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 №8 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 гибридный алгоритм оптимизации функции (имунно-бактериальный).

**Ход работы:**

Одноуровневая классификация Ванга

В соответствии с классификацией Ванга выделяют три категории гибридных алгоритмов: вложенные (embedded) алгоритмы, алгоритмы типа препроцесcор/постпроцесcор (preprocessor/postprocessor), коалгоритмы (co- algorithms).

В категории методов гибридизации вложением выделяем высокоуровневую и низкоуровневую гибридизации, как это сделано в четырехуровневой классификации Рейдла.

Высокоуровневая гибридизация вложением (high-level embedded hybridization) предполагает слабую связь объединяемых алгоритмов, обычно основанную на комбинировании популяционного алгоритма глобального поиска и того или иного алгоритма локального поиска (не обязательно популяционного). Отличительным признаком высокоуровневой гибридизации является то, что объединяемые алгоритмы сохраняют значительную автономию, так что в гибридном алгоритме относительно легко выделить каждый из используемых алгоритмов. Примером высокоуровневой гибридизации вложением может служить миметический алгоритм.

При низкоуровневой гибридизации вложением (low-level embedded hybridization) комбинируемые алгоритмы интегрированы настолько сильно, что выделить составляющие итогового алгоритма обычно невозможно. Другими словами, низкоуровневая гибридизация алгоритмов порождает, по сути, новый алгоритм. К низкоуровневой гибридизации вложением может быть отнесен, например, способ оптимизации значений свободных параметров популяционного алгоритма методом адаптивного управления параметрами.

Гибридизация типа препроцессор/постпроцессор применяется чаще всего. Обычно используют два алгоритма, функционирующих последовательно один за другим. Например, начальные итерации поиска реализует генетический алгоритм, параметры которого подобраны таким образом, чтобы обеспечить широкий обзор области поиска, а завершающие итерации - метод роя частиц, обеспечивающий точную локализацию экстремума, найденного генетическим алгоритмом. В общем случае данный тип гибридизации предполагает последовательное использование более двух алгоритмов.

Выделяем два класса методов гибридизации типа препроцессор/постпроцессор - последовательная гибридизация и конвейерная гибридизация.

Последовательная гибридизация предполагает фиксированный порядок использования двух объединяемых алгоритмов, хотя момент времени (номер поколения) переключения с одного алгоритма оптимизации на другой может априори не фиксироваться. В настоящее время известны примеры гибридизации между собой по данной схеме практически всех рассмотренных популяционных алгоритмов поисковой оптимизации.

Конвейерная гибридизация исходит из того, что более двух алгоритмов исполняются последовательно и очередность включения в работу алгоритмов определяется в процессе решения задачи с помощью некоторого адаптационного алгоритма. Примером конвейерной гибридизации является алгоритм бактериальной оптимизации CBFO-S, который использует комбинацию нескольких бактериальных алгоритмов, отличающихся величиной шага перемещения бактерий.

Коалгоритмическая гибридизация алгоритмов предполагает, что объединяемые алгоритмы, как минимум, на логическом уровне выполняются параллельно, обмениваясь между собой в процессе поиска информацией. Примером такого способа гибридизации является генетический коэволюционный алгоритм.

По четырехуровневой классификации Рейдла гибридизация вложением и гибридизация по схеме препроцессор/постпроцессор реализуют интегративную стратегию гибридизации, а коалгоритмическая гибридизация - кооперативную стратегию.

Иммунно-бактериальный гибридный алгоритм объединяет принципы работы иммунной системы и бактерий для решения задач оптимизации. Каждая из этих систем вносит свои особенности, и их объединение может привести к улучшению эффективности оптимизации. Рассмотрим некоторые потенциальные преимущества такого гибридного подхода:

1. Разнообразие поисковых стратегий:

Иммунная система и бактерии имеют различные стратегии поиска и выживания. Иммунная система может обнаруживать и адаптироваться к различным условиям, проявлять разнообразие ответов на изменения. Бактерии могут использовать хемотаксис, образовывать колонии и применять другие методы. Объединение этих стратегий может привести к более эффективному и разнообразному поиску в пространстве решений.

1. Гибкость и адаптивность:

Иммунная система известна своей способностью к адаптации к изменяющимся условиям и обнаружению новых угроз. Бактерии тоже способны адаптироваться и обмениваться информацией в колониях. Гибридный алгоритм может унаследовать эти свойства, обеспечивая более гибкий и адаптивный поиск оптимальных решений.

1. Устойчивость к локальным оптимумам:

Иммунная система может помочь избегать застревания в локальных оптимумах благодаря своей способности к разнообразию и исследованию пространства решений. Бактериальные стратегии, такие как хемотаксис, могут помочь преодолевать локальные препятствия. Это содействует увеличению вероятности обнаружения глобальных оптимумов.

1. Эффективное использование вычислительных ресурсов:

Гибридный подход может совмещать преимущества обоих методов, оптимизируя использование вычислительных ресурсов. Это может быть особенно важно в случаях, когда один из методов может быть более эффективным в одних областях пространства решений, а другой - в других.

1. Улучшенная сходимость:

Комбинирование различных методов может привести к более быстрой сходимости к оптимальному решению. Например, иммунные механизмы могут быстро реагировать на области пространства решений с высокими значениями целевой функции, в то время как бактериальные стратегии могут эффективно исследовать более широкие области.

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.

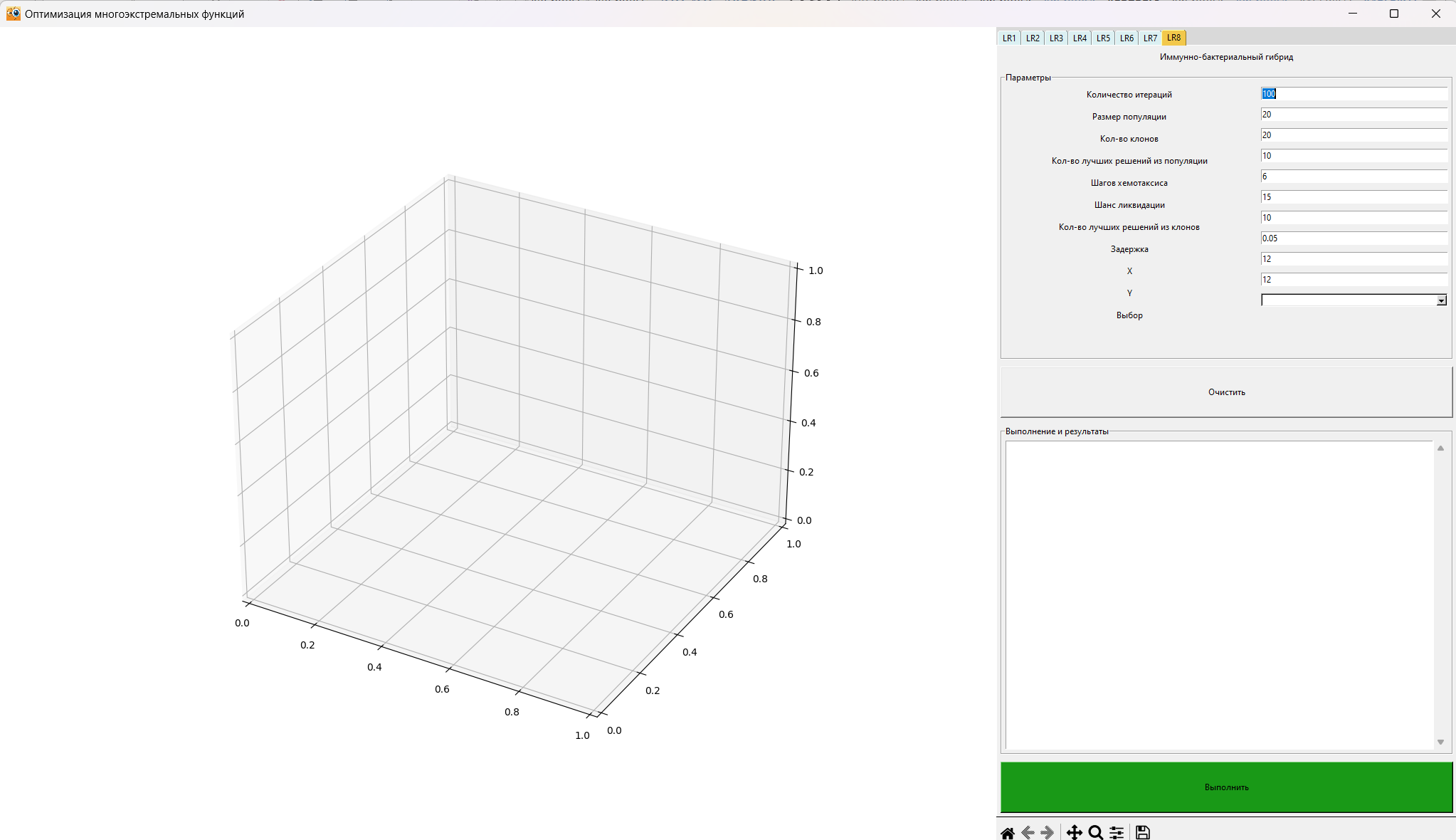


Рисунок 1 – Главное окно программы.

В данном окне можно выбрать необходимую лабораторную работу, в данном случае «8», ввести количество итераций алгоритма, размер популяции, количество клонов, количество лучших решений из популяции, шаги хемотаксиса, шанс ликвидации, 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.

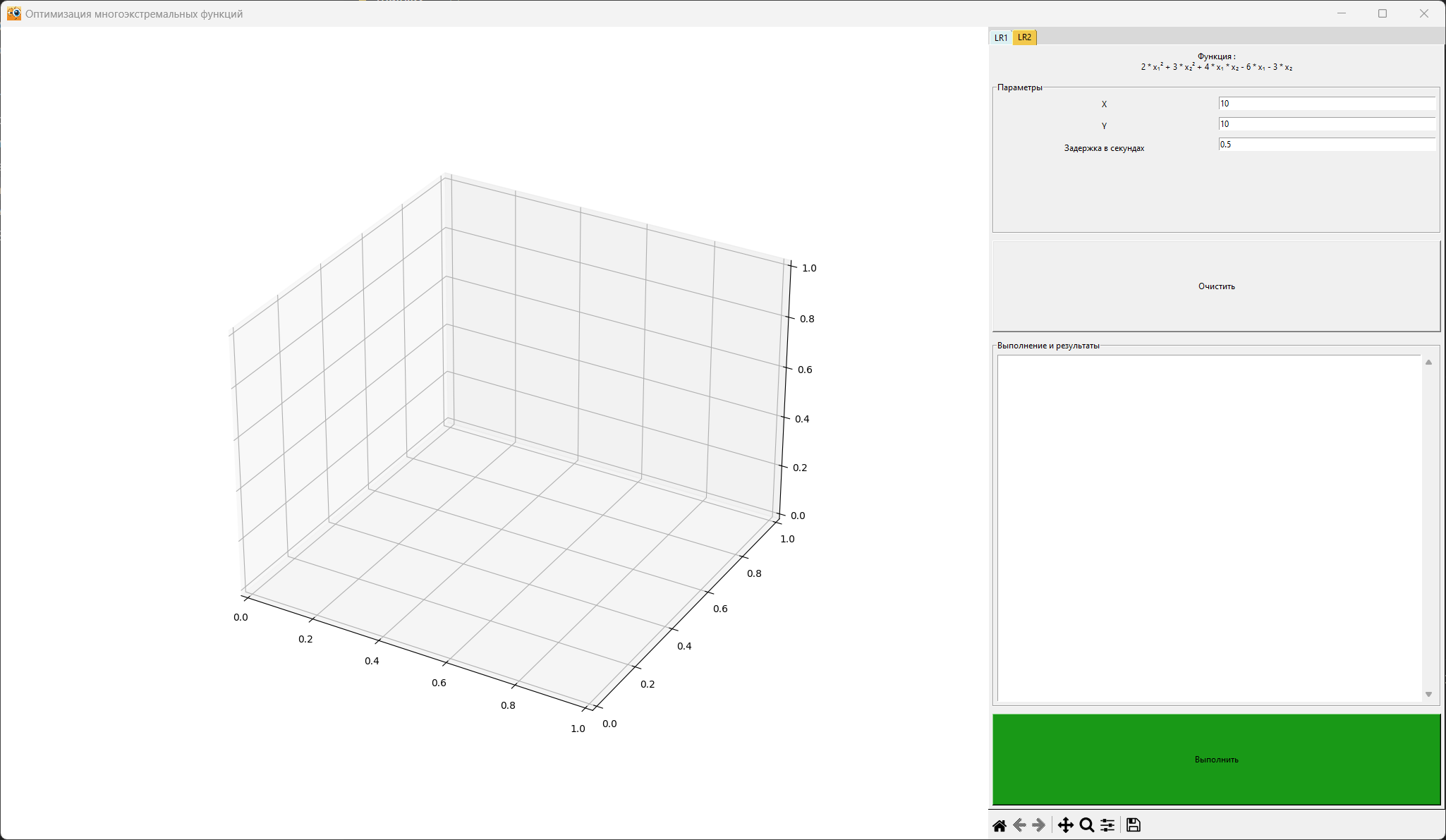


Рисунок 2 – Выполнение и результаты.

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.

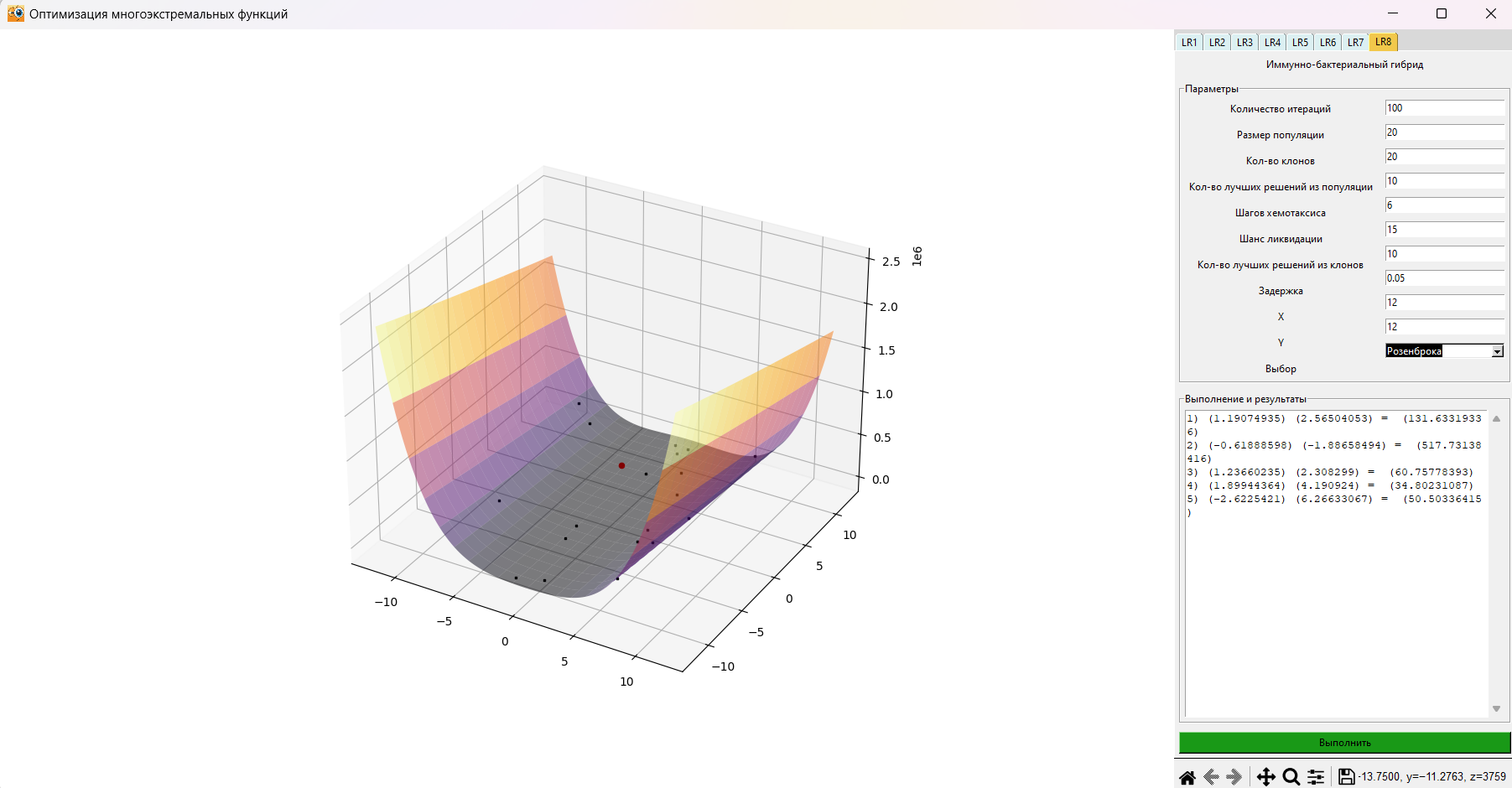


Рисунок 4 – Функция Розенброка.

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.

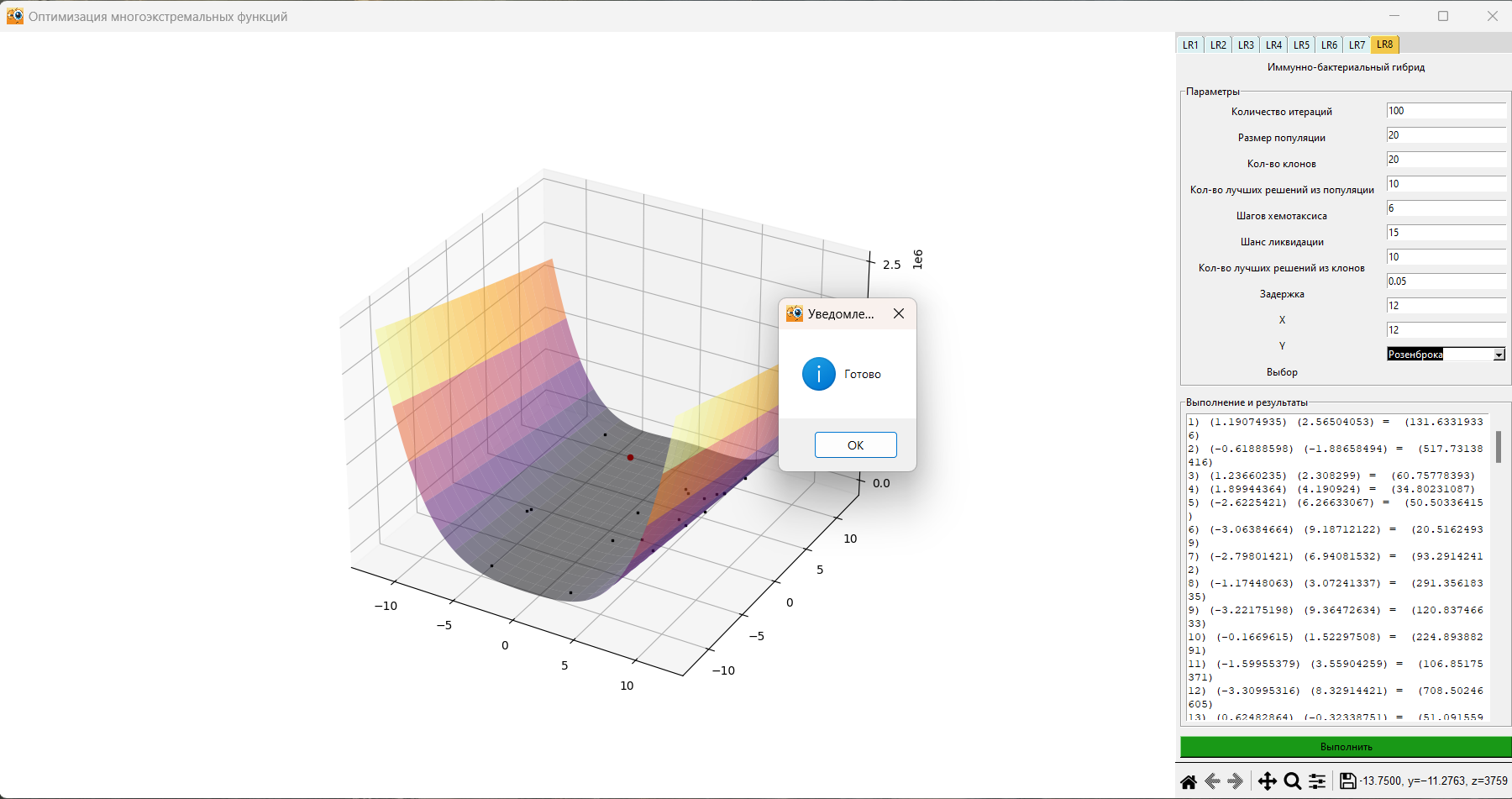


Рисунок 5 – Результат работы программы.

**Вывод:** в ходе работы был изучен и реализован гибридный имунно-бактериальный алгоритм оптимизации функции.

**Листинг программы:**

**Файл bacterias.py**

import random

from operator import itemgetter

from functions import \*

class ImmuBac:

def \_\_init\_\_(self, func, agents, clons, best, best\_clon\_numb, chemostep, licvidation, 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

self.chemo\_step = chemostep

self.licvid = licvidation

def chemotaxis(func, step, populaton, coef):

for bac in populaton:

vec = [coef \* random.uniform(-1, 1), coef \* random.uniform(-1, 1)]

for \_ in range(step):

f = bac[2]

bac[0] += vec[0] # X

bac[1] += vec[1] # Y

bac[2] = func(bac[0], bac[1]) # Z / Фитнес-Функция

if f < bac[2]:

vec = [coef \* random.uniform(-1, 1), coef \* random.uniform(-1, 1)]

return populaton

def elimnination(func, licvid, population, pos\_x, pos\_y):

for bac in population:

if random.uniform(0, 1) <= licvid:

bac[0] = random.uniform(-pos\_x, pos\_x)

bac[1] = random.uniform(-pos\_y, pos\_y)

bac[2] = func(bac[0], bac[1])

return population

def immune\_bact\_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())

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

self.agents += new\_pop

self.agents = ImmuBac.chemotaxis(self.func, self.chemo\_step, self.agents, coef).copy()

self.agents = ImmuBac.elimnination(self.func, self.licvid, self.agents, self.pos\_x, self.pos\_y).copy()

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 immune\_bacteria\_hybrid import ImmuBac

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\_8():

fig.clf()

iter\_number = int(txt\_1\_tab\_8.get())

pop\_number = int(txt\_2\_tab\_8.get())

clon = int(txt\_3\_tab\_8.get())

best\_pop = int(txt\_4\_tab\_8.get())

chemo = int(txt\_5\_tab\_8.get())

licvid = float(txt\_6\_tab\_8.get())

best\_clon = int(txt\_7\_tab\_8.get())

delay = txt\_10\_tab\_8.get()

pos\_x = int(txt\_8\_tab\_8.get())

pos\_y = int(txt\_9\_tab\_8.get())

if combo\_tab\_8.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()

immu\_ba = ImmuBac(func, pop\_number, clon, best\_pop, best\_clon, chemo, licvid, pos\_x, pos\_y)

for ag in immu\_ba.agents:

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

b = immu\_ba.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):

immu\_ba.immune\_bact\_step(1 / (i + 1))

for ag in immu\_ba.agents:

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

b = immu\_ba.get\_best()

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

txt\_tab\_8.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 immu\_ba.agents:

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

b = immu\_ba.get\_best()

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

txt\_tab\_8.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\_8():

txt\_tab\_8.delete(1.0, END)

tab\_8 = Frame(tab\_control)

tab\_control.add(tab\_8, text="LR8")

main\_f\_tab\_8 = LabelFrame(tab\_8, text="Параметры")

left\_f\_tab\_8 = Frame(main\_f\_tab\_8)

right\_f\_tab\_8 = Frame(main\_f\_tab\_8)

txt\_f\_tab\_8 = LabelFrame(tab\_8, text="Выполнение и результаты")

lbl\_5\_tab\_8 = Label(tab\_8, text="Иммунно-бактериальный гибрид")

lbl\_1\_tab\_8 = Label(left\_f\_tab\_8, text="Количество итераций")

lbl\_2\_tab\_8 = Label(left\_f\_tab\_8, text="Размер популяции")

lbl\_3\_tab\_8 = Label(left\_f\_tab\_8, text="Кол-во клонов")

lbl\_4\_tab\_8 = Label(left\_f\_tab\_8, text="Кол-во лучших решений из клонов")

lbl\_6\_tab\_8 = Label(left\_f\_tab\_8, text="Кол-во лучших решений из популяции")

lbl\_7\_tab\_8 = Label(left\_f\_tab\_8, text="Выбор")

lbl\_8\_tab\_8 = Label(left\_f\_tab\_8, text="Шагов хемотаксиса")

lbl\_9\_tab\_8 = Label(left\_f\_tab\_8, text="Шанс ликвидации")

lbl\_10\_tab\_8 = Label(left\_f\_tab\_8, text="X")

lbl\_11\_tab\_8 = Label(left\_f\_tab\_8, text="Y")

lbl\_12\_tab\_8 = Label(left\_f\_tab\_8, text="Задержка")

txt\_1\_tab\_8 = Entry(right\_f\_tab\_8)

txt\_1\_tab\_8.insert(0, "100")

txt\_2\_tab\_8 = Entry(right\_f\_tab\_8)

txt\_2\_tab\_8.insert(0, "20")

txt\_3\_tab\_8 = Entry(right\_f\_tab\_8)

txt\_3\_tab\_8.insert(0, "20")

txt\_4\_tab\_8 = Entry(right\_f\_tab\_8)

txt\_4\_tab\_8.insert(0, "10")

txt\_5\_tab\_8 = Entry(right\_f\_tab\_8)

txt\_5\_tab\_8.insert(0, "6")

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

txt\_6\_tab\_8.insert(0, "15")

txt\_7\_tab\_8 = Entry(right\_f\_tab\_8)

txt\_7\_tab\_8.insert(0, "10")

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

txt\_8\_tab\_8.insert(0, "12")

txt\_9\_tab\_8 = Entry(right\_f\_tab\_8)

txt\_9\_tab\_8.insert(0, "12")

txt\_10\_tab\_8 = Entry(right\_f\_tab\_8)

txt\_10\_tab\_8.insert(0, "0.05")

combo\_tab\_8 = Combobox(right\_f\_tab\_8)

combo\_tab\_8['values'] = ("Химмельблау", "Розенброка")

txt\_tab\_8 = scrolledtext.ScrolledText(txt\_f\_tab\_8)

btn\_del\_tab\_8 = Button(tab\_8, text="Очистить", command=delete\_lab\_8)

btn\_tab\_8 = Button(tab\_8, text="Выполнить", foreground="black", background="#199917", command=draw\_lab\_8)

lbl\_5\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

main\_f\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH, expand=True)

left\_f\_tab\_8.pack(side=LEFT, fill=BOTH, expand=True)

right\_f\_tab\_8.pack(side=RIGHT, fill=BOTH, expand=True)

lbl\_1\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

lbl\_2\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

lbl\_3\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

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

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

lbl\_9\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

lbl\_4\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

lbl\_12\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

lbl\_10\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

lbl\_11\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

lbl\_7\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

txt\_1\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

txt\_2\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

txt\_3\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

txt\_4\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

txt\_5\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

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

txt\_7\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

txt\_10\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

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

txt\_9\_tab\_8.pack(side=TOP, padx=5, pady=5, fill=BOTH)

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

txt\_tab\_8.pack(padx=5, pady=5, fill=BOTH, expand=True)

btn\_tab\_8.pack(side=BOTTOM, padx=5, pady=5, fill=BOTH, expand=True)

txt\_f\_tab\_8.pack(side=BOTTOM, padx=5, pady=5, fill=BOTH, expand=True)

btn\_del\_tab\_8.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()