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 36 группы

Агаджанян А.С.

Гаранина Л.В.

Преподаватель:

Нигодин Е.А.

Краснодар 2025

**Цель работы:** разработать гибридный алгоритм оптимизации функции. В качестве гибридизируемых алгоритмов использовать любые два из раннее разработанных алгоритмов.

**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 не всегда приводит к успеху. Поэтому одним из основных путей повышения эффективности решения задач глобального поиска является разработка гибридных популяционных алгоритмов.

В соответствии с классификацией Ванга выделяют 3 категории гибридных алгоритмов: вложенные, типа препроцессор/постпроцессор и коалгоритмы.

В категории гибридизации вложением выделяют высокоуровневую и низкоуровневую гибридизацию. Высокоуровневая гибридизация предполагает слабую связь объединяемых алгоритмов, так что легко выделить каждый из используемых алгоритмов. При низкоуровневой гибридизации комбинируемые алгоритмы интегрированы настолько сильно, что выделить составляющие итогового алгоритма обычно невозможно. То есть, порождается по сути новый алгоритм.

В гибридизации типа препроцессор/постпроцессор используются 2 алгоритма, функционирующих последовательно один за другим. Выделяют 2 класса методов: последовательная и конвейерная. Последовательная предполагает фиксированный порядок использования двух алгоритмов, хотя момент времени переключения с одного на другой может априори не фиксироваться. 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.

Коалгоритмическая гибридизация предполагает, что объединяемые алгоритмы, как минимум, на логическом уровне выполняются параллельно, обмениваясь между собой в процессе поиска информацией.

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

**Особенности реализации гибридного алгоритма.**

Для создания программы использовался язык программирования Python. Для графической визуализации были подключены графические фреймворки Tkinter и Matplotlib. В реализуемом гибридном алгоритме используются иммунный и бактериальный алгоритмы.

Интерфейс программы имеет вид:

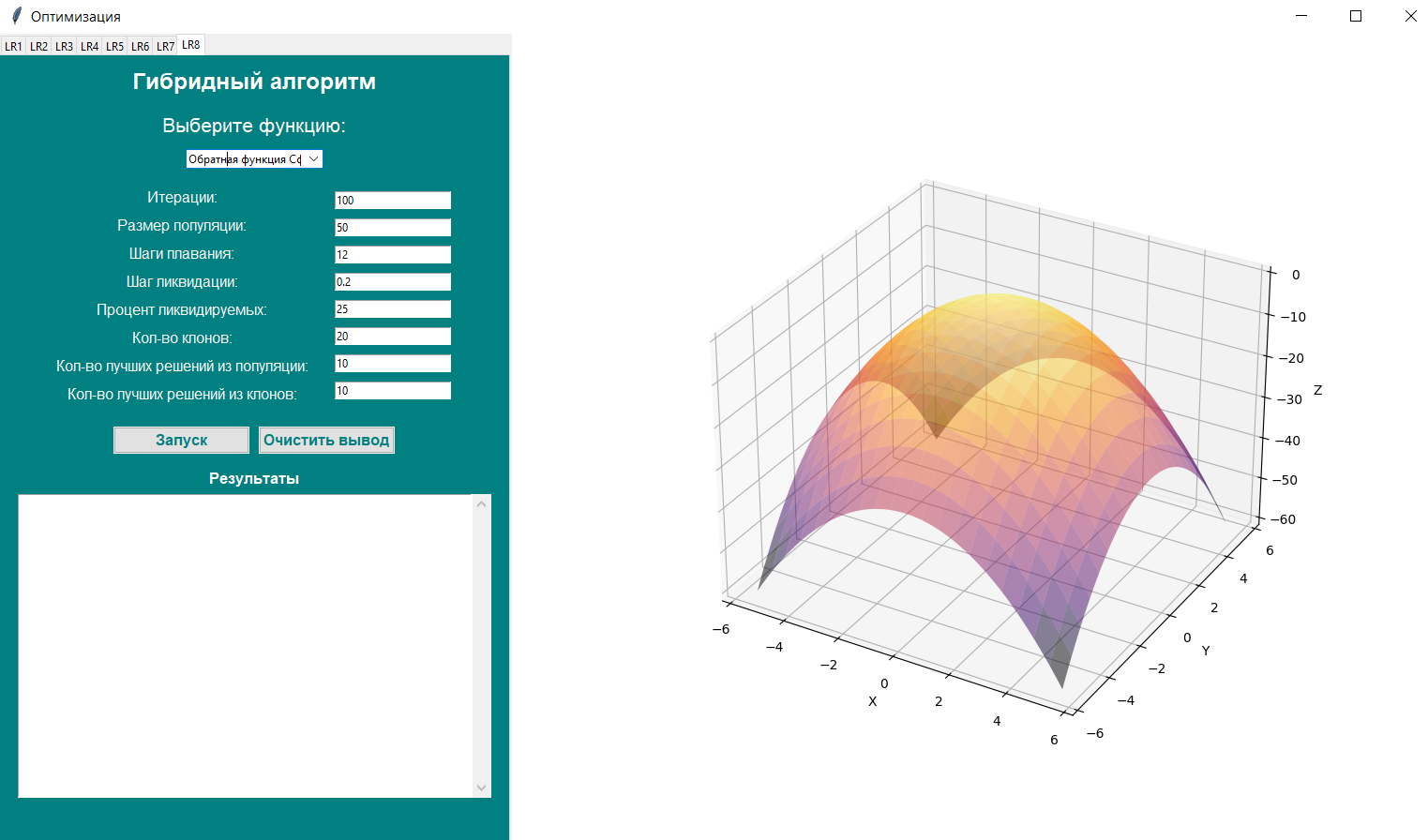


Рисунок 1 – Главное окно программы

В данном окне можно выбрать необходимую вкладку соответствующей лабораторной работы, в данном случае «8», и ввести данные из иммунного (количество клонов, количество лучших решений из популяции, количество лучших решений из клонов) и бактериального (шаги плавания, шаг ликвидации, процент ликвидируемых) алгоритмов, а так же размер популяции и количество итераций. 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.

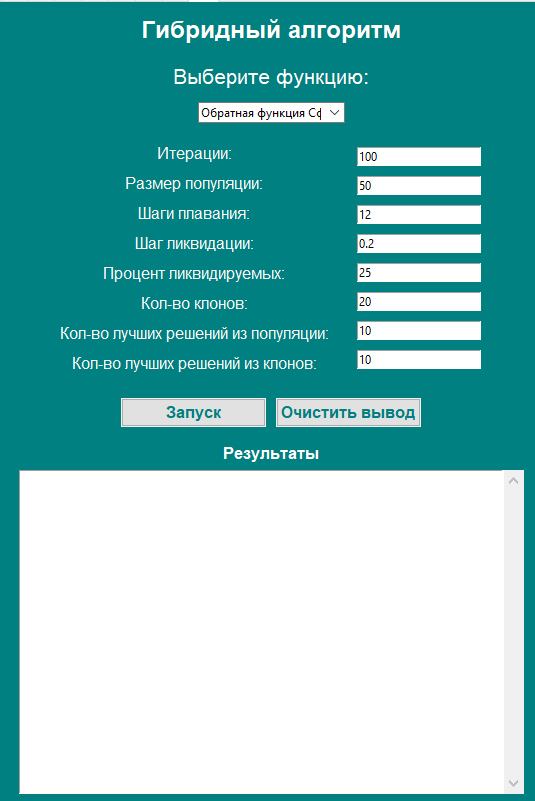


Рисунок 2 – Основная панель

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.

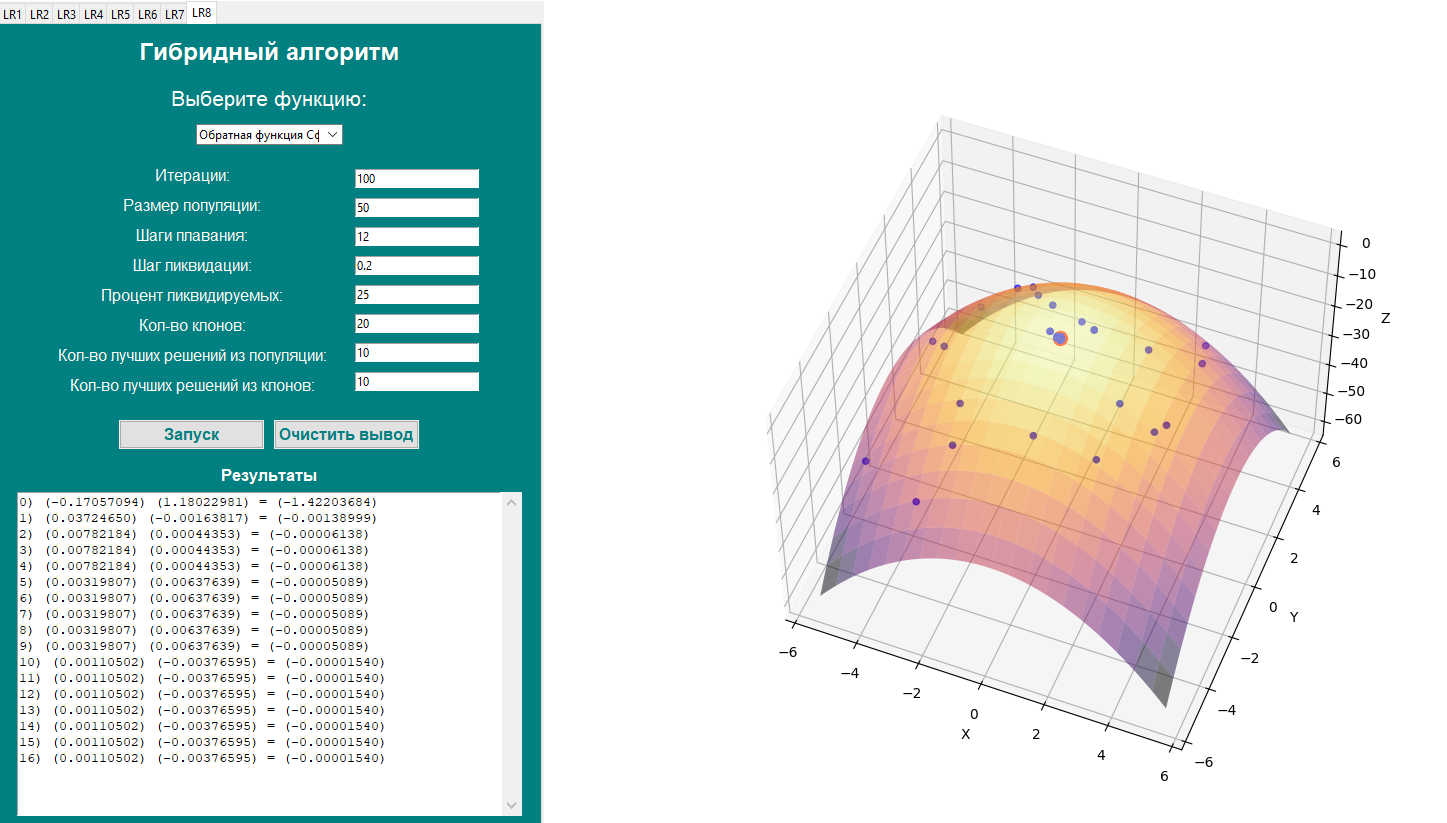


Рисунок 3 – Результат работы программы.

В реализуемом гибридном алгоритме бактериальный алгоритм применяется для глобального поиска, а иммунный – для локального поиска и улучшения решений. В данном случае бактериальный алгоритм случайно перемещает агенты по всему пространству, исследуя в разных направлениях и находя потенциально хорошие области (хемотаксис, плавание). Иммунный алгоритм действует как локальный поиск, так как клонирует и модифицирует лучших агентов в пределах ограниченного пространства вокруг найденных решений, улучшая их точность и сходясь к экстремуму (оценка, селекция, клонирование, мутация, отбор). После происходит репродукция (удвоение лучших) и случайная дисперсия (разброс).

То есть, по классификации Ванга, гибридный алгоритм является вложенным низкоуровневым, так как иммунный алгоритм встраивается в механизм бактериального и работает внутри каждой итерации, не управляя всей стратегией поиска (реализация локального модификатора).

По классификации Рейдла, реализована интегративная стратегия гибридизации, так как один алгоритм встраивается внутрь другого и участвует в самом процессе поиска – не отдельно, не поэтапно, а как часть механизма.

**Вывод:** в ходе работы был реализован гибридный алгоритм из иммунного и бактериального для нахождения экстремума функции.

**Листинг программы**

Файл hybrid.py

import numpy as np

from immune import immune\_step

def hybrid\_algorithm(

func, n, iterations, swim\_steps, step\_size, dispersion\_prob,

best\_numb, clon\_numb, best\_clon\_numb, bounds, maximize=False, Nc=2

):

lb, ub = bounds[0][0], bounds[0][1]

dispersion\_prob /= 100

agents = np.random.uniform(lb, ub, (n, 2))

all\_positions = [agents.copy()]

fitness\_history = []

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

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

gbest\_value = func(gbest\_position)

def is\_better(a, b):

return a > b if maximize else a < b

for t in range(iterations):

step\_size = C\_list[t]

J\_chem = []

for \_ in range(Nc):

for i in range(n):

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

direction /= np.linalg.norm(direction)

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

current\_val = func(current\_pos)

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

new\_val = func(new\_pos)

if is\_better(new\_val, current\_val):

agents[i] = new\_pos

current\_val = new\_val

for \_ in range(1, swim\_steps):

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

next\_val = func(next\_pos)

if is\_better(next\_val, current\_val):

agents[i] = next\_pos

current\_val = next\_val

else:

break

if is\_better(current\_val, gbest\_value):

gbest\_value = current\_val

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

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

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

# Иммунная фаза

evaluated = [[agent[0], agent[1], func(agent)] for agent in agents]

evaluated = immune\_step(

evaluated, func, best\_numb, clon\_numb, best\_clon\_numb,

coef=step\_size, bounds=[(lb, ub), (lb, ub)], rev=maximize

)

agents = np.array([[x, y] for x, y, \_ in evaluated])

# Репродукция

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

health.sort(reverse=maximize)

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\_prob:

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

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

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

return all\_positions, gbest\_position

Файл immune.py

def immune\_step(agents, func, best\_numb, clon\_numb, best\_clon\_numb, coef, bounds, rev=False):

best = sorted(agents, key=itemgetter(2), reverse=rev)[:best\_numb]

clones = []

for agent in best:

for \_ in range(clon\_numb):

x = agent[0] + coef \* random.uniform(-1, 1)

y = agent[1] + coef \* random.uniform(-1, 1)

# Ограничиваем значения в пределах границ

x = max(bounds[0][0], min(bounds[0][1], x))

y = max(bounds[1][0], min(bounds[1][1], y))

z = func([x, y])

clones.append([x, y, z])

top\_clones = sorted(clones, key=itemgetter(2), reverse=rev)[:best\_clon\_numb]

new\_population = agents + top\_clones

new\_population = sorted(new\_population, key=itemgetter(2), reverse=rev)[:len(agents)]

return new\_population

Файл main.py

def draw\_hybrid():

txt\_f\_tab\_8.delete("1.0", tk.END)

pop\_number = int(size8.get()) # количество агентов

iter\_number = int(it8.get()) # число итераций

swim\_steps = int(swim8.get()) # шаги плавания

elim\_step = float(elim8.get()) # шаг ликвидации

elim\_num = int(elim\_num8.get()) # процент ликвидируемых

selected\_func = combobox\_functions8.get() # используемая функция

# === Параметры иммунного алгоритма ===

best\_numb = int(best\_numb8.get())

clon\_numb = int(clon\_numb8.get())

best\_clon\_numb = int(best\_clon\_numb8.get())

if pop\_number % 2 != 0:

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

return

if selected\_func == "Функция Растригина":

func = rastrigin

maxi = False

elif selected\_func == "Обратная функция Сферы":

func = inverse\_sphere

maxi = True

ax.cla()

plot\_surface(ax, selected\_func)

bounds = [(-5, 5), (-5, 5)]

all\_positions, best\_agent = hybrid\_algorithm(

func=func,

n=pop\_number,

iterations=iter\_number,

swim\_steps=swim\_steps,

step\_size=elim\_step,

dispersion\_prob=elim\_num,

best\_numb=best\_numb,

clon\_numb=clon\_numb,

best\_clon\_numb=best\_clon\_numb,

bounds=bounds,

maximize=maxi

)

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

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

ax.cla()

plot\_surface(ax, selected\_func)

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

[x[1] for x in agents],

z\_vals,

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

if (maxi):

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

else:

best\_idx = np.argmin(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\_8.insert(tk.END,

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

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

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

txt\_f\_tab\_8.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\_8.insert(tk.END, f"\nЛучшее найденное решение: \n({best\_agent[0]:.8f}) ({best\_agent[1]:.8f}) = ({best\_val:.8f})")

txt\_f\_tab\_8.yview\_moveto(1)

canvas.draw()

window.update()

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

# Гибридный алгоритм

tab8 = tk.Frame(tab\_control, bg="#008080")

tab\_control.add(tab8, text="LR8")

label\_title81 = tk.Label(tab8, text="Гибридный алгоритм", font=("Arial", 18, "bold"), fg="#FFFFFF", bg="#008080")

label\_title81.pack(pady=10)

label\_function8 = tk.Label(tab8, text="Выберите функцию:", bg="#008080", font=("Arial", 16), fg="#FFFFFF")

label\_function8.pack(pady=5)

combobox\_functions8 = Combobox(tab8, values=["Обратная функция Сферы", "Функция Растригина"])

combobox\_functions8.pack(pady=5)

combobox\_functions8.set("Обратная функция Сферы")

combobox\_functions8.bind("<<ComboboxSelected>>", update\_plot)

input\_frame8 = tk.Frame(tab8, bg="#008080")

input\_frame8.pack(pady=10)

# Фрейм слева для лейблов

labels\_frame8 = tk.Frame(input\_frame8, bg="#008080")

labels\_frame8.pack(side=tk.LEFT, padx=10)

# Фрейм справа для полей ввода

entries\_frame8 = tk.Frame(input\_frame8, bg="#008080")

entries\_frame8.pack(side=tk.LEFT, padx=10)

# Поля ввода параметров

label\_iterations8 = tk.Label(labels\_frame8, text="Итерации:", bg="#008080", font=("Arial", 12), fg="#FFFFFF")

label\_iterations8.pack(pady=2.5)

it8 = tk.Entry(entries\_frame8)

it8.insert(0, "100")

it8.pack(padx=5, pady=5)

label\_size8 = tk.Label(labels\_frame8, text="Размер популяции:", bg="#008080", font=("Arial", 12), fg="#FFFFFF")

label\_size8.pack(pady=2.5)

size8 = tk.Entry(entries\_frame8)

size8.insert(0, "50")

size8.pack(padx=5, pady=5)

label\_swim8 = tk.Label(labels\_frame8, text="Шаги плавания:", bg="#008080", font=("Arial", 12), fg="#FFFFFF")

label\_swim8.pack(pady=2.5)

swim8 = tk.Entry(entries\_frame8)

swim8.insert(0, "12")

swim8.pack(padx=5, pady=5)

label\_elim8 = tk.Label(labels\_frame8, text="Шаг ликвидации:", bg="#008080", font=("Arial", 12), fg="#FFFFFF")

label\_elim8.pack(pady=2.5)

elim8 = tk.Entry(entries\_frame8)

elim8.insert(0, "0.2")

elim8.pack(padx=5, pady=5)

label\_elim\_num8 = tk.Label(labels\_frame8, text="Процент ликвидируемых:", bg="#008080", font=("Arial", 12), fg="#FFFFFF")

label\_elim\_num8.pack(pady=2.5)

elim\_num8 = tk.Entry(entries\_frame8)

elim\_num8.insert(0, "25")

elim\_num8.pack(padx=5, pady=5)

label\_83 = tk.Label(labels\_frame8, text="Кол-во клонов:", bg="#008080", font=("Arial", 12), fg="#FFFFFF")

label\_83.pack(pady=2.5)

clon\_numb8 = tk.Entry(entries\_frame8)

clon\_numb8.insert(0, "20")

clon\_numb8.pack(padx=5, pady=5)

label\_84 = tk.Label(labels\_frame8, text="Кол-во лучших решений из популяции:", bg="#008080", font=("Arial", 12), fg="#FFFFFF")

label\_84.pack(pady=2.5)

best\_numb8 = tk.Entry(entries\_frame8)

best\_numb8.insert(0, "10")

best\_numb8.pack(padx=5, pady=5)

label\_85 = tk.Label(labels\_frame8, text="Кол-во лучших решений из клонов:", bg="#008080", font=("Arial", 12), fg="#FFFFFF")

label\_85.pack(pady=2.5)

best\_clon\_numb8 = tk.Entry(entries\_frame8)

best\_clon\_numb8.insert(0, "10")

best\_clon\_numb8.pack(padx=5, pady=5)

button\_hybrid = tk.Frame(tab8, bg="#008080")

button\_hybrid.pack(pady=10)

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

ttk.Button(button\_hybrid, text="Очистить вывод", style="Custom.TButton", command=lambda: txt\_f\_tab\_8.delete(1.0, tk.END)).pack(side="left", padx=5)

ttk.Label(tab8, text="Результаты", font=("Helvetica", 12, "bold"), background="#008080", foreground="#FFFFFF").pack(pady=5)

txt\_f\_tab\_8 = scrolledtext.ScrolledText(tab8, width=60, height=20)

txt\_f\_tab\_8.pack()

canvas.draw()

window.mainloop()