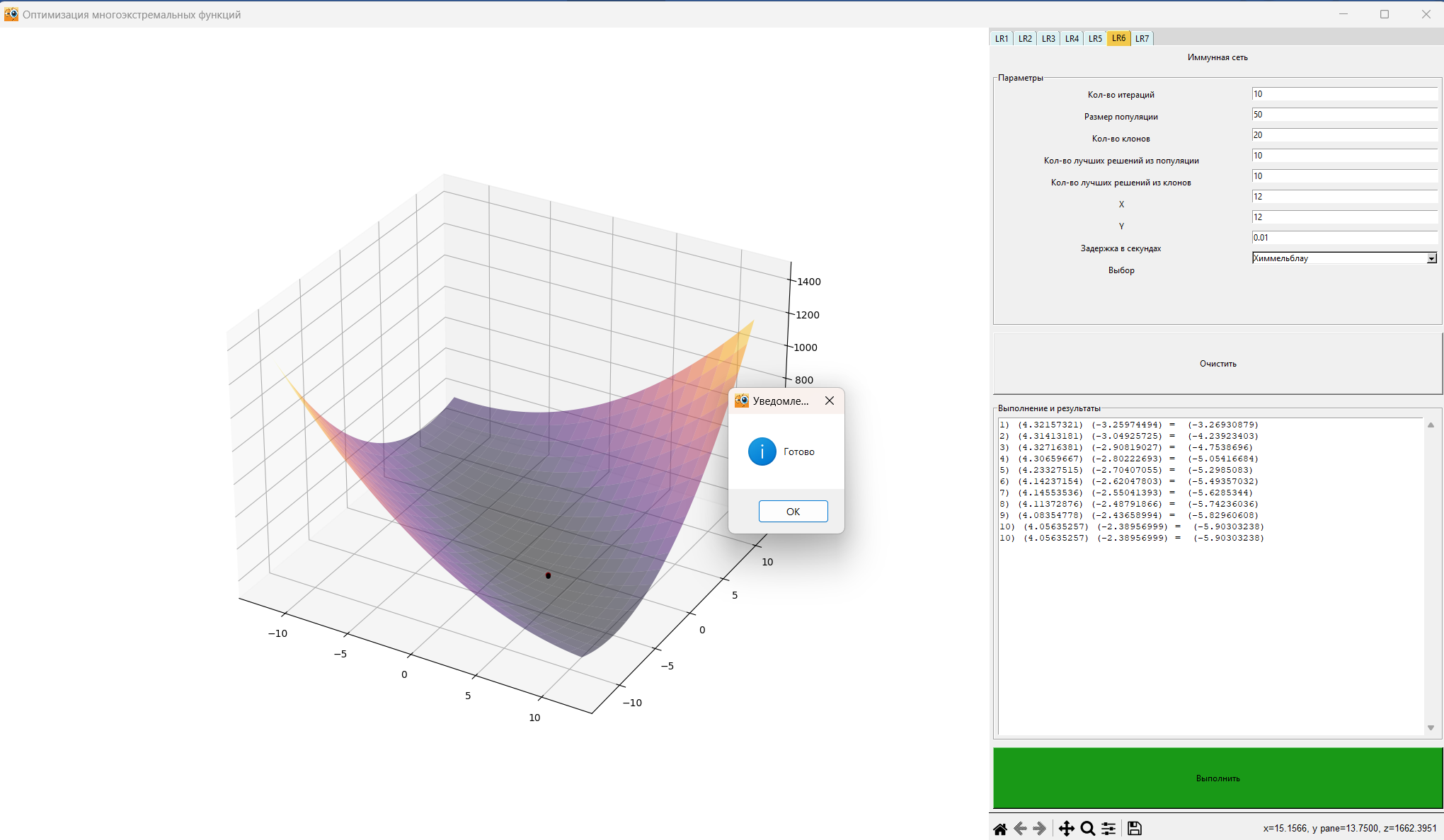


Рисунок 5 – Результат работы программы.

**Вывод:** в ходе работы был изучен и реализован алгоритм иммунной сети на определенной функции.

**Листинг программы:**

**Файл immune.py**

import random

from operator import itemgetter

from functions import \*

class Immunity:

def \_\_init\_\_(self, func, agents, clons, best, best\_clon\_numb, position\_x, position\_y):

self.func = func

self.pos\_x = float(position\_x)

self.pos\_y = float(position\_y)

self.agents\_numb = agents

self.agents = [[random.uniform(-self.pos\_x, self.pos\_x), random.uniform(-self.pos\_y, self.pos\_y), 0.0] for \_ in

range(self.agents\_numb)]

for i in self.agents:

i[2] = self.func(i[0], i[1])

self.best = best

self.best\_clon\_numb = best\_clon\_numb

self.clon\_numb = clons

def immune\_step(self, coef):

best\_pop = sorted(self.agents, key=itemgetter(2), reverse=False)[:self.best]

new\_pop = list()

for pop in best\_pop:

for \_ in range(self.clon\_numb):

new\_pop.append(pop.copy())

for npop in new\_pop:

npop[0] = npop[0] + coef \* random.uniform(-0.5, 0.5)

npop[1] = npop[1] + coef \* random.uniform(-0.5, 0.5)

npop[2] = self.func(npop[0], npop[1])

new\_pop = sorted(new\_pop, key=itemgetter(2), reverse=False)[:self.best\_clon\_numb]

self.agents += new\_pop

self.agents = sorted(self.agents, key=itemgetter(2), reverse=False)[:self.agents\_numb]

def get\_best(self):

return self.agents[0]

**Файл main.py**

import tkinter

import time

import sys

from tkinter import \*

from tkinter import scrolledtext, messagebox

from tkinter.ttk import Combobox, Notebook, Style

from matplotlib import pyplot as plt

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

from Gradient import make\_data\_lab\_1, funct\_consider

from SLSQP import make\_data\_lab\_2, kp

from Rosenbrock\_function import make\_data\_lab\_3

from genetic\_algorithm\_l3 import GeneticAlgorithmL3

from pso import PSO

from bees import Bees

from immune import Immunity

from bacterias import Bacteria

from functions import \*

def main():

window = Tk()

window.iconbitmap(r'pic/hto.ico')

width = window.winfo\_screenwidth()

height = window.winfo\_screenheight()

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

window.title("Оптимизация многоэкстремальных функций")

fig = plt.figure(figsize=(14, 14))

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

canvas = FigureCanvasTkAgg(fig, master=window)

canvas.draw()

canvas.get\_tk\_widget().pack(side=tkinter.LEFT, fill=tkinter.BOTH)

toolbar = NavigationToolbar2Tk(canvas, window)

toolbar.update()

canvas.get\_tk\_widget().pack(side=tkinter.LEFT, fill=tkinter.BOTH)

sky = "#DCF0F2"

yellow = "#F2C84B"

style = Style()

style.theme\_create("dummy", parent="alt", settings={

"TNotebook": {"configure": {"tabmargins": [2, 5, 2, 0]}},

"TNotebook.Tab": {

"configure": {"padding": [5, 1], "background": sky},

"map": {"background": [("selected", yellow)],

"expand": [("selected", [1, 1, 1, 0])]}}})

style.theme\_use("dummy")

tab\_control = Notebook(window)

def draw\_lab\_6():

fig.clf()

pop\_number = int(txt\_2\_tab\_6.get())

iter\_number = int(txt\_1\_tab\_6.get())

clon = int(txt\_3\_tab\_6.get())

best\_clon = int(txt\_5\_tab\_6.get())

best\_pop = int(txt\_4\_tab\_6.get())

pos\_x = int(txt\_6\_tab\_6.get())

pos\_y = int(txt\_7\_tab\_6.get())

delay = txt\_8\_tab\_6.get()

if combo\_tab\_6.get() == "Химмельблау":

func = himmelblau\_2

x, y, z = make\_data\_himmelblau(pos\_x, pos\_y)

else:

func = rosenbrock\_2

x, y, z = make\_data\_rosenbrock(pos\_x, pos\_y)

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

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

canvas.draw()

immunity = Immunity(func, pop\_number, clon, best\_pop, best\_clon, pos\_x, pos\_y)

for ag in immunity.agents:

ax.scatter(ag[0], ag[1], ag[2], c="black", s=1, marker="s")

b = immunity.get\_best()

ax.scatter(b[0], b[1], b[2], c="red")

canvas.draw()

window.update()

fig.clf()

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

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

canvas.draw()

for i in range(iter\_number):

immunity.immune\_step(1 / (i + 1))

for ag in immunity.agents:

ax.scatter(ag[0], ag[1], ag[2], c="black", s=1, marker="s")

b = immunity.get\_best()

ax.scatter(b[0], b[1], b[2], c="red")

txt\_tab\_6.insert(INSERT,

f"{i + 1}) ({round(b[0], 8)})"

f" ({round(b[1], 8)}) = "

f" ({round(b[2], 8)})\n")

canvas.draw()

window.update()

time.sleep(float(delay))

fig.clf()

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

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

canvas.draw()

for ag in immunity.agents:

ax.scatter(ag[0], ag[1], ag[2], c="black", s=1, marker="s")

b = immunity.get\_best()

ax.scatter(b[0], b[1], b[2], c="red")

txt\_tab\_6.insert(INSERT,

f"{i + 1}) ({round(b[0], 8)})"

f" ({round(b[1], 8)}) = "

f" ({round(b[2], 8)})\n")

canvas.draw()

window.update()

messagebox.showinfo('Уведомление', 'Готово')

def delete\_lab\_6():

txt\_tab\_6.delete(1.0, END)

tab\_6 = Frame(tab\_control)

tab\_control.add(tab\_6, text="LR6")

main\_f\_tab\_6 = LabelFrame(tab\_6, text="Параметры")

left\_f\_tab\_6 = Frame(main\_f\_tab\_6)

right\_f\_tab\_6 = Frame(main\_f\_tab\_6)

txt\_f\_tab\_6 = LabelFrame(tab\_6, text="Выполнение и результаты")

lbl\_1\_tab\_6 = Label(left\_f\_tab\_6, text="Кол-во итераций")

lbl\_2\_tab\_6 = Label(left\_f\_tab\_6, text="Размер популяции")

lbl\_3\_tab\_6 = Label(left\_f\_tab\_6, text="Кол-во клонов")

lbl\_4\_tab\_6 = Label(left\_f\_tab\_6, text="Кол-во лучших решений из клонов")

lbl\_5\_tab\_6 = Label(left\_f\_tab\_6, text="Задержка в секундах")

lbl\_6\_tab\_6 = Label(tab\_6, text="Иммунная сеть")

lbl\_7\_tab\_6 = Label(left\_f\_tab\_6, text="Кол-во лучших решений из популяции")

lbl\_8\_tab\_6 = Label(left\_f\_tab\_6, text="X")

lbl\_9\_tab\_6 = Label(left\_f\_tab\_6, text="Y")

lbl\_10\_tab\_6 = Label(left\_f\_tab\_6, text="Выбор")

txt\_1\_tab\_6 = Entry(right\_f\_tab\_6)

txt\_1\_tab\_6.insert(0, "200")

txt\_2\_tab\_6 = Entry(right\_f\_tab\_6)

txt\_2\_tab\_6.insert(0, "50")

txt\_3\_tab\_6 = Entry(right\_f\_tab\_6)

txt\_3\_tab\_6.insert(0, "20")

txt\_4\_tab\_6 = Entry(right\_f\_tab\_6)

txt\_4\_tab\_6.insert(0, "10")

txt\_5\_tab\_6 = Entry(right\_f\_tab\_6)

txt\_5\_tab\_6.insert(0, "10")

txt\_6\_tab\_6 = Entry(right\_f\_tab\_6)

txt\_6\_tab\_6.insert(0, "12")

txt\_7\_tab\_6 = Entry(right\_f\_tab\_6)

txt\_7\_tab\_6.insert(0, "12")

txt\_8\_tab\_6 = Entry(right\_f\_tab\_6)

txt\_8\_tab\_6.insert(0, "0.5")

combo\_tab\_6 = Combobox(right\_f\_tab\_6)

combo\_tab\_6['values'] = ("Химмельблау", "Розенброка")

combo\_tab\_6.set("Химмельблау")

txt\_tab\_6 = scrolledtext.ScrolledText(txt\_f\_tab\_6)

btn\_del\_tab\_6 = Button(tab\_6, text="Очистить", command=delete\_lab\_6)

btn\_tab\_6 = Button(tab\_6, text="Выполнить", foreground="black", background="#199917", command=draw\_lab\_6)

lbl\_6\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

main\_f\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH, expand=True)

left\_f\_tab\_6.pack(side=LEFT, fill=BOTH, expand=True)

right\_f\_tab\_6.pack(side=RIGHT, fill=BOTH, expand=True)

lbl\_1\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

lbl\_2\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

lbl\_3\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

lbl\_7\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

lbl\_4\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

lbl\_8\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

lbl\_9\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

lbl\_5\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

lbl\_10\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

txt\_1\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

txt\_2\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

txt\_3\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

txt\_4\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

txt\_5\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

txt\_6\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

txt\_7\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

txt\_8\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

combo\_tab\_6.pack(side=TOP, padx=5, pady=5, fill=BOTH)

txt\_tab\_6.pack(padx=5, pady=5, fill=BOTH, expand=True)

btn\_tab\_6.pack(side=BOTTOM, padx=5, pady=5, fill=BOTH, expand=True)

txt\_f\_tab\_6.pack(side=BOTTOM, padx=5, pady=5, fill=BOTH, expand=True)

btn\_del\_tab\_6.pack(side=BOTTOM, padx=5, pady=5, fill=BOTH, expand=True)

tab\_control.pack(side=RIGHT, fill=BOTH, expand=True)

window.mainloop()

if \_\_name\_\_ == '\_\_main\_\_':

main()