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 49/1 группы

Епифанцев В.А.

Григорьян А.А.

Преподаватель:

Нигодин Е.А.

Краснодар 2023

**Цель работы:** разработать генетический алгоритм оптимизации функции Розенброка.

**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 полно-переборное решение невозможно или неэффективно. Генетические алгоритмы часто применяются для нахождения приближенных решений в множестве возможных вариантов.

Рассмотрим ключевые элементы генетического алгоритма и его работу:

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 останова**. Генетический алгоритм продолжает создавать новые поколения и улучшать популяцию до тех пор, пока не выполнится определенный критерий останова. 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 оптимальным значением z, то есть особь с минимальным значением z.

**Особенности реализации генетического алгоритма**

Для создания программы использовался язык программирования Python и среда разработки PyCharm. Для графической визуализации были подключен графический фреймворк Matplotlib.

Программа выводит в качестве результата окно с отображением функции (рис. 1).

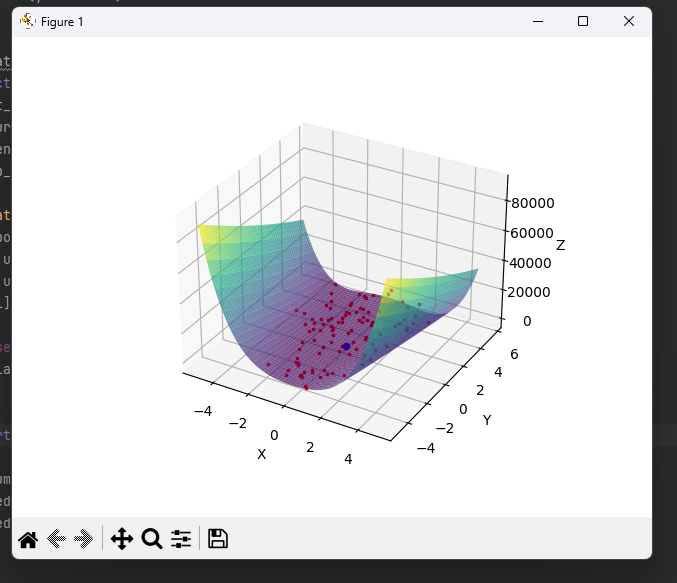


Рисунок 1 – Главное окно программы

**Вывод:** в ходе работы был изучен и реализован генетический алгоритм на определенной функции.

**Листинг программы**

Файл lab3.py

import numpy as np

import matplotlib.pyplot as plt

def rosenbrock\_func(x, y):

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

class GeneticAlgorithm:

def \_\_init\_\_(self, generations=50, mut\_chance=0.8, survive\_cof=0.8, pop\_number=100):

self.population = dict()

self.mut\_chance = mut\_chance

self.survive\_cof = survive\_cof

self.generations = generations

self.pop\_number = pop\_number

def generate\_start\_population(self, x, y):

for i in range(self.pop\_number):

po\_x = np.random.uniform(-x, x)

po\_y = np.random.uniform(-y, y)

self.population[i] = [po\_x, po\_y, rosenbrock\_func(po\_x, po\_y)]

def get\_best\_individual(self):

return min(self.population.items(), key=lambda item: item[1][2])

def select(self):

sorted\_pop = dict(sorted(self.population.items(), key=lambda item: item[1][2], reverse=True))

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 np.random.random() > 0.5:

pop[0] = parents1[i][1][0]

pop[1] = parents2[i][1][1]

pop[2] = rosenbrock\_func(parents1[i][1][0], parents2[i][1][1])

else:

pop[0] = parents2[i][1][0]

pop[1] = parents1[i][1][1]

pop[2] = rosenbrock\_func(parents2[i][1][0], parents1[i][1][1])

i += 1

if i >= cof:

break

self.population = sorted\_pop

def mutation(self, cur\_gen):

for pop in self.population.values():

if np.random.random() < self.mut\_chance:

pop[0] += (np.random.random() - 0.5) \* ((self.generations - cur\_gen) / self.generations)

if np.random.random() < self.mut\_chance:

pop[1] += (np.random.random() - 0.5) \* ((self.generations - cur\_gen) / self.generations)

pop[2] = rosenbrock\_func(pop[0], pop[1])

def visualize\_evolution(genetic, X, Y, Z, ax, pop\_number, iter\_number):

for i in range(iter\_number):

ax.clear()

ax.set\_xlabel('X')

ax.set\_ylabel('Y')

ax.set\_zlabel('Z')

ax.plot\_surface(X, Y, Z, cmap='viridis', alpha=0.7)

for j in range(pop\_number):

ax.scatter(genetic.population[j][0], genetic.population[j][1], genetic.population[j][2], c="red", s=1, marker="s")

best\_individual = genetic.get\_best\_individual()

ax.scatter(best\_individual[1][0], best\_individual[1][1], best\_individual[1][2], c="blue")

plt.draw()

plt.pause(0.1)

genetic.select()

genetic.mutation(i)

ax.clear()

ax.set\_xlabel('X')

ax.set\_ylabel('Y')

ax.set\_zlabel('Z')

ax.plot\_surface(X, Y, Z, cmap='viridis', alpha=0.7)

for j in range(pop\_number):

ax.scatter(genetic.population[j][0], genetic.population[j][1], genetic.population[j][2], c="red", s=1, marker="s")

best\_individual = genetic.get\_best\_individual()

ax.scatter(best\_individual[1][0], best\_individual[1][1], best\_individual[1][2], c="blue")

plt.show()

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

ax.set\_xlabel('X')

ax.set\_ylabel('Y')

ax.set\_zlabel('Z')

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

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

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

Z = rosenbrock\_func(X, Y)

ax.plot\_surface(X, Y, Z, cmap='viridis', alpha=0.7)

generations = 20

mut\_chance = 0.8

survive\_cof = 0.8

pop\_number = 100

iter\_number = 20

genetic = GeneticAlgorithm(generations, mut\_chance, survive\_cof, pop\_number)

genetic.generate\_start\_population(5, 5)

visualize\_evolution(genetic, X, Y, Z, ax, pop\_number, iter\_number)