МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра вычислительных технологий

ЛАБОРАТОРНАЯ РАБОТА №3

Дисциплина: Методы поисковой оптимизации

Тема: Генетический алгоритм оптимизации функции Розенброка

Работу выполнили: _____ Парфинцов. Е. А.

_____ Татарян. Е. В.

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Направленность (профиль): Математическое и программное обеспечение компьютерных технологий

Преподаватель: _____ Полупанова Е. Е.

Краснодар
2024

Цель работы: разработать генетический алгоритм оптимизации функции Розенброка.

Ход работы

Генетический алгоритм (ГА) — это метод оптимизации, вдохновленный процессами естественного отбора и генетики. Этот алгоритм используется для решения задач оптимизации и поиска, особенно в ситуациях, где простое и полное переборное решение невозможно или неэффективно.

Ключевые элементы генетического алгоритма:

1. *Популяция* – набор особей с заданными свойствами.
2. *Фитнес-функция* – функция, принимающая в качестве аргумента особь и возвращающая численное значение, характеризующее приспособленность особи.
3. *Селекция* – процесс отбора наиболее приспособленных особей.
4. *Скрещивание (кроссовер)* – формирование новой особи на основе двух других особей. При скрещивании характеристики особей комбинируются, что вносит разнообразие в популяцию.
5. *Мутация* – случайные изменения, происходящие с характеристиками особей в процессе отбора.

Рассмотрим реализацию генетического алгоритма на функции *Розенброка*:

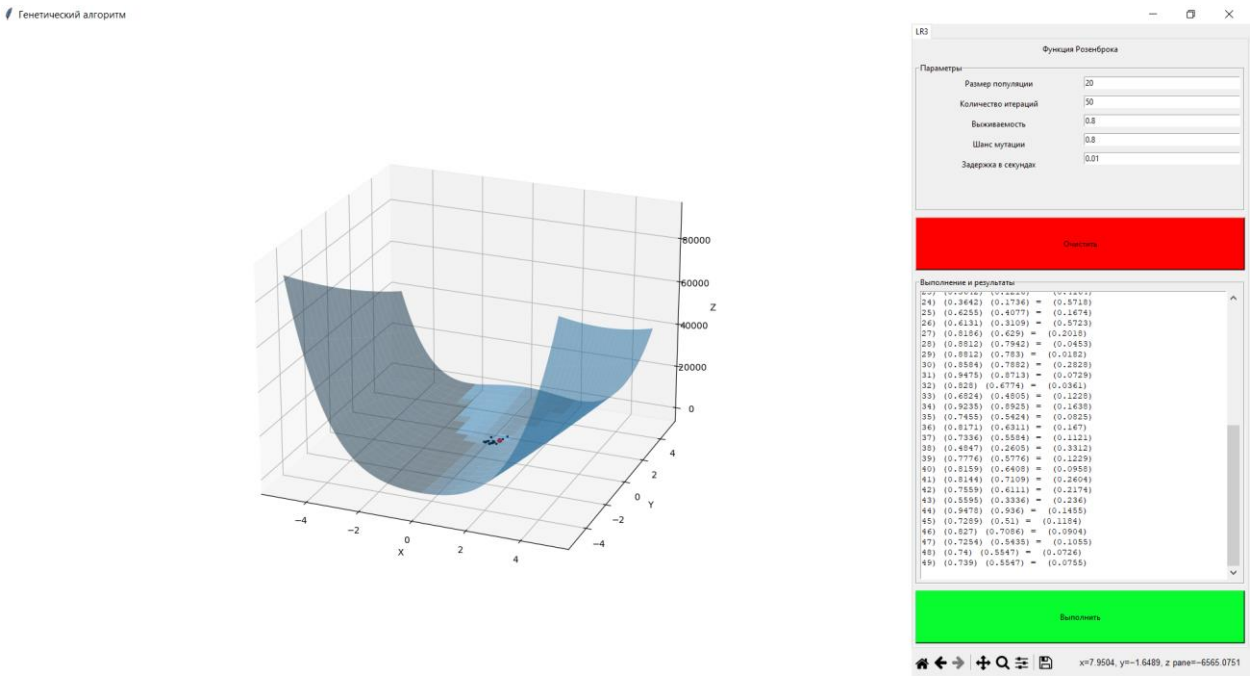
1. В начале работы алгоритма создается начальная популяция из случайных особей. Каждая особь представляется двумя генами (x, y), и ей ставится в соответствие значение фитнес-функции (z), которое рассчитывается на основе значений x и y.

2. Особи в текущей популяции упорядочиваются 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). 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. Этот процесс повторяется для нескольких новых особей. Также каждая особь в популяции имеет шанс быть подверженной мутации. Это означает, что гены могут быть незначительно изменены случайным образом. Вероятность мутации определяется параметром mut_chance.

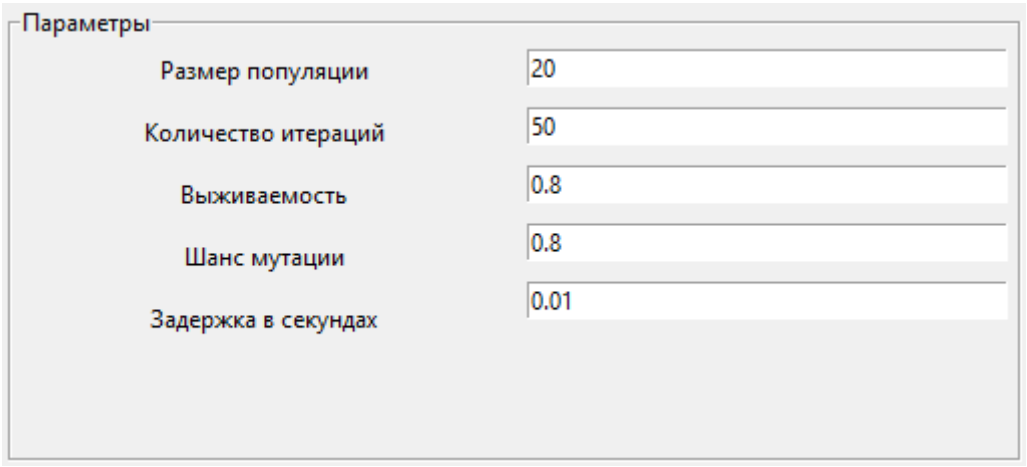
3. Шаг 2 повторяется в течение заданного количества поколений (generations). В каждом поколении лучшие особи сохраняются, а менее успешные могут быть заменены новыми особями.

4. После завершения всех итераций алгоритм возвращает особь с минимальным значением z.

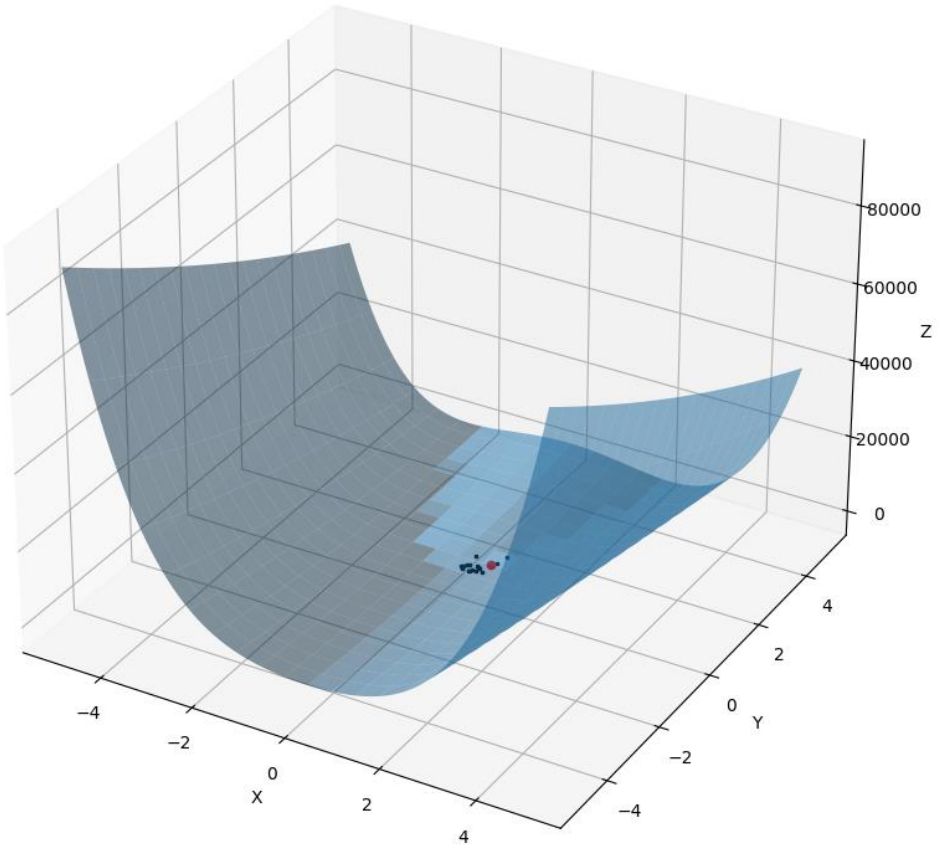
Реализуем данный алгоритм на языке Python (листинги 1–2):



Скриншот 1. Внешний вид приложения.



Скриншот 2. Параметры алгоритма.



Скриншот 2. Вид графика функции Розенброка.

```
49) (0.739) (0.5547) =  (0.0755)
```

Скриншот 3. Результат работы

Функция Розенброка достигает глобального минимума = 0 в точке (1, 1).

Таким образом, генетический алгоритм с заданными параметрами позволяет найти значение минимума функции с относительной погрешностью ≅7.6 %.

Листинг 1. (Шаги алгоритма)

```
from random import uniform, random

class GeneticAlgorithmL3:
    # Функция приспособленности, количество поколений, вероятность мутации,
коэффициент выживаемости, размер популяции.
    def __init__(self, func, generations=50, min=True, mut_chance=0.8,
survive_cof=0.8, pop_number=100):
        self.func = func
        self.population = dict()
        self.mut_chance = mut_chance
        self.survive_cof = survive_cof
        self.generations = generations
        self.pop_number = pop_number
        self.min_func = min

# Генерирует исходную популяцию особей, каждая из которых представлена двумя
генами (x и y)
# и соответствующим им значением приспособленности.
    def generate_start_population(self, x, y):
        for i in range(self.pop_number):
            po_x = uniform(-x, x)
            po_y = uniform(-y, y)
            self.population[i] = [po_x, po_y, self.func(po_x, po_y)]  # Создание
начальной популяции

# Возвращает лучших и худших индивидуумов в текущей популяции на основе их
показателей физической подготовки.
    def statistic(self):
        return [max(self.population.items(), key=lambda item: item[1][2]),
                min(self.population.items(), key=lambda item: item[1][2])]

# Выполняет процесс отбора, который упорядочивает особей в популяции на основе их
показателей пригодности
# и выбирает родителей для следующего поколения.
    def select(self):
        sorted_pop = dict(
            sorted(self.population.items(), key=lambda item: item[1][2],
reverse=self.min_func))  # Ранжирование
# Кроссинговер - случайным образом выбираются 2 родителя и создаются 2 ребенка
путем обмена их генами.
        cof = int(self.pop_number * (1 - self.survive_cof))
        parents1 = list(sorted_pop.items())[cof: cof * 2]
        parents2 = list(sorted_pop.items())[self.pop_number - cof:
self.pop_number]

        i = 0
        for pop in sorted_pop.values():
            if random() > 0.5:
                pop[0] = parents1[i][1][0]
```

```
                pop[1] = parents2[i][1][1]
                pop[2] = self.func(parents1[i][1][0], parents2[i][1][1])
            else:
                pop[0] = parents2[i][1][0]
                pop[1] = parents1[i][1][1]
                pop[2] = self.func(parents2[i][1][0], parents1[i][1][1])
            i += 1
            if i >= cof:
                break

        self.population = sorted_pop

# Вносятся случайные изменения в гены индивидуумов с определенной вероятностью,
# что помогает исследовать новые области пространства решений.
    def mutation(self, cur_gen):
        for pop in self.population.values():
            if random() < self.mut_chance:
                pop[0] += (random() - 0.5) * ((self.generations - cur_gen) /
self.generations)
            if random() < self.mut_chance:
                pop[1] += (random() - 0.5) * ((self.generations - cur_gen) /
self.generations)
            pop[2] = self.func(pop[0], pop[1])
```

Листинг 2. (Приложение и запуск алгоритма)

```
import tkinter
import time
from tkinter import *
from tkinter import scrolledtext, messagebox
from tkinter.ttk import Combobox, Notebook
from matplotlib import pyplot as plt
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,
NavigationToolbar2Tk)
from Rosenbrock_function import make_data_lab_3
from genetic_algorithm_l3 import GeneticAlgorithmL3
from functions import *

def main():
    window = Tk()

    window.geometry("1000x800")

    window.title("Генетический алгоритм")

    fig = plt.figure(figsize=(14, 14))
    fig.add_subplot(projection='3d')

    canvas = FigureCanvasTkAgg(fig, master=window)
    canvas.draw()
    canvas.get_tk_widget().pack(side=tkinter.LEFT, fill=tkinter.BOTH)

    toolbar = NavigationToolbar2Tk(canvas, window)
    toolbar.update()
    canvas.get_tk_widget().pack(side=tkinter.LEFT, fill=tkinter.BOTH)


    tab_control = Notebook(window)

    # ЛР №3
    def draw_lab_3():
        fig.clf()

        x, y, z = make_data_lab_3()

        pop_number = int(txt_1_tab_3.get())
        iter_number = int(txt_2_tab_3.get())
        survive = float(txt_3_tab_3.get())
        mutation = float(txt_4_tab_3.get())
        delay = txt_5_tab_3.get()

        if combo_tab_3.get() == "Min":
            min_max = True
        else:
            min_max = False
```

```
        ax = fig.add_subplot(projection='3d')
        ax.plot_surface(x, y, z, rstride=5, cstride=5, alpha=0.5)
        canvas.draw()

        genetic = GeneticAlgorithmL3(rosenbrock_2, iter_number, min_max,
mutation, survive, pop_number)
        genetic.generate_start_population(5, 5)

        for j in range(pop_number):
            ax.scatter(genetic.population[j][0], genetic.population[j][1],
genetic.population[j][2], c="black", s=1,
                       marker="s")
        if min_max:
            gen_stat = list(genetic.statistic()[1])
        else:
            gen_stat = list(genetic.statistic()[0])

        ax.scatter(gen_stat[1][0], gen_stat[1][1], gen_stat[1][2], c="red")
        canvas.draw()
        window.update()

        #Строки 89-92 удаляют точк(у/и)
        fig.clf()
        ax = fig.add_subplot(projection='3d')
        ax.plot_surface(x, y, z, rstride=5, cstride=5, alpha=0.5)
        canvas.draw()

        for i in range(50):
            for j in range(pop_number):
                ax.scatter(genetic.population[j][0], genetic.population[j][1],
genetic.population[j][2], c="black", s=1,
                           marker="s")

            genetic.select()
            genetic.mutation(i)

            if min_max:
                gen_stat = list(genetic.statistic()[1])
            else:
                gen_stat = list(genetic.statistic()[0])

            ax.scatter(gen_stat[1][0], gen_stat[1][1], gen_stat[1][2], c="red")

            txt_tab_3.insert(INSERT,
                             f"{i}) ({round(gen_stat[1][0], 4)})
({round(gen_stat[1][1], 4)}) = "
                             f" ({round(gen_stat[1][2], 4)})\n")

            canvas.draw()
            window.update()
```

```
            time.sleep(float(delay))

            fig.clf()
            ax = fig.add_subplot(projection='3d')
            ax.plot_surface(x, y, z, rstride=5, cstride=5, alpha=0.5)
            canvas.draw()

        for j in range(pop_number):
            ax.scatter(genetic.population[j][0], genetic.population[j][1],
genetic.population[j][2], c="black", s=1,
                       marker="s")
        if min_max:
            gen_stat = list(genetic.statistic()[1])
        else:
            gen_stat = list(genetic.statistic()[0])

        ax.scatter(gen_stat[1][0], gen_stat[1][1], gen_stat[1][2], c="red")

        canvas.draw()
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        window.update()

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

    def delete_lab_3():
        txt_tab_3.delete(1.0, END)

    tab_3 = Frame(tab_control)
    tab_control.add(tab_3, text="LR3")

    main_f_tab_3 = LabelFrame(tab_3, text="Параметры")
    left_f_tab_3 = Frame(main_f_tab_3)
    right_f_tab_3 = Frame(main_f_tab_3)
    txt_f_tab_3 = LabelFrame(tab_3, text="Выполнение и результаты")

    lbl_1_tab_3 = Label(left_f_tab_3, text="Размер популяции")
    lbl_2_tab_3 = Label(left_f_tab_3, text="Количество итераций")
    lbl_3_tab_3 = Label(left_f_tab_3, text="Выживаемость")
    lbl_7_tab_3 = Label(left_f_tab_3, text="Шанс мутации")
    #lbl_4_tab_3 = Label(left_f_tab_3, text="Выбор точки поиска")
    lbl_5_tab_3 = Label(left_f_tab_3, text="Задержка в секундах")
    lbl_6_tab_3 = Label(tab_3, text="Функция Розенброка")

    txt_1_tab_3 = Entry(right_f_tab_3)
    txt_1_tab_3.insert(0,"20")
    txt_2_tab_3 = Entry(right_f_tab_3)
    txt_2_tab_3.insert(0,"50")
    txt_3_tab_3 = Entry(right_f_tab_3)
    txt_3_tab_3.insert(0,"0.8")
```

```
    txt_4_tab_3 = Entry(right_f_tab_3)
    txt_4_tab_3.insert(0,"0.8")
    txt_5_tab_3 = Entry(right_f_tab_3)
    txt_5_tab_3.insert(0,"0.01")

    combo_tab_3 = Combobox(right_f_tab_3)
    combo_tab_3['values'] = ("Min", "Max")
    combo_tab_3.set("Min")

    txt_tab_3 = scrolledtext.ScrolledText(txt_f_tab_3)
    btn_del_tab_3 = Button(tab_3, text="Очистить", foreground="black",
background="red", command=delete_lab_3)
    btn_tab_3 = Button(tab_3, text="Выполнить", foreground="black",
background="#08fc30", command=draw_lab_3)

    lbl_6_tab_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    main_f_tab_3.pack(side=TOP, padx=5, pady=5, fill=BOTH, expand=True)
    left_f_tab_3.pack(side=LEFT, fill=BOTH, expand=True)
    right_f_tab_3.pack(side=RIGHT, fill=BOTH, expand=True)

    lbl_1_tab_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    lbl_2_tab_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    lbl_3_tab_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    lbl_7_tab_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    lbl_5_tab_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    #lbl_4_tab_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)

    txt_1_tab_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    txt_2_tab_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    txt_3_tab_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    txt_4_tab_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)  # задержка в секундах
    txt_5_tab_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)  # шанс мутации
    #combo_tab_3.pack(side=TOP, padx=5, pady=5, fill=BOTH)

    txt_tab_3.pack(padx=5, pady=5, fill=BOTH, expand=True)

    btn_tab_3.pack(side=BOTTOM, padx=5, pady=5, fill=BOTH, expand=True)
    txt_f_tab_3.pack(side=BOTTOM, padx=5, pady=5, fill=BOTH, expand=True)
    btn_del_tab_3.pack(side=BOTTOM, padx=5, pady=5, fill=BOTH, expand=True)

    tab_control.pack(side=RIGHT, fill=BOTH, expand=True)
    window.mainloop()

if __name__ == '__main__':
    main()
```