ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

д-р техн. наук, профессор		Ю. А. Скобцов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №5

Оптимизация многомерных функций с помощью эволюционной стратегии.

по курсу: ЭВОЛЮЦИОННЫЕ МЕТОДЫ ПРОЕКТИРОВАНИЯ ПРОГРАММНО-ИНФОРМАЦИОННЫХ СИСТЕМ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4031		
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2023

Задание

1) Создать программу, использующую ЭС для нахождения оптимума функции согласно таблице вариантов, приведенной в приложении А. Для всех Benchmark-ов оптимумом является минимум

2) Для n=2 вывести на экран график данной функции с указанием найденного экстремума, точек популяции. Для вывода графиков использовать стандартные возможности пакета Matlab. Предусмотреть возможность пошагового просмотра процесса поиска решения.

3) Исследовать зависимость времени поиска, числа поколений (генераций), точности нахождения решения от основных параметров генетического алгоритма:

   - число особей в популяции

   - вероятность мутации.

4) Критерий остановки вычислений – повторение лучшего результата заданное количество раз или достижение популяцией определенного возраста (например, 100 эпох).

5) Повторить процесс поиска решения для n=3, сравнить результаты, скорость работы программы.

6) **Вариант 2:**

Вид функции	Кол-во переменных N	Промежуток исследования
Ackley’s path	2	-1 <= x(i) <= 1

Теоретические сведения.

ЭС изначально разработаны для решения многомерных оптимизационных задач, где пространство поиска – многомерное пространство вещественных чисел [1]. Иногда при решении задачи накладываются некоторые ограничения, например вида $g_i(x)>0$.

Ранние эволюционные стратегии (ЭС) основывались на популяции, состоящей из одной особи, и в них использовался только один генетический оператор – мутация. Здесь для представления особи (потенциального решения) была использована идея, не представленная в классическом генетическом алгоритме, которая заключается в следующем.

Результат выполнения программы.

Используемые операторы в ГА:

1. **Генерация начальной популяции:** Полная и растущая генерация.
2. **Оператор селекции:**
   - Турнирный отбор из 2-ух особей.
   - Количество участников в турнире: 2.
3. **Оператор скрещивания:**
   - Узловой одноточечный кроссовер.
4. **Оператор мутации:**
   - Растущая мутация.

Параметры ГА:

- **Фитнес-функция:**

  Оценка методом наименьших квадратов функции Розенброка.

- **Мощность популяции:**

  400 индивидуумов в популяции.

- **Максимум генераций:**

  500 поколений.

- **Вероятность кроссинговера:**

  0.5 (50%).

- **Вероятность мутации:**

  0.1 (10%).

Задача оптимизации:

- **Цель:**
  - Минимизация значения функции.
- **Ограничения:**
  - Переменные хромосомы ограничены интервалом от -1 до 1.

Визуализация:

- Визуализация популяции в двумерном пространстве с использованием цветовой карты.

Вывод результатов:

- Отображение лучшего индивида на каждой итерации, а также предоставление информации о популяции в выбранных поколениях.

Демонстрация работы программы:

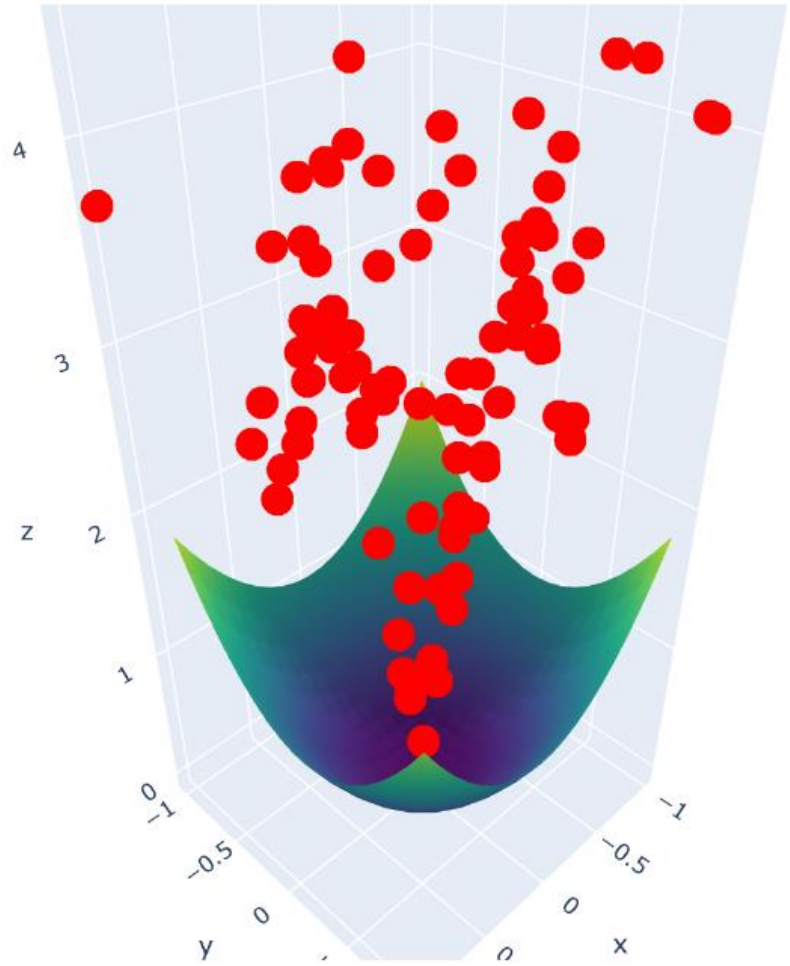
Параметры:

N = 2

Размер популяции = 200

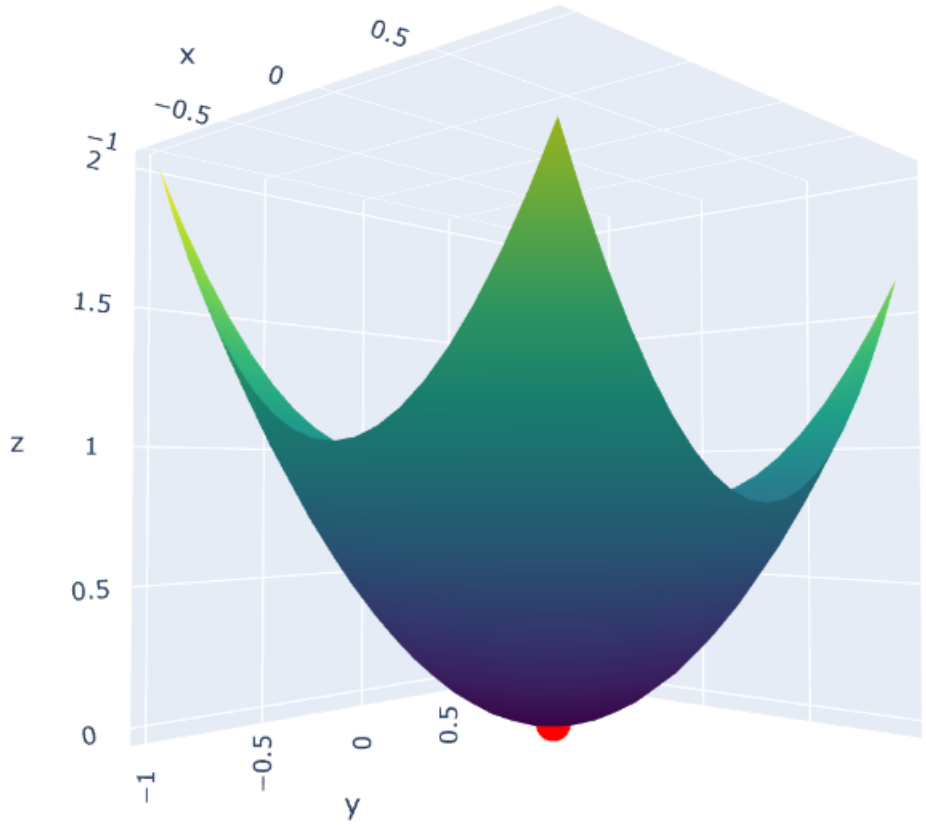
Количество генераций = 1500

Кроссинговер = 0.5




```
Лучший индивидуум в поколении:
Chromosome: -0.040656   -0.035167   Fitness Value: 0.227532
```

Рис. 1. Начальное поколение




```
Лучший индивидуум в поколении:
Chromosome: 0.001278    -2.6e-05    Fitness Value: 0.00366
```

Рис. 2. Конечное поколение

```
Best value across all generations:
Chromosome: 0.001278    -2.6e-05    Fitness Value: 0.00366
```

Рис. 3. Результат выполнения ГА

Параметры:

N = 3

Размер популяции = 200

Количество генераций = 1500

Кроссинговер = 0.5

Результат работы программы:

```
Best value across all generations:
Chromosome: -0.004733   -0.001113   -0.006288   Fitness Value: 0.019478
```

Параметры:

N = 10

Размер популяции = 100

Количество генераций = 500

Кроссинговер = 0.5

Результат работы программы:

```
Best value across all generations:
Chromosome: 0.150189   0.085967   -0.46491   -0.015847   0.037684   0.043644   -0.006758   0.019818   -0.08361   0.254124   Fitness Value: 1.558641
```

Выводы

В ходе выполнения лабораторной работы была изучена оптимизация функций многих переменных методом эволюционной стратегии. Данный алгоритм является очень быстрым и хорошо находит оптимальные значения при небольшом количестве переменных, но при увеличении количества переменных эффективность алгоритма падает. При увеличении количества переменных n, время выполнения программы увеличивается.

Листинг программы

```
 import numpy as np
import random
import math
import os
import plotly.graph_objects as go
from math import pi

LEFT_REST = -1  # Update the left boundary
RIGHT_REST = 1  # Update the right boundary
MEAN = 0
SIGMA = 0.5
TOUR_POWER = 3
VAR_COUNT = 10  # Number of variables
POPULATION_SIZE = 100
MAX_GENERATIONS = 500
P_CROSSOVER = 0.5

class Individual(object):
    def __init__(self, chromosome, value=0):
        self.fit = 0
        if value == 0:
            self.list = chromosome
        elif value == "Clone":
            buff = list()
            for i in range(0, len(chromosome)):
                buff.append(chromosome[i])
            self.list = buff
        elif value == "Random":
            self.list = list()
            for i in range(0, chromosome):
                self.list.append(random.uniform(LEFT_REST, RIGHT_REST))

    def print(self):
        print("Chromosome:", end=" ")
        for i in range(0, len(self.list)):
            print(round(self.list[i], 6), end="\t")
        print("Fitness Value:", round(realFitness(self), 6))

def realFitness(individual):
    sum1 = 0.0
    sum2 = 0.0
    for i in range(0, VAR_COUNT):
        x = individual.list[i]
        sum1 += x**2
        sum2 += math.cos(2 * pi * x)
    term1 = -20.0 * math.exp(-0.2 * math.sqrt(sum1 / VAR_COUNT))
    term2 = -math.exp(sum2 / VAR_COUNT)
    individual.fit = term1 + term2 + 20 + math.exp(1)
    return individual.fit

def findMinInPopulation(population):
    return min(population, key=lambda x: realFitness(x))

def tournament(population):
    uniques = list(set(population))
    offspring = []
    for n in range(len(population)):
        aspirants = list()
        for i in range(0, TOUR_POWER):
            aspirants.append(Individual(uniques[random.randint(0, len(uniques) -
1)].list, "Clone"))
```

```
        offspring.append(min(aspirants, key=lambda x: realFitness(x)))
    return offspring

def crossover(parent):
    son = Individual(parent.list, "Clone")
    for i in range(0, len(parent.list)):
        son.list[i] += np.random.normal(MEAN, SIGMA)
    if realFitness(son) < realFitness(parent):
        return son
    else:
        return parent

population = [Individual(VAR_COUNT, "Random") for i in range(POPULATION_SIZE)]
data = list()
best = population[0]

for j in range(MAX_GENERATIONS):
    current = list()
    for i in range(len(population)):
        if random.random() < P_CROSSOVER:
            population[i] = crossover(population[i])
    population = tournament(population)
    for pop in population:
        current.append(pop)
    data.append(current)
    temp = findMinInPopulation(population)
    if temp.fit < best.fit:
        best = temp
    print(j)
    best.print()
    print()
    print()

print("Best value across all generations:")
best.print()

print("Введите номер нужного поколения ( 0 :", MAX_GENERATIONS - 1, ") или слово
break для выхода из цикла:")
text = input()
while (str)(text) != "break":
    print()
    if (int)(text) >= 0 and (int)(text) < MAX_GENERATIONS:
        index = (int)(text)
        x, y, z = list(), list(), list()
        print("Популяция в", text, "поколении")
        for i in data[index]:
            if len(i.list) == 2:
                x.append(i.list[0])
                y.append(i.list[1])
                z.append(i.fit)
                i.print()
        print()
        print("\nЛучший индивидуум в поколении:")
        r = findMinInPopulation(data[index])
        r.print()
        if VAR COUNT == 2:
            graphic = []
            for i in range(3):
                graphic.append(list())
            x_vals = np.arange(LEFT_REST, RIGHT_REST, 0.1)
            y_vals = np.arange(LEFT_REST, RIGHT_REST, 0.1)
            x_mesh, y_mesh = np.meshgrid(x_vals, y_vals)
            z_mesh = x_mesh**2 + y_mesh**2
            layout1 = go.Layout(title=go.layout.Title(text=" ", x=0.5))
```

```
            fig = go.Figure()
            fig.add_trace(go.Surface(x=x_mesh, y=y_mesh, z=z_mesh,
colorscale='Viridis'))
            fig.add_trace(go.Scatter3d(x=x, y=y, z=z, mode='markers',
marker=dict(size=10, color="red")))
            fig.show()
    print("Введите номер нужного поколения ( 0 -", MAX_GENERATIONS - 1, ") или
слово break для выхода из цикла:")
    text = input()
```