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 №4 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 роя частиц оптимизации функции Розенброкка.

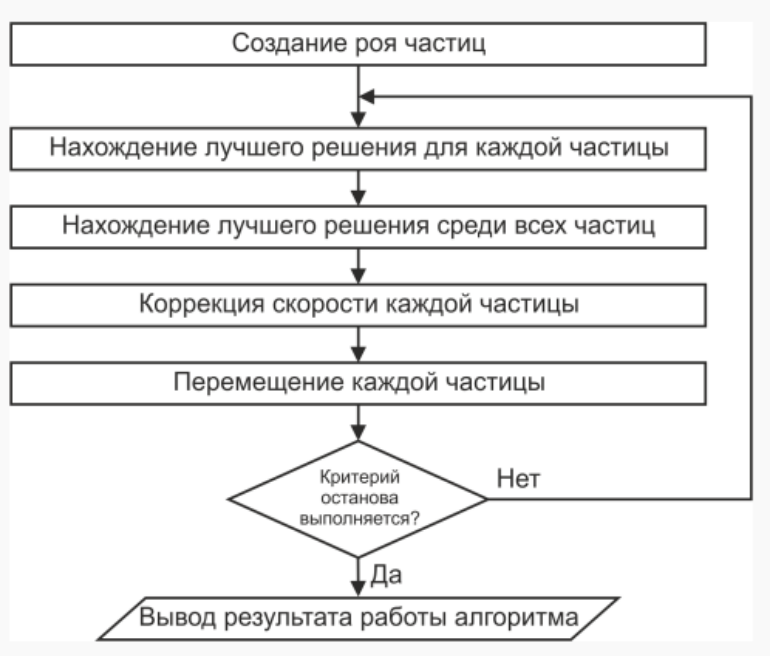
**Ход работы:**

Алгоритм роя частиц был предложен в 1995 году Джеймсом Кеннеди (Kennedy) и Расселом Еберхартом (Eberhart).

Идея алгоритма была частично заимствована из исследований поведения скоплений животных (косяков рыб, стай птиц и т.п.), модель была немного упрощена и добавлены элементы поведения толпы людей, поэтому, в отличие, например, от алгоритма пчел агенты алгоритма (возможные

решения) и были названы нейтрально - частицы.

Блок-схема алгоритма выглядит следующим образом:



Чтобы понять алгоритм роя частиц, представьте себе n-мерное пространство (область поиска), в котором рыщут частицы (агенты алгоритма). В начале частицы разбросаны случайным образом по всей области поиска и каждая частица имеет случайный вектор скорости. В каждой точке, где побывала частица, рассчитывается значение целевой функции. При этом каждая частица запоминает, какое (и где) лучшее значение целевой функции она лично нашла, а также каждая частица знает где расположена точка, являющаяся лучшей среди всех точек, которые разведали частицы. На каждой итерации частицы корректируют свою скорость (модуль и направление), чтобы, с одной стороны, быть поближе к лучшей точке, которую частица нашла сама (авторы алгоритма назвали этот аспект поведения "ностальгией"), и, в то же время, приблизиться к точке,

которая в данный момент является глобально лучшей. Через некоторое количество итераций частицы должны собраться вблизи наиболее хорошей точки, хотя возможно, что часть частиц останется где-то в относительно неплохом локальном экстремуме, но главное, чтобы хотя бы одна

частица оказалась вблизи глобального экстремума.

Самое интересное в алгоритме — это коррекция скорости, именно от этого шага зависит сходимость алгоритма. Коррекция скорости выглядит следующим образом:

Здесь – i-я компонента скорости при t-ой итерации алгоритма – i-я координата частицы при t-ой итерации алгоритма – i-я координата лучшего решения, найденного частицей – i-я координата лучшего решения, найденного всеми частицами , – случайные числа в интервале (0, 1) , – весовые коэффициенты, которые надо подбирать под конкретную задачу.

Затем корректируем текущую координату каждой частицы:

После этого рассчитываем значение целевой функции в каждой новой точке, каждая частица проверяет, не стала ли новая координата лучшей среди всех точек, где она побывала. Затем среди всех новых точек проверяем, не нашли ли мы новую глобально лучшую точку, и, если нашли, запоминаем ее координаты и значение целевой функции в ней.

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.

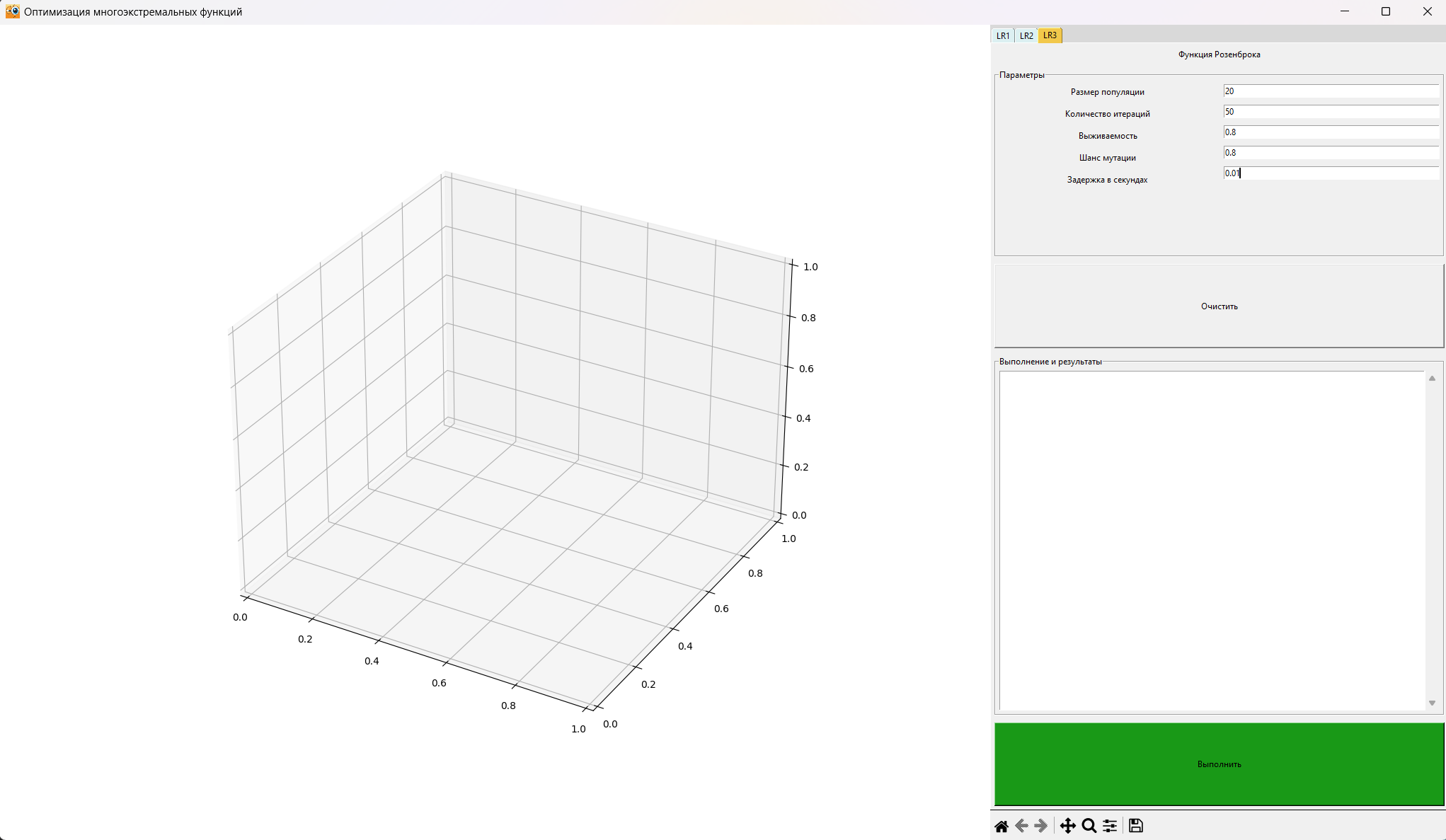


Рисунок 1 – Главное окно программы.

В данном окне можно выбрать необходимую лабораторную работу, в данном случае «3», ввести размер популяции, количество итераций алгоритма, выживаемость, шанс мутации 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.

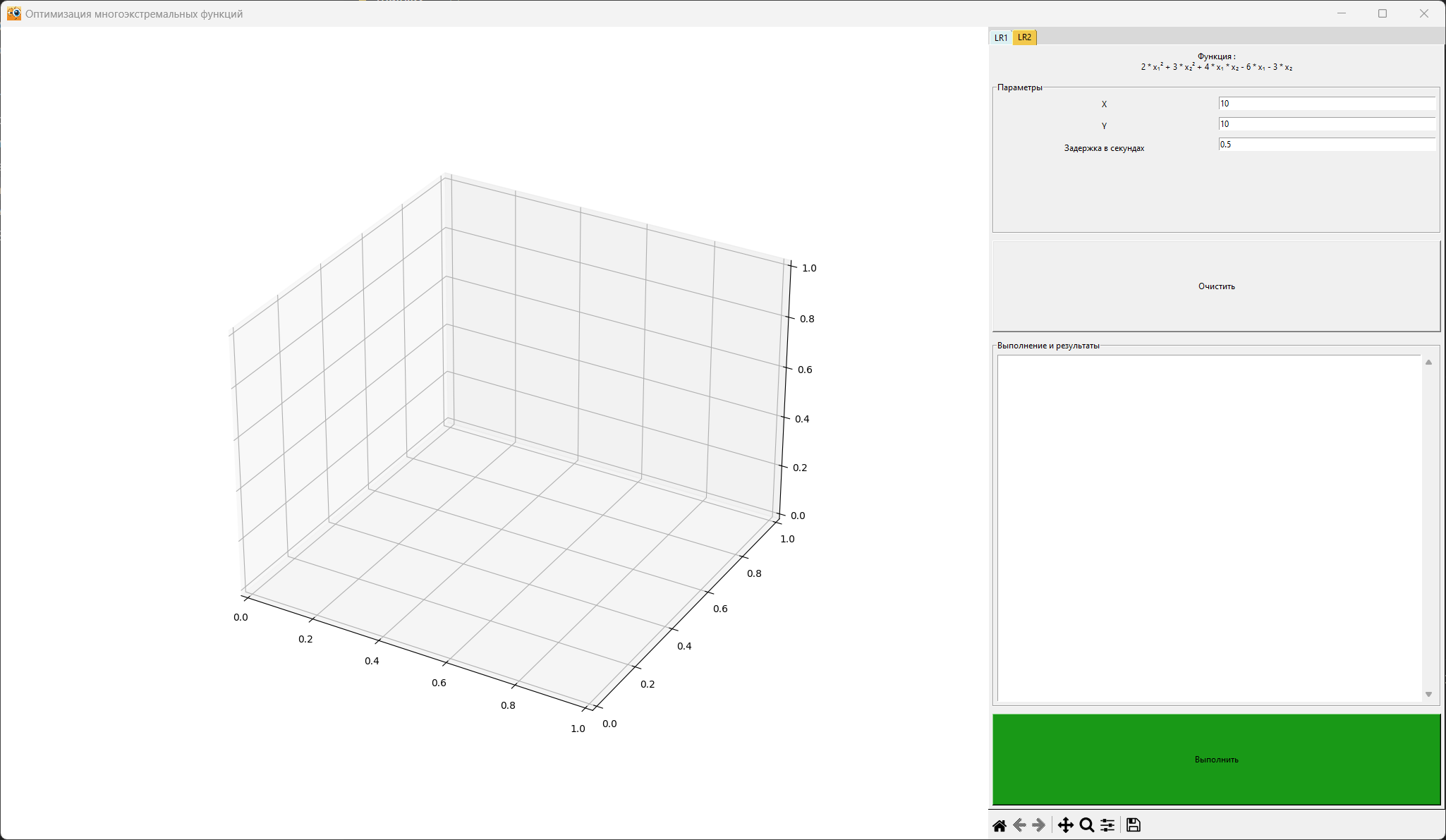


Рисунок 2 – Выполнение и результаты.

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.

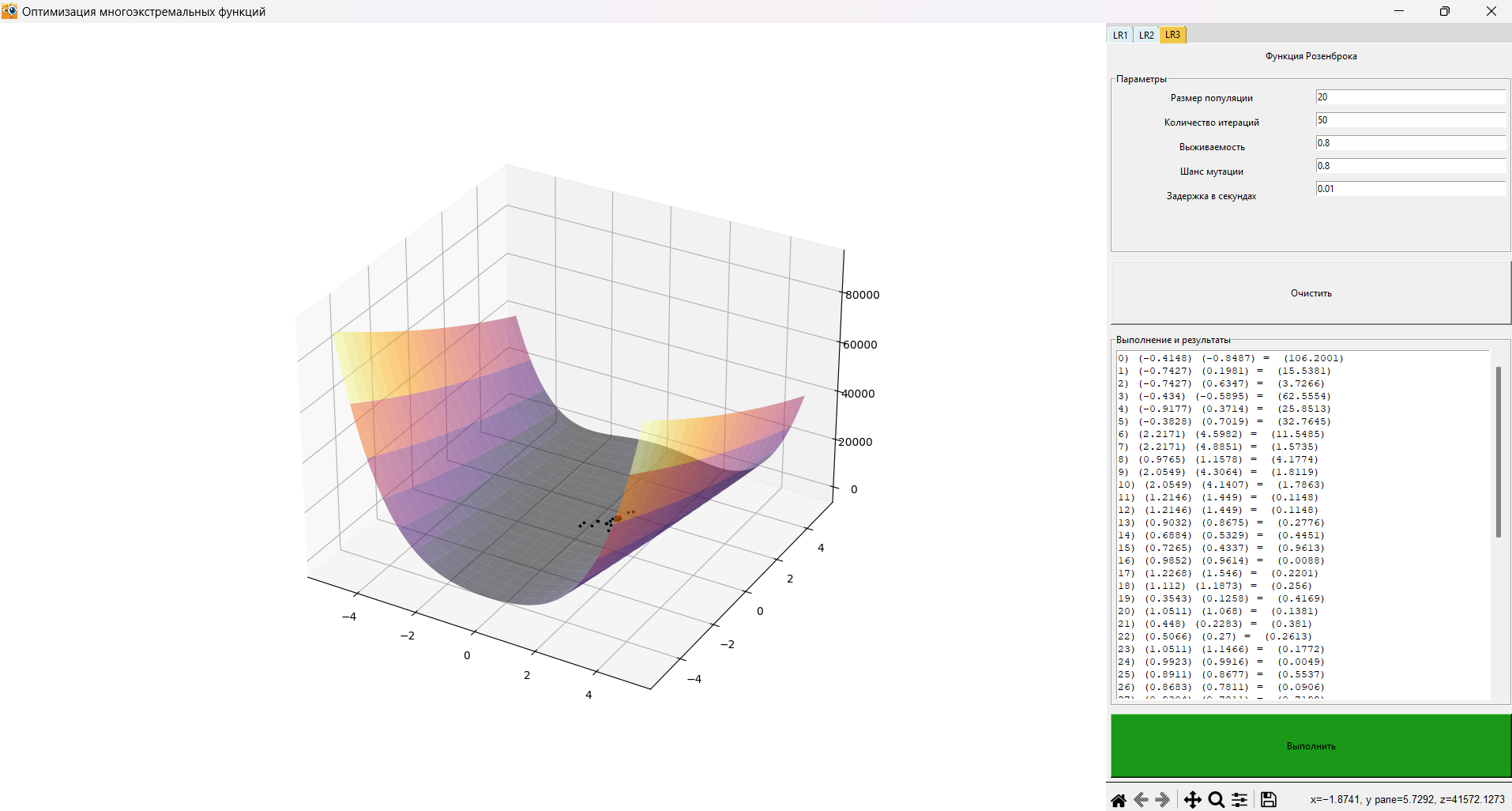


Рисунок 4 – Функция Рохенброка.

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.

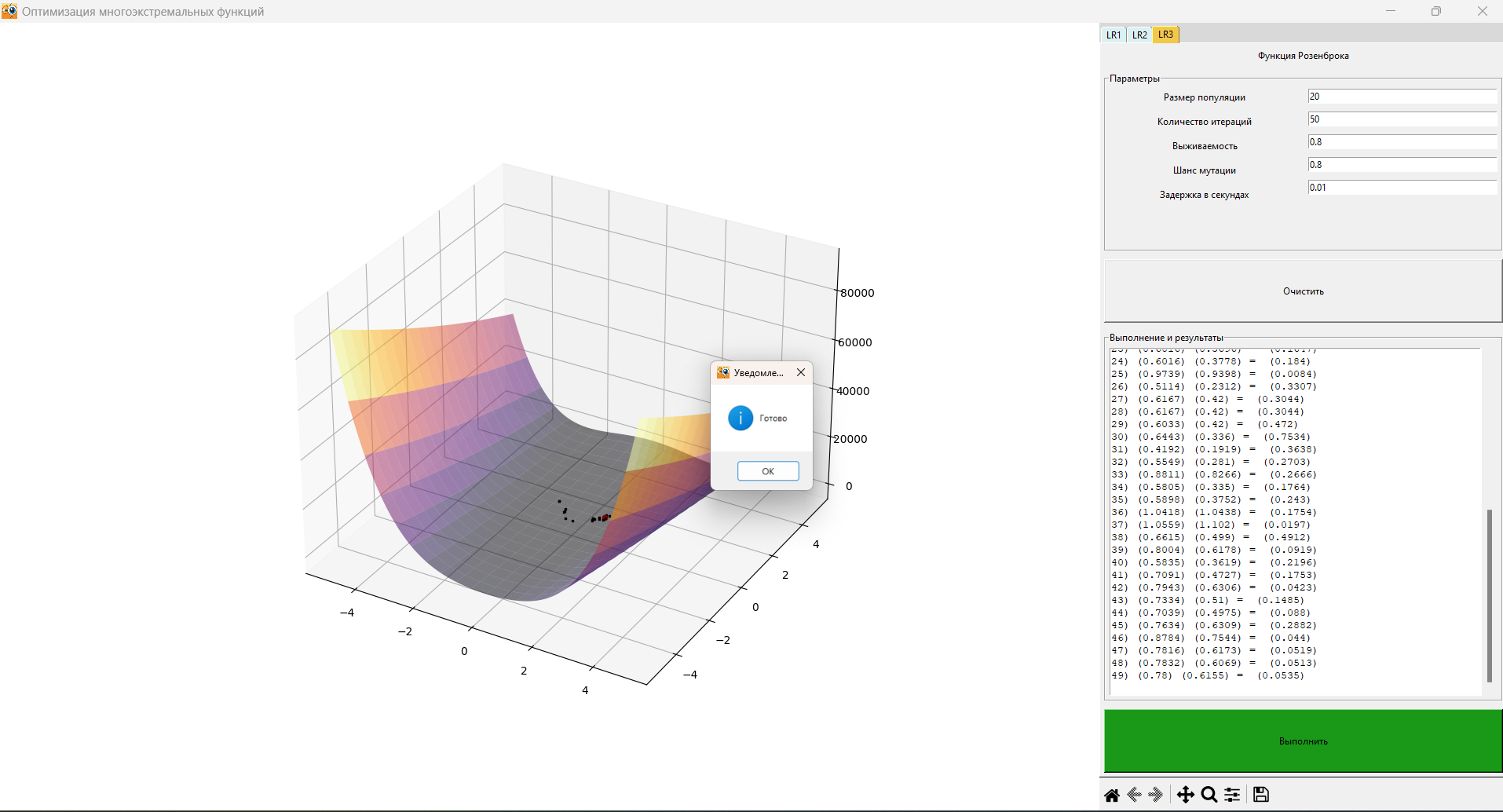


Рисунок 5 – Результат работы программы.

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

**Листинг программы:**

**Файл genetic\_algorithm\_l3.py**

from random import uniform, random

class GeneticAlgorithmL3:

def \_\_init\_\_(self, func, generations=50, min=True, mut\_chance=0.8, survive\_cof=0.8, pop\_number=100):

self.func = func

self.population = dict()

self.mut\_chance = mut\_chance

self.survive\_cof = survive\_cof

self.generations = generations

self.pop\_number = pop\_number

self.min\_func = min

def generate\_start\_population(self, x, y):

for i in range(self.pop\_number):

po\_x = uniform(-x, x)

po\_y = uniform(-y, y)

self.population[i] = [po\_x, po\_y, self.func(po\_x, po\_y)] # Создание начальной популяции

# 2 гена (x, y) и значение фитнес-функции(z)

def statistic(self):

return [max(self.population.items(), key=lambda item: item[1][2]),

min(self.population.items(), key=lambda item: item[1][2])]

def select(self):

sorted\_pop = dict(

sorted(self.population.items(), key=lambda item: item[1][2], reverse=self.min\_func)) # Ранжирование

cof = int(self.pop\_number \* (1 - self.survive\_cof))

parents1 = list(sorted\_pop.items())[cof: cof \* 2]

parents2 = list(sorted\_pop.items())[self.pop\_number - cof: self.pop\_number]

i = 0

for pop in sorted\_pop.values():

if random() > 0.5:

pop[0] = parents1[i][1][0]

pop[1] = parents2[i][1][1]

pop[2] = self.func(parents1[i][1][0], parents2[i][1][1])

else:

pop[0] = parents2[i][1][0]

pop[1] = parents1[i][1][1]

pop[2] = self.func(parents2[i][1][0], parents1[i][1][1])

i += 1

if i >= cof:

break

self.population = sorted\_pop

def mutation(self, cur\_gen):

for pop in self.population.values():

if random() < self.mut\_chance:

pop[0] += (random() - 0.5) \* ((self.generations - cur\_gen) / self.generations)

if random() < self.mut\_chance:

pop[1] += (random() - 0.5) \* ((self.generations - cur\_gen) / self.generations)

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

**Файл Rosenbrock\_function.py**

import numpy as np

def rosenbrock(x):

return np.sum(100.0 \* (x[1:] - x[:-1] \*\* 2.0) \*\* 2.0 + (1 - x[:-1]) \*\* 2.0, axis=0)

def make\_data\_lab\_3():

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

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

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

z = rosenbrock(np.array([x\_grid, y\_grid]))

return x\_grid, y\_grid, z

**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 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)

# Лаба 3

def draw\_lab\_3():

fig.clf()

x, y, z = make\_data\_lab\_3()

pop\_number = int(txt\_1\_tab\_3.get())

iter\_number = int(txt\_2\_tab\_3.get())

survive = float(txt\_3\_tab\_3.get())

mutation = float(txt\_4\_tab\_3.get())

delay = txt\_5\_tab\_3.get()

if combo\_tab\_3.get() == "Min":

min\_max = True

else:

min\_max = False

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

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

canvas.draw()

genetic = GeneticAlgorithmL3(rosenbrock\_2, iter\_number, min\_max, mutation, survive, pop\_number)

genetic.generate\_start\_population(5, 5)

for j in range(pop\_number):

ax.scatter(genetic.population[j][0], genetic.population[j][1], genetic.population[j][2], c="black", s=1,

marker="s")

if min\_max:

gen\_stat = list(genetic.statistic()[1])

else:

gen\_stat = list(genetic.statistic()[0])

ax.scatter(gen\_stat[1][0], gen\_stat[1][1], gen\_stat[1][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(50):

for j in range(pop\_number):

ax.scatter(genetic.population[j][0], genetic.population[j][1], genetic.population[j][2], c="black", s=1,

marker="s")

genetic.select()

genetic.mutation(i)

if min\_max:

gen\_stat = list(genetic.statistic()[1])

else:

gen\_stat = list(genetic.statistic()[0])

ax.scatter(gen\_stat[1][0], gen\_stat[1][1], gen\_stat[1][2], c="red")

txt\_tab\_3.insert(INSERT,

f"{i}) ({round(gen\_stat[1][0], 4)}) ({round(gen\_stat[1][1], 4)}) = "

f" ({round(gen\_stat[1][2], 4)})\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 j in range(pop\_number):

ax.scatter(genetic.population[j][0], genetic.population[j][1], genetic.population[j][2], c="black", s=1,

marker="s")

if min\_max:

gen\_stat = list(genetic.statistic()[1])

else:

gen\_stat = list(genetic.statistic()[0])

ax.scatter(gen\_stat[1][0], gen\_stat[1][1], gen\_stat[1][2], c="red")

canvas.draw()

ax.set\_xlabel('X')

ax.set\_ylabel('Y')

ax.set\_zlabel('Z')

window.update()

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

def delete\_lab\_3():

txt\_tab\_3.delete(1.0, END)

tab\_3 = Frame(tab\_control)

tab\_control.add(tab\_3, text="LR3")

main\_f\_tab\_3 = LabelFrame(tab\_3, text="Параметры")

left\_f\_tab\_3 = Frame(main\_f\_tab\_3)

right\_f\_tab\_3 = Frame(main\_f\_tab\_3)

txt\_f\_tab\_3 = LabelFrame(tab\_3, text="Выполнение и результаты")

lbl\_1\_tab\_3 = Label(left\_f\_tab\_3, text="Размер популяции")

lbl\_2\_tab\_3 = Label(left\_f\_tab\_3, text="Количество итераций")

lbl\_3\_tab\_3 = Label(left\_f\_tab\_3, text="Выживаемость")

lbl\_7\_tab\_3 = Label(left\_f\_tab\_3, text="Шанс мутации")

lbl\_4\_tab\_3 = Label(left\_f\_tab\_3, text="Выбор точки поиска")

lbl\_5\_tab\_3 = Label(left\_f\_tab\_3, text="Задержка в секундах")

lbl\_6\_tab\_3 = Label(tab\_3, text="Функция Розенброка")

txt\_1\_tab\_3 = Entry(right\_f\_tab\_3)

txt\_1\_tab\_3.insert(0,"20")

txt\_2\_tab\_3 = Entry(right\_f\_tab\_3)

txt\_2\_tab\_3.insert(0,"50")

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

txt\_3\_tab\_3.insert(0,"0.8")

txt\_4\_tab\_3 = Entry(right\_f\_tab\_3)

txt\_4\_tab\_3.insert(0,"0.8")

txt\_5\_tab\_3 = Entry(right\_f\_tab\_3)

txt\_5\_tab\_3.insert(0,"0.01")

combo\_tab\_3 = Combobox(right\_f\_tab\_3)

combo\_tab\_3['values'] = ("Min", "Max")

combo\_tab\_3.set("Min")

txt\_tab\_3 = scrolledtext.ScrolledText(txt\_f\_tab\_3)

btn\_del\_tab\_3 = Button(tab\_3, text="Очистить", command=delete\_lab\_3)

btn\_tab\_3 = Button(tab\_3, text="Выполнить", foreground="black", background="#199917", command=draw\_lab\_3)

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

main\_f\_tab\_3.pack(side=TOP, padx=5, pady=5, fill=BOTH, expand=True)

left\_f\_tab\_3.pack(side=LEFT, fill=BOTH, expand=True)

right\_f\_tab\_3.pack(side=RIGHT, fill=BOTH, expand=True)

lbl\_1\_tab\_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)

lbl\_2\_tab\_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)

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

lbl\_7\_tab\_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)

lbl\_5\_tab\_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)

#lbl\_4\_tab\_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)

txt\_1\_tab\_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)

txt\_2\_tab\_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)

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

txt\_4\_tab\_3.pack(side=TOP, padx=5, pady=5, fill=BOTH) # задержка в секундах

txt\_5\_tab\_3.pack(side=TOP, padx=5, pady=5, fill=BOTH) # шанс мутации

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

txt\_tab\_3.pack(padx=5, pady=5, fill=BOTH, expand=True)

btn\_tab\_3.pack(side=BOTTOM, padx=5, pady=5, fill=BOTH, expand=True)

txt\_f\_tab\_3.pack(side=BOTTOM, padx=5, pady=5, fill=BOTH, expand=True)

btn\_del\_tab\_3.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()