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 36 группы

Агаджанян А.С.

Гаранина Л.В.

Преподаватель:

Нигодин Е.А.

Краснодар 2025

**Цель работы:** разработать алгоритм бактериальной оптимизации обратной сферической функции, имеющей вид f(.

**Ход работы**

Поведение бактерий обусловлено механизмом, который называется бактериальным хемотаксисом и представляет собой двигательную реакцию микроорганизмов на химический раздражитель. Данный механизм позволяет бактерии двигаться по направлению к аттрактантам (питательным веществам) и от репеллентов (потенциально вредных для бактерии веществ).

**Канонический алгоритм бактериальной оптимизации**

Канонический алгоритм бактериальной оптимизации основан на использовании трех основных механизмов: хемотаксис, репродукция, ликвидация и рассеивание.

Хемотаксис – процесс направленного движения бактерии в ответ на химические раздражители, поиск лучшего положения в пространстве. Каждая бактерия выбирает направление движения случайным образом. Она делает шаг в этом направлении и оценивает, стало ли значение функции лучше. Если стало лучше – она продолжает двигаться дальше в том же направлении еще несколько шагов (плавание). Как только перестает улучшаться – бактерия останавливается и ждет следующего хемотактического цикла. То есть, хемотаксис позволяет бактериям локально исследовать пространство решений и двигаться к максимуму/минимуму функции.

Репродукция – этап, на котором "здоровые" бактерии размножаются, а "слабые" удаляются. Здоровые бактерии – суммарное значение функции за все предыдущие хемотактические циклы. Бактерии сортируются по здоровью, половина лучших бактерий остается, и их копии заменяют худших. Это необходимо, чтобы популяция со временем сосредоточилась вокруг лучших решений и ускорила сходимость алгоритма.

Ликвидация и рассеивание (элиминация-дисперсия) – этап, который помогает избежать застревания в локальном оптимуме. Когда бактерии "засиделись" в каком-то месте и перестают находить лучшие решения, какой-то процент их удаляется (ликвидируется), а на их место создаются новые случайные бактерии (рассеивание), чтобы заново исследовать территории.

**Особенности реализации бактериального алгоритма.**

Для создания программы использовался язык программирования Python. Для графической визуализации были подключены графические фреймворки Tkinter и Matplotlib.

Интерфейс программы имеет вид:

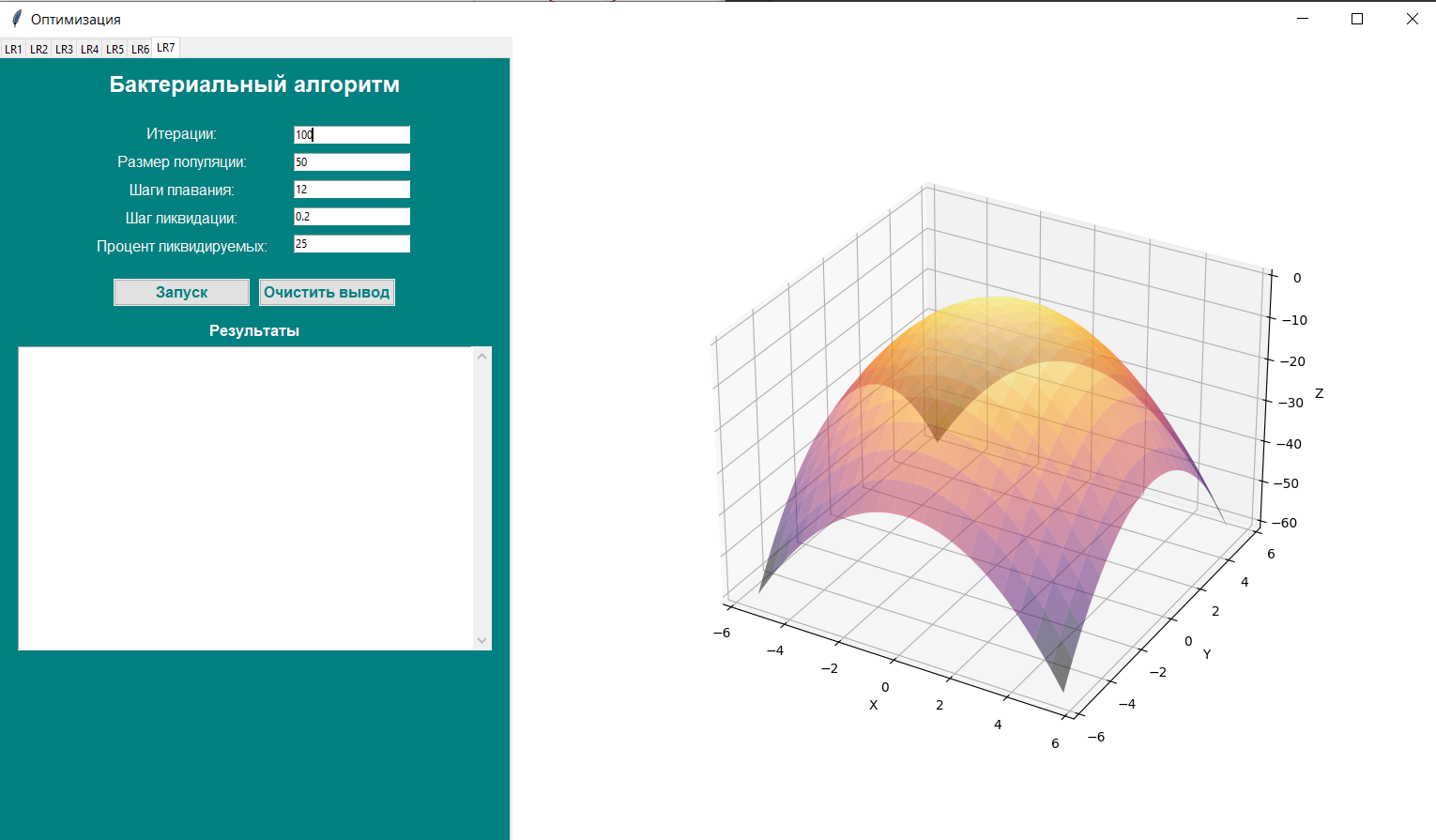


Рисунок 1 – Главное окно программы

В данном окне можно выбрать необходимую вкладку соответствующей лабораторной работы, в данном случае «7», ввести количество итераций, размер популяции, шаги плавания, шаг ликвидации и процент ликвидируемых. В поле «Результаты» динамически выводятся результаты работы алгоритма в виде шагов, представленных координатами и значениями оптимизируемой функции в этих координатах.

На рисунке 2 показана основная панель.

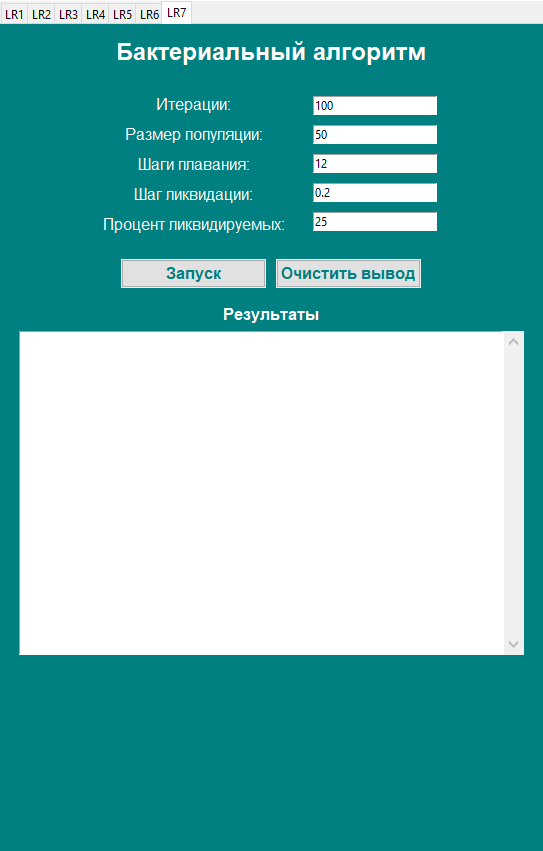


Рисунок 2 – Основная панель

Для запуска алгоритма необходимо нажать кнопку «Запуск», при этом поле «Результаты» начнёт динамически заполняться данными, а на отображаемой функции можно наглядно увидеть функционирование алгоритма - искомые точки, которые также отображаются динамически как показано на рисунке 3. Результирующая точка выделена красным цветом.

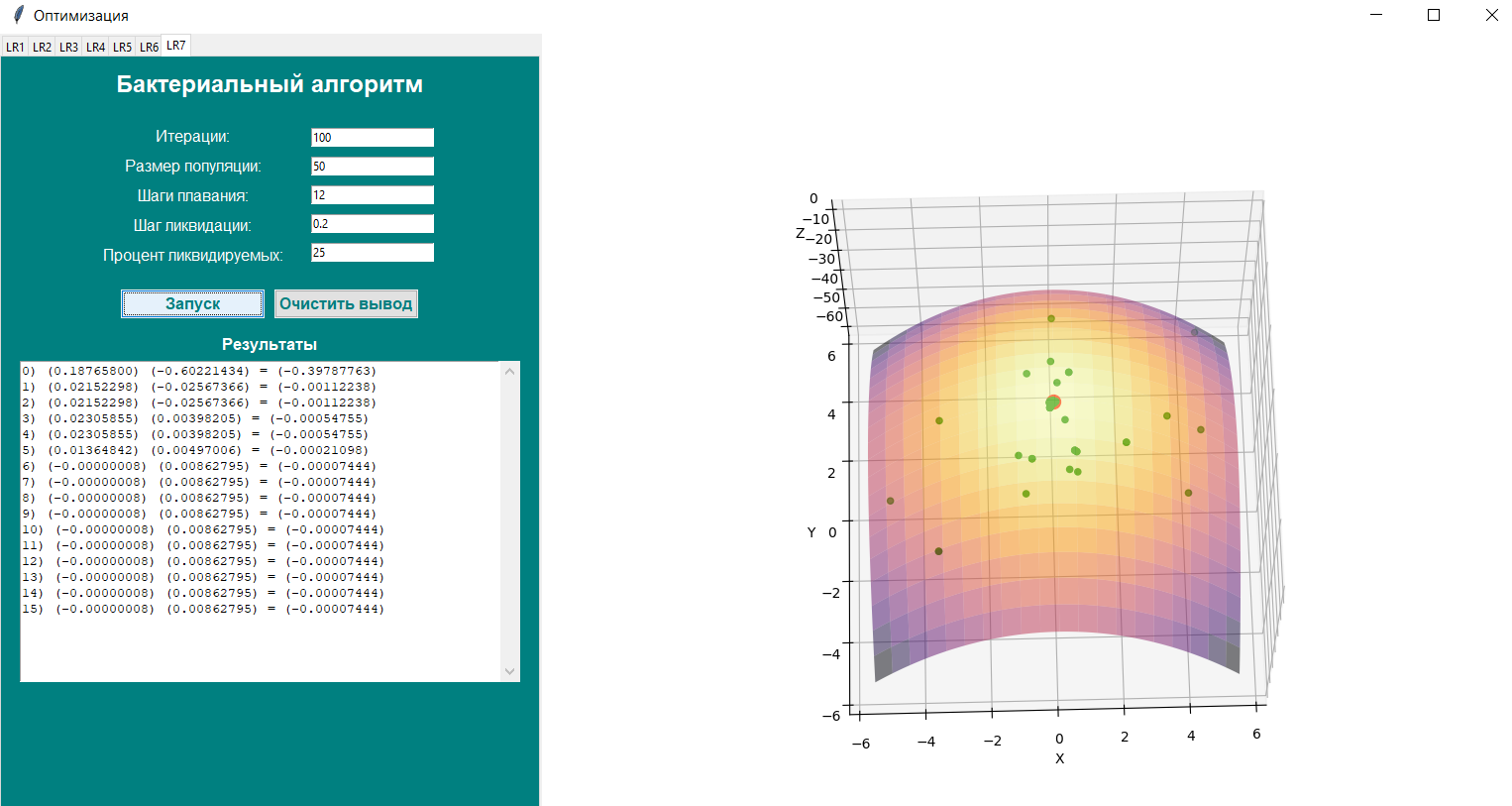


Рисунок 3 – Результат работы программы.

**Вывод:** в ходе работы изучили иреализовали бактериальный алгоритм для нахождения глобального максимума обратной функции сферы.

**Листинг программы**

Файл bacteria.py

import numpy as np

def bacteria\_foraging\_optimization(n, function, iteration, num\_swimming\_steps, step\_size, dispersion\_probability, Nc=2, lb=-5, ub=5):

dispersion\_probability /= 100 # Преобразуем проценты в дробную величину

agents = np.random.uniform(lb, ub, (n, 2)) # Инициализация агентов случайно

all\_positions = [agents.copy()] # Список для хранения всех позиций агентов

fitness\_history = [] # История значений фитнес-функции

C\_list = [step\_size - step\_size \* 0.9 \* t / iteration for t in range(iteration)]

gbest\_position = agents[0].copy()

gbest\_value = function(gbest\_position)

for t in range(iteration):

step\_size = C\_list[t]

J\_chem = [] # История фитнес-функций на данном шаге

for \_ in range(Nc): # Nc хемотактических шагов

for i in range(n):

direction = np.random.uniform(-1, 1, 2)

direction /= np.linalg.norm(direction)

current\_pos = agents[i].copy()

current\_val = function(current\_pos)

new\_pos = current\_pos + step\_size \* direction

new\_val = function(new\_pos)

if new\_val > current\_val:

agents[i] = new\_pos

current\_val = new\_val

for \_ in range(1, num\_swimming\_steps):

next\_pos = agents[i] + step\_size \* direction

next\_val = function(next\_pos)

if next\_val > current\_val:

agents[i] = next\_pos

current\_val = next\_val

else:

break

if current\_val > gbest\_value:

gbest\_value = current\_val

gbest\_position = agents[i].copy()

J\_chem.append(np.array([function(agent) for agent in agents]))

fitness\_history.append(J\_chem[-1])

health = [(sum(fitness\_history[k][i] for k in range(len(fitness\_history))), i) for i in range(n)]

health.sort(reverse=True

half = n // 2

survivors = [agents[i].copy() for (\_, i) in health[:half]]

agents = np.array(survivors \* 2)

for i in range(n):

if np.random.rand() < dispersion\_probability:

agents[i] = np.random.uniform(lb, ub, 2)

agents[0] = gbest\_position.copy()

all\_positions.append(agents.copy())

return all\_positions

Файл main.py

def draw\_bfo():

txt\_f\_tab\_7.delete("1.0", tk.END)

pop\_number = int(size7.get()) # количество агентов

iter\_number = int(it7.get()) # число итераций

swim\_steps = int(swim7.get()) # шаги плавания

elim\_step = float(elim7.get()) # шаг ликвидации

elim\_num = int(elim\_num7.get()) # процент ликвидируемых

func = inverse\_sphere

if pop\_number % 2 != 0:

messagebox.showerror("Ошибка", "Количество агентов должно быть чётным!")

return

ax.cla()

plot\_surface(ax, "Обратная функция Сферы")

all\_positions = bacteria\_foraging\_optimization(

n=pop\_number,

function=func,

iteration=iter\_number,

num\_swimming\_steps=swim\_steps,

step\_size=elim\_step,

dispersion\_probability=elim\_num

)

for t, agents in enumerate(all\_positions):

z\_vals = [func(x) for x in agents]

ax.cla()

plot\_surface(ax, "Обратная функция Сферы")

ax.scatter([x[0] for x in agents],

[x[1] for x in agents],

z\_vals,

c="green", s=20, marker="o")

best\_idx = np.argmax(z\_vals)

best\_agent = agents[best\_idx]

best\_val = z\_vals[best\_idx]

ax.scatter(best\_agent[0], best\_agent[1], best\_val, c="red", s=100, marker="o")

txt\_f\_tab\_7.insert(tk.END,

f"{t}) ({best\_agent[0]:.8f}) "

f"({best\_agent[1]:.8f}) = "

f"({best\_val:.8f})\n")

txt\_f\_tab\_7.yview\_moveto(1)

canvas.draw()

window.update()

ax.scatter(best\_agent[0], best\_agent[1], best\_val, c="red", s=40, marker="x")

txt\_f\_tab\_7.insert(tk.END, f"\nЛучшее найденное решение: \n({best\_agent[0]:.8f}) ({best\_agent[1]:.8f}) = ({best\_val:.8f})")

txt\_f\_tab\_7.yview\_moveto(1)

canvas.draw()

window.update()

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

# Бактериальный алгоритм

tab7 = tk.Frame(tab\_control, bg="#008080")

tab\_control.add(tab7, text="LR7")

label\_title71 = tk.Label(tab7, text="Бактериальный алгоритм", font=("Arial", 18, "bold"), fg="#FFFFFF", bg="#008080")

label\_title71.pack(pady=10)

input\_frame7 = tk.Frame(tab7, bg="#008080")

input\_frame7.pack(pady=10)

labels\_frame7 = tk.Frame(input\_frame7, bg="#008080")

labels\_frame7.pack(side=tk.LEFT, padx=10)

entries\_frame7 = tk.Frame(input\_frame7, bg="#008080")

entries\_frame7.pack(side=tk.LEFT, padx=10)

label\_iterations7 = tk.Label(labels\_frame7, text="Итерации:", bg="#008080", font=("Arial", 12), fg="#FFFFFF")

label\_iterations7.pack(pady=2.5)

it7 = tk.Entry(entries\_frame7)

it7.insert(0, "100")

it7.pack(padx=5, pady=5)

label\_size7 = tk.Label(labels\_frame7, text="Размер популяции:", bg="#008080", font=("Arial", 12), fg="#FFFFFF")

label\_size7.pack(pady=2.5)

size7 = tk.Entry(entries\_frame7)

size7.insert(0, "50")

size7.pack(padx=5, pady=5)

label\_swim7 = tk.Label(labels\_frame7, text="Шаги плавания:", bg="#008080", font=("Arial", 12), fg="#FFFFFF")

label\_swim7.pack(pady=2.5)

swim7 = tk.Entry(entries\_frame7)

swim7.insert(0, "12")

swim7.pack(padx=5, pady=5)

label\_elim7 = tk.Label(labels\_frame7, text="Шаг ликвидации:", bg="#008080", font=("Arial", 12), fg="#FFFFFF")

label\_elim7.pack(pady=2.5)

elim7 = tk.Entry(entries\_frame7)

elim7.insert(0, "0.2")

elim7.pack(padx=5, pady=5)

label\_elim\_num7 = tk.Label(labels\_frame7, text="Процент ликвидируемых:", bg="#008080", font=("Arial", 12), fg="#FFFFFF")

label\_elim\_num7.pack(pady=2.5)

elim\_num7 = tk.Entry(entries\_frame7)

elim\_num7.insert(0, "25")

elim\_num7.pack(padx=5, pady=5)

button\_bact = tk.Frame(tab7, bg="#008080")

button\_bact.pack(pady=10)

ttk.Button(button\_bact, text="Запуск", style="Custom.TButton", command=draw\_bfo).pack(side="left", padx=5)

ttk.Button(button\_bact, text="Очистить вывод", style="Custom.TButton", command=lambda: txt\_f\_tab\_7.delete(1.0, tk.END)).pack(side="left", padx=5)

ttk.Label(tab7, text="Результаты", font=("Helvetica", 12, "bold"), background="#008080", foreground="#FFFFFF").pack(pady=5)

txt\_f\_tab\_7 = scrolledtext.ScrolledText(tab7, width=60, height=20)

txt\_f\_tab\_7.pack()

canvas.draw()

window.mainloop()