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 №7 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 алгоритм бактериальной оптимизации обратной сферической функции.

**Ход работы:**

Алгоритм бактериальной оптимизации (BFO) — это метод оптимизации, который использует аналогии с поведением бактерий при поиске пищи для решения задач оптимизации. Он был разработан в 1990-х годах и с тех пор получил широкое распространение в научных кругах. Поведение бактерий обусловлено механизмом, который называется бактериальным хемотаксисом и представляет собой двигательную реакцию микроорганизмов на химический раздражитель. Данный механизм позволяет бактерии двигаться по направлениям к аттрактантам (чаще всего, питательным веществам) и от репеллентов (потенциально вредных для бактерии веществ).

Оптимизация функции алгоритмом бактериальной оптимизации включает в себя следующие шаги.

1. Инициализация:

* Создается популяция бактерий в пространстве поиска.
* Каждая бактерия представлена своей позицией и связанным с ней значением пригодности.
* Лучшая бактерия изначально устанавливается как та, у которой минимальное значение пригодности в популяции.

1. Хемотаксис:

* На каждом шаге генерируется случайный размер шага для координат x и y.
* Позиция бактерии обновляется, добавляя шаг к обеим координатам.
* Если пригодность новой позиции лучше, чем текущая, обновляется позиция и пригодность бактерии.
* Если пригодность не улучшается, позиция возвращается к предыдущему состоянию.

1. Репродукция:

* Здоровье каждой бактерии обновляется, добавляя ей текущее значение пригодности.
* Бактерии сортируются по их здоровью в порядке возрастания.
* Вторая половина бактерий заменяется первой половиной.
* Это имитирует процесс репродукции, где здоровые бактерии имеют более высокие шансы передать свои характеристики следующему поколению.

1. Ликвидация и рассеивание:

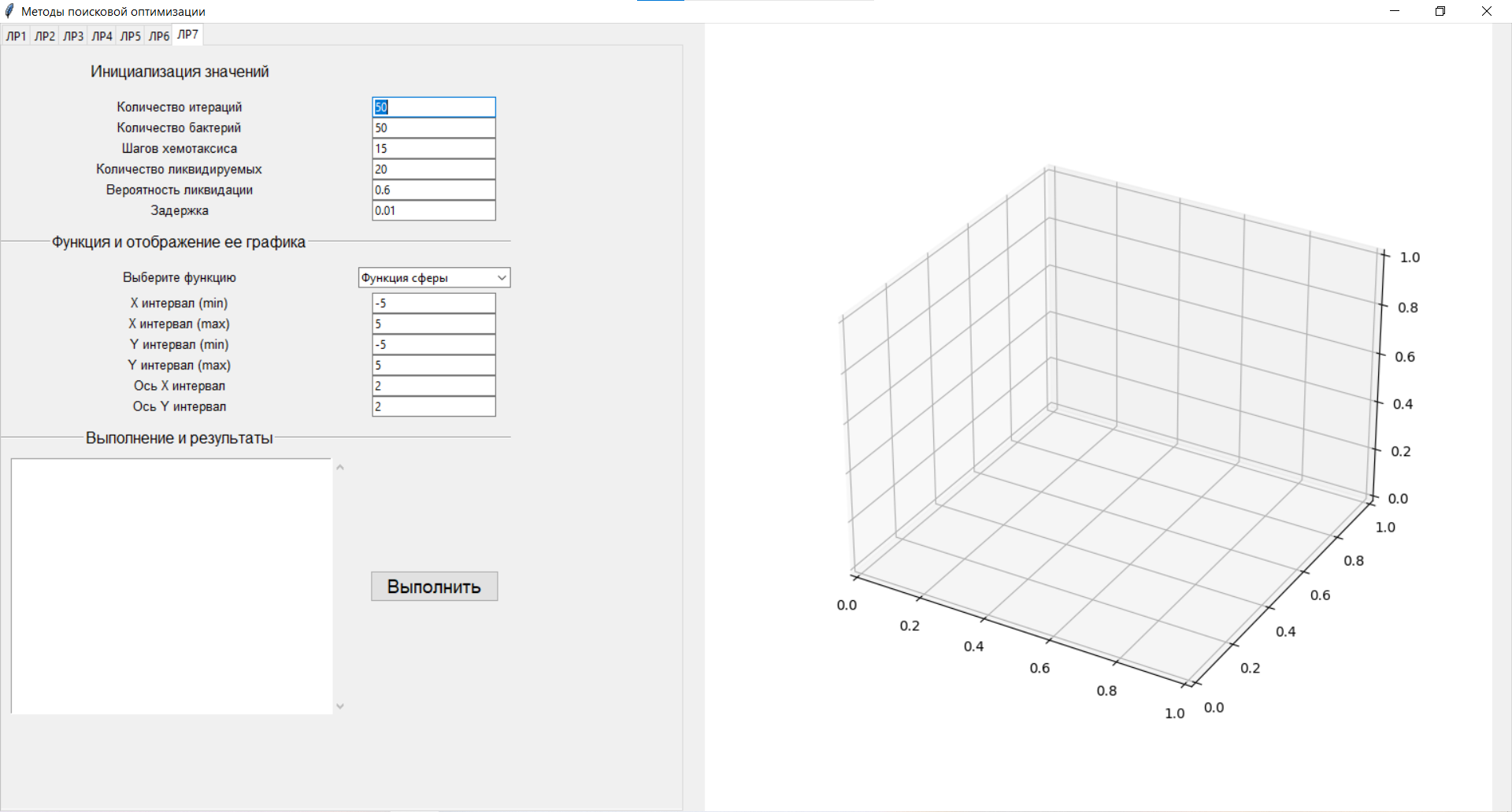
* Выбирается определенное количество бактерий для ликвидации.
* Для каждой выбранной бактерии случайным образом решается, будет ли она ликвидирована.
* Если случайное число больше вероятности ликвидации, позиция бактерии заменяется новой случайной позицией, и ей присваивается новое значение пригодности.

Преимущества алгоритма бактериальной оптимизации включают его способность решать сложные задачи оптимизации, а также его эффективность в поиске нескольких оптимальных решений. Недостатки включают сложность понимания и реализации алгоритма, а также его чувствительность к выбору параметров и настроек.

**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 «ЛР7», ввести количество итераций, количество бактерий, шагов хемотаксиса, количество ликвидируемых, вероятность ликвидации, 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

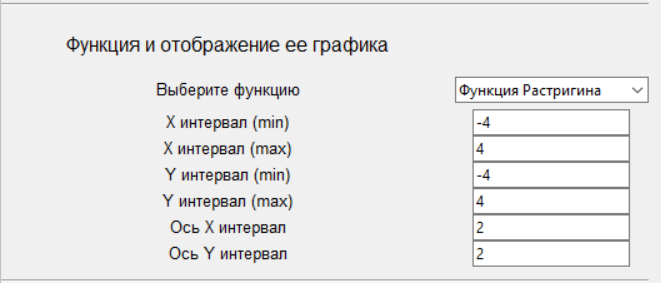
Автоматически созданное описание

Рисунок 2 – Выполнение и результаты.

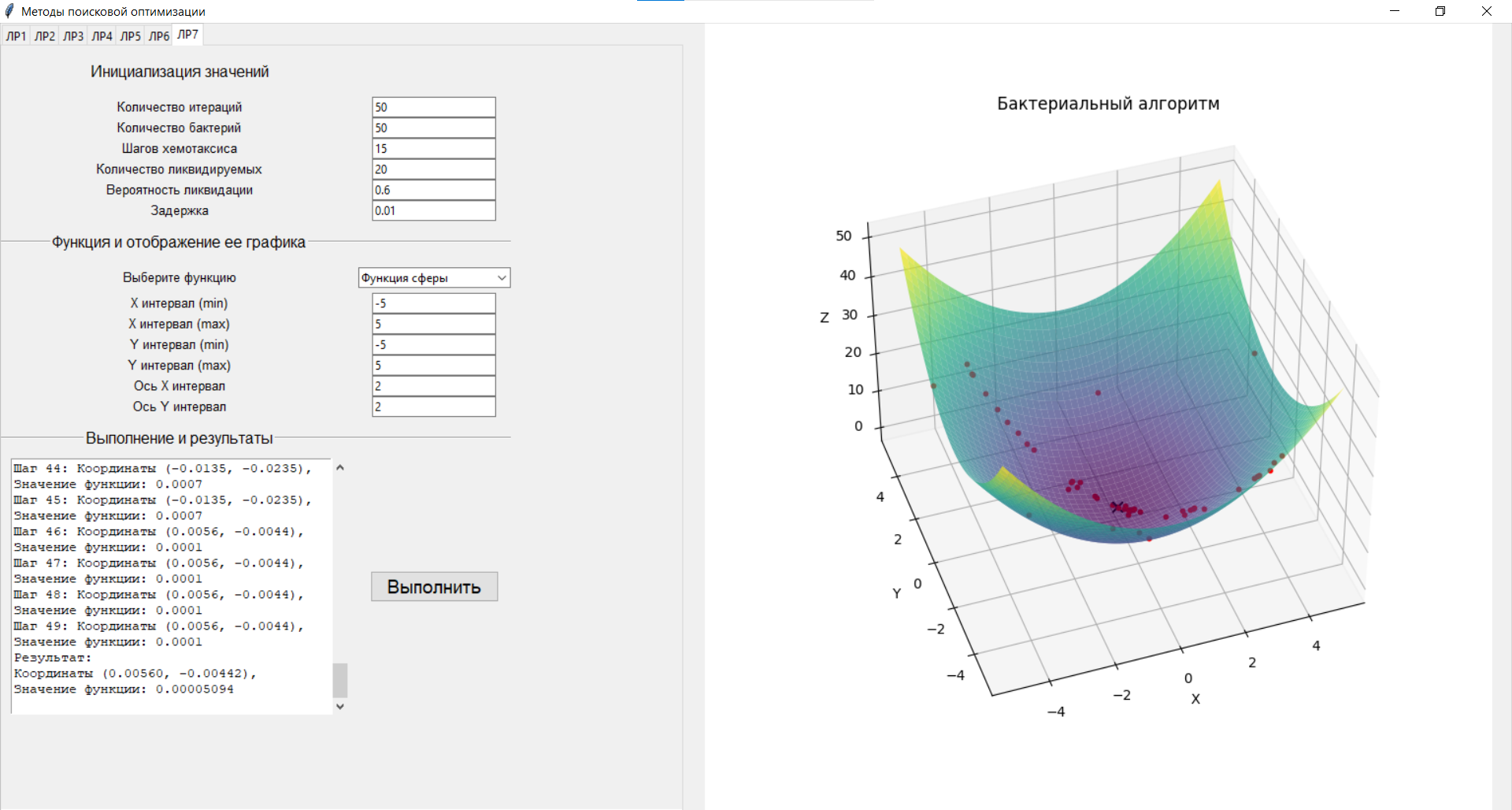
Также, в интерфейсе программы есть панель управления «Функция и отображение её графика», в котором можно выбрать необходимую функцию и задать её параметры:

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 алгоритм бактериальной оптимизации обратной сферической функции.

**Листинг программы**

Файл LR7.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  
  
  
def BFO(frame,root,ax,canvas):  
 # Функция Розенброка для оптимизации  
 def sphere(x, y):  
 return x \*\* 2 + y \*\* 2  
  
 class BFO:  
 def \_\_init\_\_(self, num\_bacteries, chemotaxis\_steps, num\_to\_eliminate,  
 elimination\_probability, x\_range, y\_range):  
 self.num\_bacteries = num\_bacteries  
 self.chemotaxis\_steps = chemotaxis\_steps  
 self.num\_to\_eliminate = num\_to\_eliminate  
 self.elimination\_probability = elimination\_probability  
 self.x\_range = x\_range  
 self.y\_range = y\_range  
  
 self.bacteries = [[random.uniform(self.x\_range[0], self.x\_range[1]),  
 random.uniform(self.y\_range[0], self.y\_range[1]),  
 0.0] for \_ in range(self.num\_bacteries)]  
 for bacteria in self.bacteries:  
 bacteria[2] = sphere(bacteria[0], bacteria[1])  
  
 self.bacteria\_best = min(self.bacteries, key=itemgetter(2))  
 self.hp = [bacteria[2] for bacteria in self.bacteries]  
  
 def next\_iteration(self):  
 for i in range(self.num\_bacteries):  
 # хемотаксис  
 for t in range(self.chemotaxis\_steps):  
 step = np.random.uniform(-1, 1)  
 new\_x = np.clip(self.bacteries[i][0] + step, self.x\_range[0], self.x\_range[1])  
 new\_y = np.clip(self.bacteries[i][1] + step, self.y\_range[0], self.y\_range[1])  
  
 new\_fitness = sphere(new\_x, new\_y)  
  
 if new\_fitness < self.bacteries[i][2]:  
 self.bacteries[i][0] = new\_x  
 self.bacteries[i][1] = new\_y  
 self.bacteries[i][2] = new\_fitness  
 # break  
  
 # репродукция  
 self.hp[i] += self.bacteries[i][2]  
  
 # Сортировка бактерий в порядке возрастания состояний здоровья  
 sorted\_indices = np.argsort(self.hp)  
 self.bacteries = [self.bacteries[i] for i in sorted\_indices]  
 self.hp = [self.hp[i] for i in sorted\_indices]  
  
 # Замена второй половины бактерий первой  
 half\_point = self.num\_bacteries // 2  
 self.bacteries[:half\_point], self.bacteries[half\_point:] = self.bacteries[  
 half\_point:], self.bacteries[  
 :half\_point]  
 self.hp[:half\_point], self.hp[half\_point:] = self.hp[half\_point:], self.hp[:half\_point]  
  
 # Ликвидация и рассеивание  
 indices\_to\_eliminate = np.random.choice(self.num\_bacteries, size=self.num\_to\_eliminate,  
 replace=False)  
 for i in indices\_to\_eliminate:  
 if np.random.rand() > self.elimination\_probability:  
 self.bacteries[i] = [random.uniform(self.x\_range[0], self.x\_range[1]),  
 random.uniform(self.y\_range[0], self.y\_range[1]),  
 0]  
 self.bacteries[i][2] = sphere(self.bacteries[i][0], self.bacteries[i][1])  
  
 self.bacteria\_best = min(self.bacteries, key=itemgetter(2))  
  
 def run\_optimization():  
  
 iter\_number = iterations\_var.get()  
 bacteries\_number = bacteries\_number\_var.get()  
 steps\_of\_chemotaxis = chemotaxis\_steps\_var.get()  
 eliminate\_number = num\_to\_eliminate\_var.get()  
 elimination\_prob = elimination\_probability\_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 = sphere(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\_title("Бактериальный алгоритм")  
 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()))  
  
  
 bfo = BFO(bacteries\_number, steps\_of\_chemotaxis, eliminate\_number, elimination\_prob,  
 [x\_interval\_min.get(), x\_interval\_max.get()], [y\_interval\_min.get(), y\_interval\_max.get()])  
  
 # отрисовка стартовой популяции  
 for bacteria in bfo.bacteries:  
 ax.scatter(bacteria[0], bacteria[1], bacteria[2], c="red", s=10)  
  
 # 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\_title("Бактериальный 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(iter\_number):  
 bfo.next\_iteration()  
 for bacteria in bfo.bacteries:  
 # отрисовка промежуточной популяции  
 ax.scatter(bacteria[0], bacteria[1], bacteria[2], c="red", s=10)  
  
 # ax.scatter(bfo.bacteria\_best[0], bfo.bacteria\_best[1], bfo.bacteria\_best[2], c="blue")  
 results\_text.insert(tk.END,  
 f"Шаг {i}: Координаты ({bfo.bacteria\_best[0]:.4f}, "  
 f"{bfo.bacteria\_best[1]:.4f}),"  
 f" Значение функции: {bfo.bacteria\_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\_title("Бактериальный алгоритм")  
 ax.plot\_surface(X, Y, Z, cmap='viridis', alpha=0.7)  
 canvas.draw()  
  
 # отрисовка результирующей популяции  
 for bacteria in bfo.bacteries:  
 ax.scatter(bacteria[0], bacteria[1], bacteria[2], c="red", s=10)  
  
 ax.scatter(bfo.bacteria\_best[0], bfo.bacteria\_best[1], bfo.bacteria\_best[2], c='black', marker='x',  
 s=60)  
  
 canvas.draw()  
 root.update()  
 results\_text.insert(tk.END,  
 f"Результат:\nКоординаты ({bfo.bacteria\_best[0]:.5f}, "  
 f"{bfo.bacteria\_best[1]:.5f}),\nЗначение функции: {bfo.bacteria\_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="Количество итераций", font=("Helvetica", 10)).grid(row=1, column=0)  
 ttk.Label(param\_frame2, text="Количество бактерий", font=("Helvetica", 10)).grid(row=2, column=0)  
 ttk.Label(param\_frame2, text="Шагов хемотаксиса", font=("Helvetica", 10)).grid(row=3, column=0)  
 ttk.Label(param\_frame2, text="Количество ликвидируемых", font=("Helvetica", 10)).grid(row=4, column=0)  
 ttk.Label(param\_frame2, text="Вероятность ликвидации", font=("Helvetica", 10)).grid(row=5, column=0)  
 ttk.Label(param\_frame2, text="Задержка", font=("Helvetica", 10)).grid(row=6, column=0)  
  
 iterations\_var = tk.IntVar(value=50)  
 bacteries\_number\_var = tk.IntVar(value=50)  
 chemotaxis\_steps\_var = tk.IntVar(value=15)  
 num\_to\_eliminate\_var = tk.IntVar(value=20)  
 elimination\_probability\_var = tk.DoubleVar(value=0.6)  
 delay\_var = tk.DoubleVar(value=0.01)  
  
 iterations\_entry = ttk.Entry(param\_frame2, textvariable=iterations\_var)  
 bacteries\_number\_entry = ttk.Entry(param\_frame2, textvariable=bacteries\_number\_var)  
 chemotaxis\_steps\_entry = ttk.Entry(param\_frame2, textvariable=chemotaxis\_steps\_var)  
 num\_to\_eliminate\_entry = ttk.Entry(param\_frame2, textvariable=num\_to\_eliminate\_var)  
 elimination\_probability\_entry = ttk.Entry(param\_frame2, textvariable=elimination\_probability\_var)  
 delay\_entry = ttk.Entry(param\_frame2, textvariable=delay\_var)  
  
 iterations\_entry.grid(row=1, column=1)  
 bacteries\_number\_entry.grid(row=2, column=1)  
 chemotaxis\_steps\_entry.grid(row=3, column=1)  
 num\_to\_eliminate\_entry.grid(row=4, column=1)  
 elimination\_probability\_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 = ["Функция сферы"]  
 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()