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 №3 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 генетический алгоритм оптимизации функции Розенброкка.

**Ход работы:**

Генетический алгоритм (ГА) — это метаэвристический метод оптимизации, инспирированный процессами естественного отбора и генетики. Этот алгоритм используется для решения задачи оптимизации и поиска, особенно в ситуациях, где простое и полный переборное решение невозможно или неэффективно. Генетические алгоритмы часто применяются для нахождения приближенных решений в множестве возможных вариантов.

Рассмотрим ключевые элементы генетического алгоритма и его работы:

1. **Популяция**: генетический алгоритм начинается с создания начальной популяции. Каждый элемент популяции представляет собой потенциальное решение задачи и называется "особью". Эти особи могут быть представлены в виде строк, векторов или других структур данных, в зависимости от конкретной задачи.
2. **Фитнес-функция**: для оценки качества каждой особи в популяции определяется фитнес-функция. Фитнес-функция принимает особь в качестве входных данных и возвращает численное значение, которое характеризует "подходит ли" эта особь для решения задачи. Цель состоит в том, чтобы максимизировать или минимизировать значение этой функции, в зависимости от задачи оптимизации.
3. **Селекция**: особи с более высокими значениями фитнес-функции имеют больший шанс выжить и передать свои гены следующему поколению. Это имитирует естественный отбор, где успешные особи имеют больше потомков. Селекция может быть случайной, но вероятность выбора особей зависит от их фитнес-значения.
4. **Скрещивание (кроссовер)**:выбранные особи скрещиваются, и их гены комбинируются, чтобы создать потомство. Как именно это происходит, зависит от метода скрещивания, который может варьироваться от простой одноточечной кроссовер до более сложных методов. Цель скрещивания - внести разнообразие в популяцию и, возможно, сочетать положительные черты разных особей.
5. **Мутация**: некоторые гены в потомстве могут случайным образом мутировать. Мутация вносит случайные изменения в гены особей и помогает сохранить разнообразие в популяции.
6. **Замена поколения**: новое поколение особей формируется путем комбинирования селекции, скрещивания и мутации. Часть старой популяции заменяется новой на основе их фитнес-значений.
7. **Критерий остановки**: генетический алгоритм продолжает создавать новые поколения и улучшать популяцию до тех пор, пока не выполнится определенный критерий остановки. Критерии могут включать ограничение по количеству поколений, достижение желаемого значения фитнес-функции или истечение времени.

Теперь рассмотрим подробнее реализацию генетического алгоритма на функции Розенброка:

1. **Инициализация популяции**: в начале работы алгоритма создается начальная популяция из случайных особей. Каждая особь представляется двумя генами (x, y), и ей назначается значение фитнес-функции (z), которое рассчитывается на основе значений x и y, используя вашу оптимизируемую функцию. Наша цель - найти особь (x, y), для которой значение z минимальное.
2. **Эволюция популяции:**

* *Селекция*: особи в текущей популяции ранжируются в порядке их значений фитнес-функции. Лучшие особи (те, у которых z ближе к оптимуму) имеют большие шансы быть выбранными как родители. Количество особей, которые будут выбраны в качестве родителей, определяется коэффициентом выживаемости (survive\_cof).
* *Репродукция* (скрещивание): выбранные родители принимают участие в создании потомства. Для каждой новой особи, x и y наследуются от родителей, и значение z пересчитывается на основе новых x и y. Вероятность того, что x или y будут заменены значениями другого родителя, зависит от случайного числа. Это процесс повторяется для нескольких новых особей.
* *Мутация*: каждая особь в популяции имеет шанс быть подверженной мутации. Это означает, что x и/или y могут быть незначительно изменены случайным образом. Вероятность мутации определяется параметром mut\_chance.

1. **Итерации**: эти шаги (селекция, скрещивание и мутация) повторяются в течение заданного количества поколений (generations). В каждом поколении лучшие особи сохраняются, а менее успешные могут быть заменены новыми особями.
2. **Завершение и выбор лучшей особи**: после завершения всех итераций, алгоритм возвращает особь с наилучшим значением z, то есть особь с минимальным значением z.

**Особенности реализации генетического алгоритма.**

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

Интерфейс программы имеет вид:

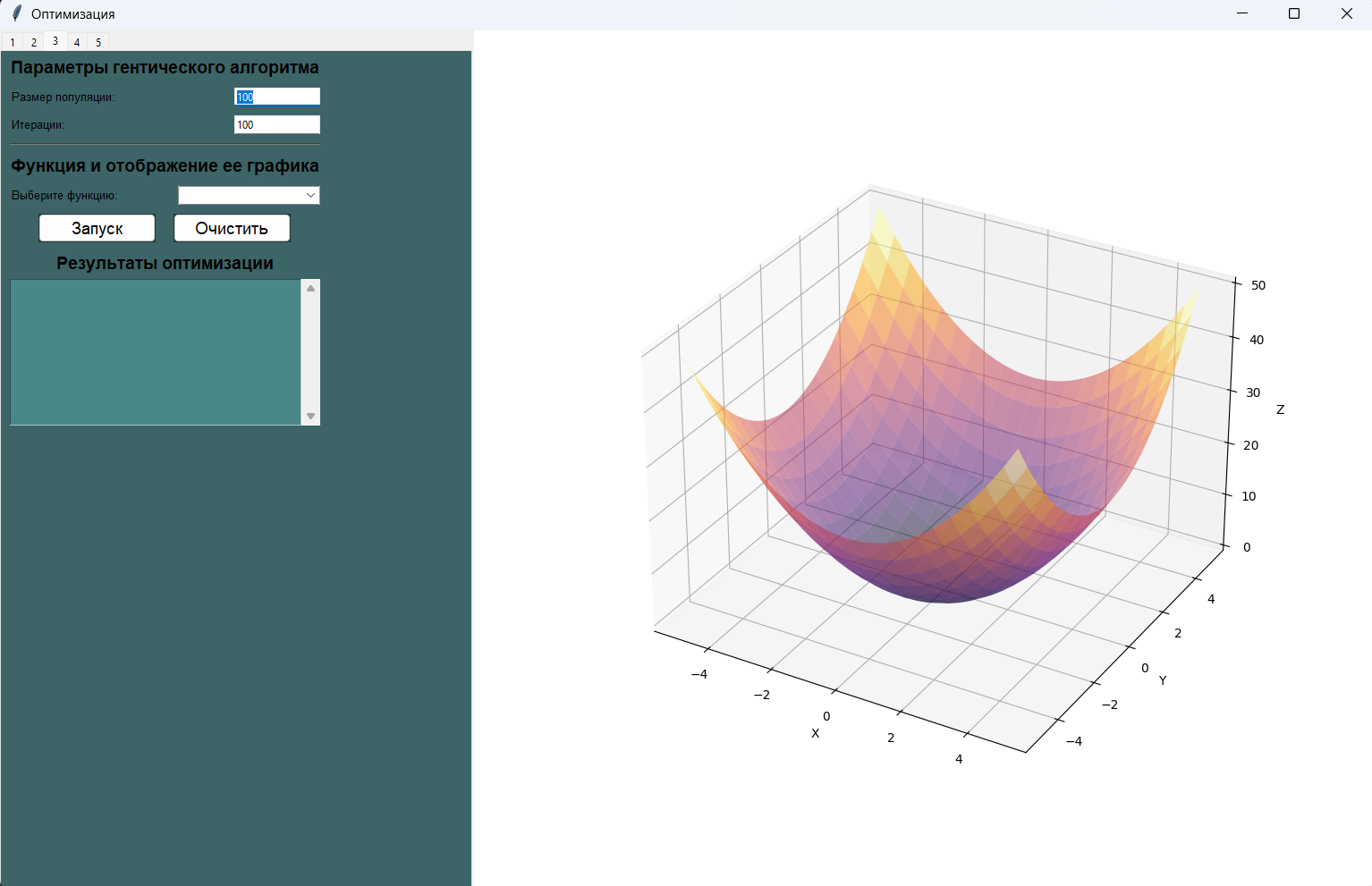


Рисунок 1 – Главное окно программы.

В данном окне можно выбрать необходимую вкладку соответствующей лабораторной работы, в данном случае «3», ввести количество особей для операций и количество итераций.

В поле «Запуск» динамически выводятся результаты работы алгоритма в виде шагов, представленных координатами и значениями оптимизируемой функции в этих координатах. На рисунке 2 показано поле «Результаты оптимизации» до запуска программы.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2 - Результаты оптимизации.

В меню можно выбрать функцию, задать значения размера популяции, количество итераций. На рисунке 3 показано основное меню для ввода параметров.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3 - Окно ввода значений.

Для запуска алгоритма необходимо нажать кнопку «Запуск», при этом поле «Результаты оптимизации» 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. Чтобы очистить поле вывода результатов, нажмите кнопку «Очистить»

Изображение выглядит как текст, диаграмма, снимок экрана, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 4 – Результат работы программы.

**Вывод:** в ходе работы были изучен, а затем разработан генетический алгоритм оптимизации функции Розенброкка.

**Листинг программы**

Файл genetic\_algorithm:

import tkinter as tk

from tkinter import ttk

from tkinter import scrolledtext

import numpy as np

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

    # Функция Розенброка для оптимизации

    def rosenbrock\_function(x, y):

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

    # Оператор селекции (выбор лучших особей)

    def selection(population, fitness\_scores):

        # Выбираем двух наилучших особей

        best\_indices = np.argsort(fitness\_scores)[:2]

        return [population[i] for i in best\_indices]

    # Оператор кроссовера (одноточечный кроссовер)

    def crossover(parent1, parent2):

        crossover\_point = np.random.randint(1, len(parent1))

        child = np.hstack((parent1[:crossover\_point], parent2[crossover\_point:]))

        return child

    # Оператор мутации

    def mutate(individual, mutation\_rate):

        mutation\_indices = np.random.rand(len(individual)) < mutation\_rate

        individual[mutation\_indices] += np.random.uniform(-0.5, 0.5)

        return individual

    def run\_optimization():

        # Генерация сетки для графика целевой функции

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

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

        X, Y = np.meshgrid(x\_range, y\_range)

        Z = rosenbrock\_function(X, Y)

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

        num\_generations = int(y\_var.get())

        # рандомно задаем популяцию от -5 до 5

        population = np.random.uniform(low=-5, high=5, size=(population\_size, 2))

        # для записи результатов

        results = []

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

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

        for generation in range(num\_generations):

            # Расчет значений функции для текущей популяции

            fitness\_scores = np.array([rosenbrock\_function(x, y) for x, y in population])

            # Выбор лучших особей

            selected\_individuals = selection(population, fitness\_scores)

            # Оператор кроссовера и мутации

            children = []

            for i in range(0, population\_size, 2):

                child1 = crossover(selected\_individuals[0], selected\_individuals[1])

                child2 = crossover(selected\_individuals[1], selected\_individuals[0])

                child1 = mutate(child1, mutation\_rate=0.1)  # Пример вероятности мутации

                child2 = mutate(child2, mutation\_rate=0.1)

                children.extend([child1, child2])

            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("Генетический алгоритм")

            for i in range(len(fitness\_scores)):

                best\_individual = population[i]

                ax.scatter(best\_individual[0], best\_individual[1], fitness\_scores[i], color='red',

                           s=10)

            # Обновление популяции

            population = np.array(children)

            # Нахождение лучшей особи на текущей итерации

            best\_fitness = np.min(fitness\_scores)

            best\_individual = population[np.argmin(fitness\_scores)]

            # Вывод лучшего решения на текущей итерации

            print(f"Поколение {generation}: Лучшее решение - {best\_individual}, Значение функции - {best\_fitness}")

            results.append((best\_individual[0], best\_individual[1], generation, best\_fitness))

            results\_text.insert(tk.END,

                                f"Поколение {generation}: Лучшее решение ({best\_individual[0]:.2f}, {best\_individual[1]:.2f}), Значение функции: {best\_fitness:.7f}\n")

            results\_text.yview\_moveto(1)

            canvas.draw()

            root.update()

        # Нахождение лучшего решения после всех итераций

        final\_fitness\_scores = np.array([rosenbrock\_function(x, y) for x, y in population])

        best\_index = np.argmin(final\_fitness\_scores)

        best\_solution = population[best\_index]

        best\_fitness\_value = final\_fitness\_scores[best\_index]

        results\_text.insert(tk.END,

                            f"\nОптимизация завершена. Лучшее решение - {best\_solution}, Значение функции - {best\_fitness\_value}")

        results\_text.yview\_moveto(1)

        ax.scatter(best\_solution[0], best\_solution[1], best\_fitness\_value, color='black', marker='x', s=60)

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

    def clear\_results():

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

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

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

    bg\_color = "#3d6466"

    fg\_color = "black"

    results\_bg\_color = "#4a8789"

    button\_color = "#28393a"

    font\_large = ("Arial", 14, "bold")

    font\_small = ("Arial", 10)

    style = ttk.Style()

    style.configure("TLabel", background=bg\_color, foreground=fg\_color, font=font\_small)

    style.configure("TButton", background=button\_color, foreground=fg\_color, font=font\_small)

    style.configure("TEntry", fieldbackground=bg\_color, foreground=fg\_color, font=font\_small)

    style.configure("TCombobox", fieldbackground=bg\_color, foreground=fg\_color, font=font\_small)

    style.configure("Horizontal.TSeparator", background=fg\_color) # Separator color

    param\_frame2 = frame

    param\_frame2.configure(bg=bg\_color)

    ttk.Label(param\_frame2, text="Genetic Algorithm Parameters", font=font\_large).grid(row=0, column=0, pady=5, columnspan=2) # Убрал интервал

    ttk.Label(param\_frame2, text="Population Size:", font=font\_small).grid(row=1, column=0, sticky="w", padx=10)

    ttk.Label(param\_frame2, text="Generations:", font=font\_small).grid(row=2, column=0, sticky="w", padx=10)

    x\_var = tk.DoubleVar(value=100)

    y\_var = tk.DoubleVar(value=100)

    x\_entry = ttk.Entry(param\_frame2, textvariable=x\_var, width=15)

    y\_entry = ttk.Entry(param\_frame2, textvariable=y\_var, width=15)

    x\_entry.grid(row=1, column=1, padx=10, pady=5, sticky="e")

    y\_entry.grid(row=2, column=1, padx=10, pady=5, sticky="e")

    separator1 = ttk.Separator(param\_frame2, orient="horizontal")

    separator1.grid(row=3, column=0, columnspan=2, sticky="ew", pady=5, padx=10) # Убрал интервал

    ttk.Label(param\_frame2, text="Function and Graph Display", font=font\_large).grid(row=4, column=0, pady=5, columnspan=2) # Убрал интервал

    ttk.Label(param\_frame2, text="Select Function:", font=font\_small).grid(row=5, column=0, sticky="w", padx=10)

    function\_choices = ["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=5, column=1, pady=5, padx=10, sticky="e")

    button\_frame = tk.Frame(param\_frame2, bg=bg\_color)

    button\_frame.grid(row=6, column=0, columnspan=2, pady=5) # Убрал интервал

    apply\_settings\_button = ttk.Button(button\_frame, text="Run Optimization", command=run\_optimization, style="My.TButton")

    apply\_settings\_button.pack(side=tk.LEFT, padx=10)

    clear\_button = ttk.Button(button\_frame, text="Clear Results", command=clear\_results, style="My.TButton")

    clear\_button.pack(side=tk.LEFT, padx=10)

    ttk.Label(param\_frame2, text="Optimization Results", font=font\_large).grid(row=7, column=0, pady=5, columnspan=2) # Убрал интервал

    results\_text = scrolledtext.ScrolledText(param\_frame2, wrap=tk.WORD, height=10, width=40, padx=2, state=tk.DISABLED, bg="#4a8789", fg=fg\_color)

    results\_text.grid(row=8, column=0, padx=10, columnspan=2, sticky="ew")