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

Стасюк К.В.

Баева Д.Н.

Преподаватель:

Нигодин Е.А.

Краснодар 2023

**Цель работы:** разработать гибридный алгоритм оптимизации функции.

**Ход работы:**

Опыт решения сложных прикладных задач, сводящихся к задаче глобальной оптимизации либо включающих в себя задачу глобальной оптимизации, показывает, что применение любого одного алгоритма оптимизации (как классического, так и популяционного) далеко не всегда приводит к успеху.

В гибридных (комбинированных) алгоритмах, объединяющих различные либо одинаковые алгоритмы, но с различными значениями свободных параметров, преимущества одного алгоритма могут компенсировать недостатки другого. Поэтому одним из основных путей повышения эффективности решения задач глобального поиска в настоящее время является разработка гибридных популяционных алгоритмов. Иногда методы комбинирования метаэвристических алгоритмов называют гиперэвристиками.

Для гибридизации был выбран генетический алгоритм популяции и алгоритм роя частиц.

Гибридный алгоритм оптимизации, который состоит из алгоритма генетической оптимизации и роевого алгоритма, является одним из наиболее эффективных методов оптимизации. Генетический алгоритм используется для генерации новых решений, а роевой алгоритм используется для улучшения этих решений.

Алгоритм гибридной оптимизации, который состоит из алгоритма генетической оптимизации и роевого алгоритма, состоит из следующих шагов:

1. Инициализация: Создание начальной популяции решений для генетического алгоритма.
2. Оценка: Оценка каждого решения в популяции с помощью функции приспособленности.
3. Селекция: Выбор лучших решений из популяции для создания новой популяции.
4. Кроссинговер: Создание новых решений путем комбинирования двух родительских решений.
5. Мутация: Изменение случайных элементов решения для создания новых решений.
6. Роевой алгоритм: Использование роевого алгоритма для улучшения решений, созданных генетическим алгоритмом.
7. Оценка: Оценка каждого решения в популяции с помощью функции приспособленности.
8. Селекция: Выбор лучших решений из популяции для создания новой популяции.
9. Повторение: Повторение шагов 4–8 до тех пор, пока не будет достигнуто оптимальное решение.

**Плюсы гибридизации:**

* Генетический алгоритм может быстро генерировать новые решения, что позволяет быстро исследовать пространство решений.
* Роевой алгоритм может быстро улучшить существующие решения, что позволяет быстро сходиться к оптимальному решению.
* Гибридный алгоритм может использовать преимущества обоих алгоритмов, чтобы достичь более высокой точности и скорости оптимизации.

**Минусы гибридизации:**

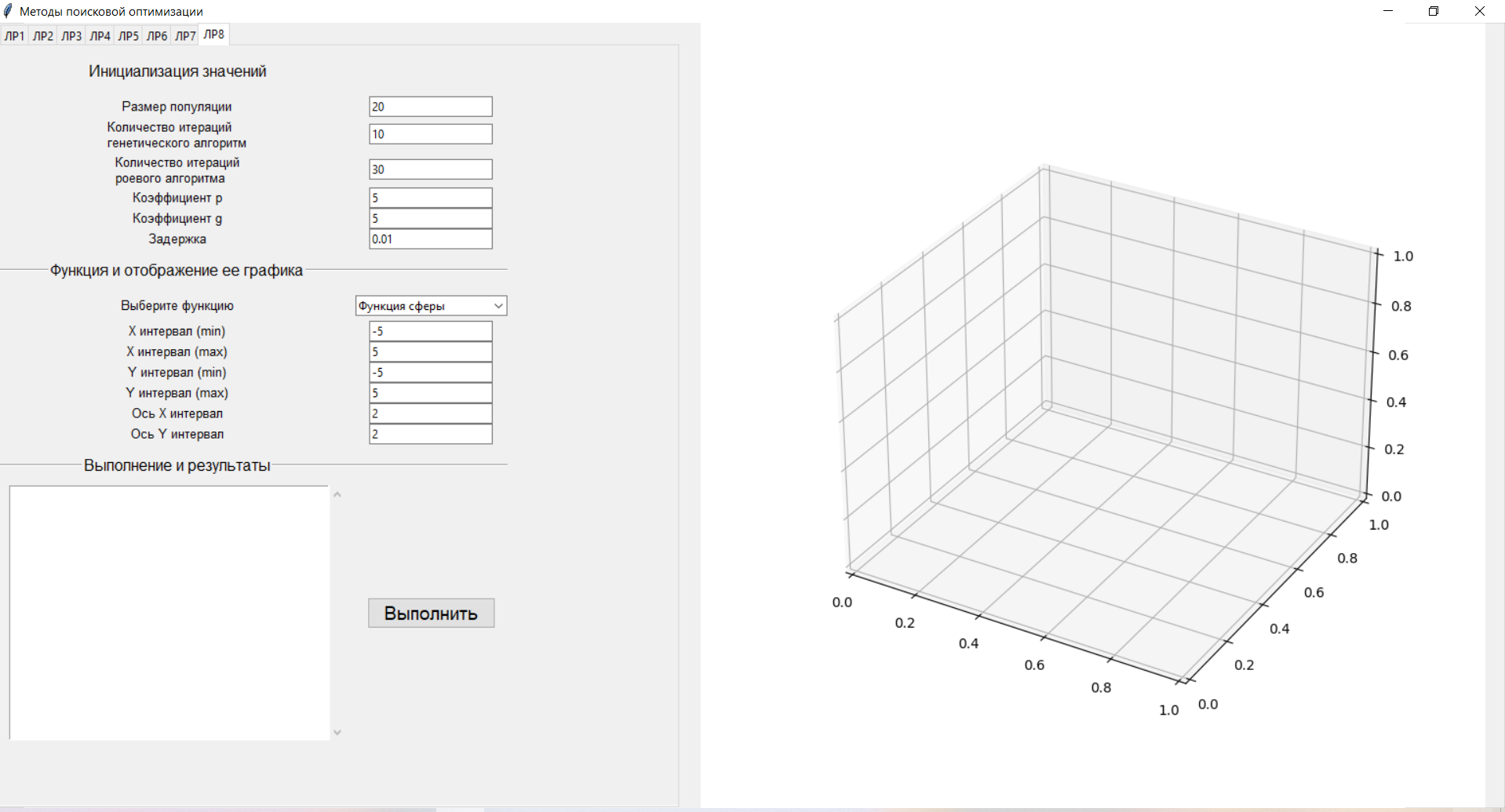
* Гибридный алгоритм может быть сложным в реализации и требовать больше вычислительных ресурсов, чем отдельные алгоритмы.
* Гибридный алгоритм может быть менее надежным, чем отдельные алгоритмы, если не настроен правильно.

Несмотря на некоторые недостатки, гибридный алгоритм оптимизации, который состоит из алгоритма генетической оптимизации и роевого алгоритма, является мощным инструментом для решения сложных задач оптимизации.

**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 «ЛР8», ввести размер популяции, количество итераций генетического алгоритма, количество итераций роевого алгоритма, коэффициент p, коэффициент g и задержка. 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

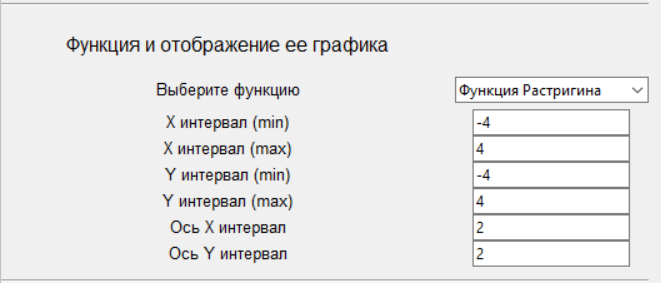
Автоматически созданное описание

Рисунок 2 – Выполнение и результаты.

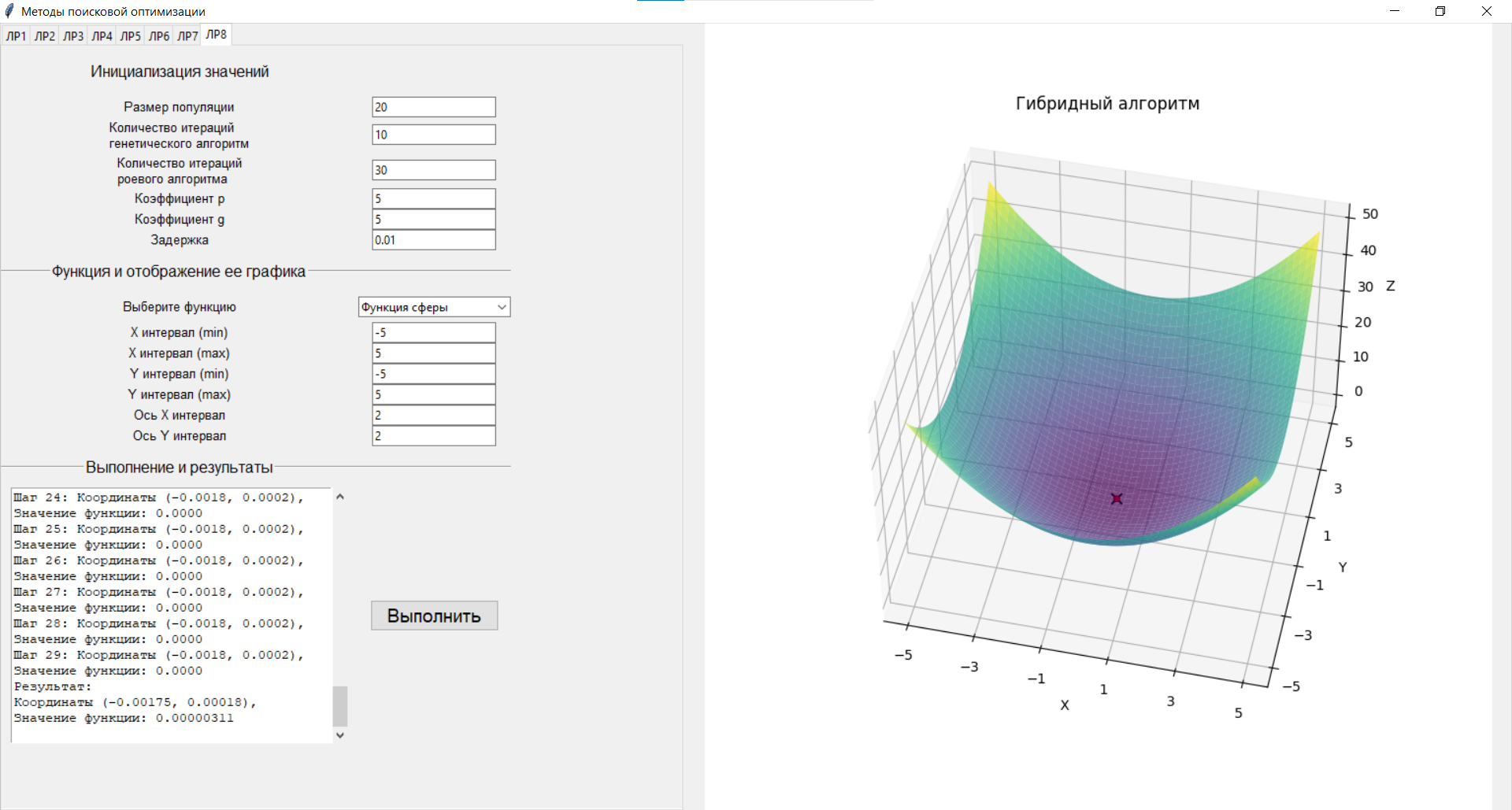
Также, в интерфейсе программы есть панель управления «Функция и отображение её графика», в котором можно выбрать необходимую функцию и задать её параметры:

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»

Рисунок 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 гибридный алгоритм оптимизации функции.

**Листинг программы**

Файл LR8.py

import tkinter as tk

from tkinter import ttk

from tkinter import scrolledtext

import numpy as np

import random

from operator import itemgetter

import time

from Genetic import GeneticAlgorithm

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

def sphere(x, y):

return x \*\* 2 + y \*\* 2

def rosenbrock\_function(x,y):

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

class PSO:

def \_\_init\_\_(self, func, pop\_size, particles, position\_x, position\_y, fi\_p, fi\_g):

self.func = func

self.pop\_size = pop\_size

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

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

assert fi\_p + fi\_g > 4, "Сумма коэффициентов должна быть > 4"

self.fi\_p = fi\_p

self.fi\_g = fi\_g

# инерционный вес

self.xi = 2 / (np.abs(2 - (fi\_p + fi\_g) - np.sqrt((fi\_p + fi\_g) \*\* 2 - 4 \* (fi\_p + fi\_g))))

self.particles = list(particles.values())

self.nostalgia = self.particles.copy()

self.velocity = [[0.0 for \_ in range(2)] for \_ in range(self.pop\_size)]

self.particle\_best = min(self.particles, key=itemgetter(2))

def update\_velocity(self, velocity, particle, point\_best) -> list:

new\_vel = list()

for i in range(2):

r\_p = random.random()

r\_g = random.random()

new\_vel.append(self.xi \* (velocity[i] + self.fi\_p \* r\_p \* (

point\_best[i] - particle[i]) + self.fi\_g \* r\_g \*

(self.particle\_best[i] - particle[i])))

return new\_vel

def update\_position(self, velocity, particle):

x = particle[0] + velocity[0]

y = particle[1] + velocity[1]

return [x, y, self.func(x, y)]

def next\_iteration(self):

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

if self.nostalgia[i][2] < self.particles[i][2]:

point\_best = self.nostalgia[i]

else:

self.nostalgia[i] = self.particles[i]

point\_best = self.particles[i]

self.velocity[i] = PSO.update\_velocity(self, self.velocity[i], self.particles[i],

point\_best)

self.particles[i] = PSO.update\_position(self, self.velocity[i], self.particles[i])

self.particle\_best = min(self.particles, key=itemgetter(2))

def run\_optimization():

function\_choice = function\_var.get()

target\_func = sphere

if function\_choice == "Функция сферы":

target\_func = sphere

elif function\_choice == "Функция Розенброка":

target\_func = rosenbrock\_function

pso\_iter = pso\_iterations\_var.get()

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

coef\_p = coef\_p\_var.get()

coef\_g = coef\_g\_var.get()

ga\_iter = ga\_iterations\_var.get()

delay = delay\_var.get()

# 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\_func(X, Y)

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("Гибридный алгоритм")

genetic = GeneticAlgorithm(target\_func, ga\_iter, population\_size,

[x\_interval\_min.get(), x\_interval\_max.get()],

[y\_interval\_min.get(), y\_interval\_max.get()])

genetic.generate\_start\_population()

# отрисовка стартовой популяции

for j in range(population\_size):

ax.scatter(genetic.population[j][0], genetic.population[j][1], genetic.population[j][2],

c="red", s=10)

best\_individual = genetic.get\_best\_individual()

# ax.scatter(ais.antibody\_best[0], ais.antibody\_best[1], ais.antibody\_best[2], c="blue")

canvas.draw()

root.update()

# очистка графика

ax.cla()

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("Гибридный алгоритм")

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

canvas.draw()

cnt = 0

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

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

# отрисовка промежуточной популяции и эволюция

for i in range(ga\_iter):

for j in range(population\_size):

# отрисовка промежуточной популяции

ax.scatter(genetic.population[j][0], genetic.population[j][1], genetic.population[j][2],

c="red", s=10)

genetic.select()

genetic.mutation(i)

best\_individual = genetic.get\_best\_individual()

# ax.scatter(best\_individual[1][0], best\_individual[1][1], best\_individual[1][2], c="green")

results\_text.insert(tk.END,

f"Шаг {i}: Координаты ({best\_individual[1][0]:.4f}, {best\_individual[1][1]:.4f}),"

f" Значение функции: {best\_individual[1][2]:.4f}\n")

results\_text.yview\_moveto(1)

canvas.draw()

root.update()

time.sleep(delay)

# очистка графика

ax.cla()

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("Гибридный алгоритм")

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

canvas.draw()

# отрисовка результирующей популяции

# отрисовка результирующей популяции

for j in range(population\_size):

ax.scatter(genetic.population[j][0], genetic.population[j][1], genetic.population[j][2],

c="red", s=10)

best\_individual = genetic.get\_best\_individual()

ax.scatter(best\_individual[1][0], best\_individual[1][1], best\_individual[1][2], c="green")

canvas.draw()

root.update()

# очистка графика

ax.cla()

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("Гибридный алгоритм")

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

canvas.draw()

results\_text.insert(tk.END, "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nНачало работы РА\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n")

results\_text.yview\_moveto(1)

pso = PSO(target\_func, population\_size, genetic.population, 5, 5, coef\_p, coef\_g)

# отрисовка стартовой популяции

for particle in pso.particles:

ax.scatter(particle[0], particle[1], particle[2], c="red", s=10)

canvas.draw()

root.update()

# очистка графика

ax.cla()

ax.set\_xlabel('X')

ax.set\_ylabel('Y')

ax.set\_zlabel('Z')

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

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("Гибридный алгоритм")

canvas.draw()

# отрисовка промежуточной популяции и эволюция

for i in range(pso\_iter):

pso.next\_iteration()

for particle in pso.particles:

# отрисовка промежуточной популяции

ax.scatter(particle[0], particle[1], particle[2], c="red", s=10)

results\_text.insert(tk.END,

f"Шаг {i}: Координаты ({pso.particle\_best[0]:.4f}, "

f"{pso.particle\_best[1]:.4f}),"

f" Значение функции: {pso.particle\_best[2]:.4f}\n")

results\_text.yview\_moveto(1)

canvas.draw()

root.update()

time.sleep(delay)

# очистка графика

ax.cla()

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("Гибридный алгоритм")

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

canvas.draw()

# отрисовка результирующей популяции

for particle in pso.particles:

ax.scatter(particle[0], particle[1], particle[2], c="red", s=10)

ax.scatter(pso.particle\_best[0], pso.particle\_best[1], pso.particle\_best[2], color='black', marker='x',

s=60)

canvas.draw()

root.update()

results\_text.insert(tk.END,

f"Результат:\nКоординаты ({pso.particle\_best[0]:.5f}, "

f"{pso.particle\_best[1]:.5f}),\nЗначение функции: {pso.particle\_best[2]:.8f}\n")

results\_text.yview\_moveto(1)

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="Размер популяции", font=("Helvetica", 10)).grid(row=1, column=0)

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

ttk.Label(param\_frame2, text="Количество итераций\nроевого алгоритма", font=("Helvetica", 10)).grid(row=3, column=0)

ttk.Label(param\_frame2, text="Коэффициент p", font=("Helvetica", 10)).grid(row=4, column=0)

ttk.Label(param\_frame2, text="Коэффициент g", font=("Helvetica", 10)).grid(row=5, column=0)

ttk.Label(param\_frame2, text="Задержка", font=("Helvetica", 10)).grid(row=6, column=0)

population\_size\_var = tk.IntVar(value=20)

ga\_iterations\_var = tk.IntVar(value=10)

pso\_iterations\_var = tk.IntVar(value=30)

coef\_p\_var = tk.DoubleVar(value=5)

coef\_g\_var = tk.DoubleVar(value=5)

delay\_var = tk.DoubleVar(value=0.01)

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

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

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

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

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

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

population\_size\_entry.grid(row=1, column=1)

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

pso\_iterations\_entry.grid(row=3, column=1)

coef\_p\_entry.grid(row=4, column=1)

coef\_g\_entry.grid(row=5, column=1)

delay\_entry.grid(row=6, column=1)

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

separator.grid(row=9, 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 = ["Нажмите для выбора", "Функция сферы", "Функция Розенброка"]

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=18, column=0, pady=10)

results\_text = scrolledtext.ScrolledText(param\_frame2, wrap=tk.WORD, height=16, width=40, padx=2, state=tk.DISABLED)

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

root.mainloop()