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 №7

Оптимизация путей на графах с помощью роевых алгоритмов

по курсу: ЭВОЛЮЦИОННЫЕ МЕТОДЫ ПРОЕКТИРОВАНИЯ ПРОГРАММНО-ИНФОРМАЦИОННЫХ СИСТЕМ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №		
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2023

Задание

1) Создать программу, использующую РА 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: Критерий остановки вычислений – повторение лучшего результата заданное количество раз или достижение популяцией определенного возраста (например, 100 эпох).

4) Повторить процесс поиска решения для n=3, n=5, n=10, сравнить результаты, скорость работы программы.

5) Вариант 10

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

Теоретические сведения

Роевой алгоритм (РА) использует рой частиц, где каждая частица представляет потенциальное решение проблемы. Поведение частицы в гиперпространстве поиска решения все время подстраивается в соответствии со своим опытом и опытом своих соседей. Кроме этого, каждая частица помнит свою лучшую позицию с достигнутым локальным лучшим значением целевой (фитнесс-) функции и знает наилучшую позицию частиц - своих соседей, где достигнут глобальный на текущий момент оптимум.

В процессе поиска частицы роя обмениваются информацией о достигнутых лучших результатах и изменяют свои позиции и скорости по определенным правилам на основе имеющейся на текущий момент информации о локальных и глобальных достижениях.

При этом глобальный лучший результат известен всем частицам и немедленно корректируется в том случае, когда некоторая частица роя находит лучшую позицию с результатом, превосходящим текущий глобальный оптимум.

Описание работы программы.

Используемые операторы в оптимизации с помощью алгоритма роя частиц (PSO):

1. **Инициализация популяции частиц:**
   - Создание начальной популяции частиц с случайными координатами в пространстве поиска.
2. **Оператор коррекции скорости:**
   - Коррекция скорости частицы с использованием коэффициентов c1 и c2 для управления влиянием лучших индивидуальных и глобальных позиций.
3. **Оператор коррекции позиции:**
   - Обновление позиции частицы на основе её скорости.

Параметры оптимизации с использованием PSO:

- **Количество частиц в популяции:**
  - 300 частиц.
- **Размерность пространства поиска:**
  - 2 переменные (n=2).
- **Коэффициенты коррекции скорости c1 и c2:**
  - Оба коэффициента установлены в 1.5.
- **Интервалы для каждой переменной:**
  - От -1 до 1
- **Число итераций:**
  - 1000 итераций.

Демонстрация работы программы:

Количество переменных 2

Количество генераций: 500

Количество особей в популяции: 300

Коэффициент c1=3

Коэффициент c2=2

Левая граница: -5.12

Правая граница: 5.12

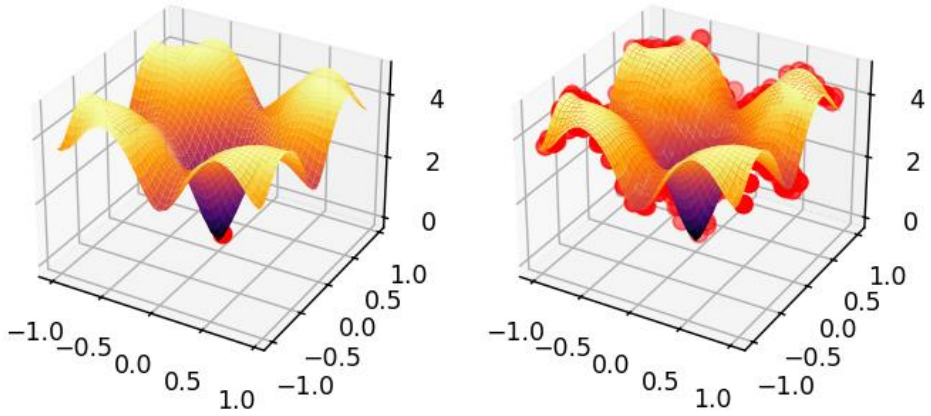

Рис. 1 Начальная популяция

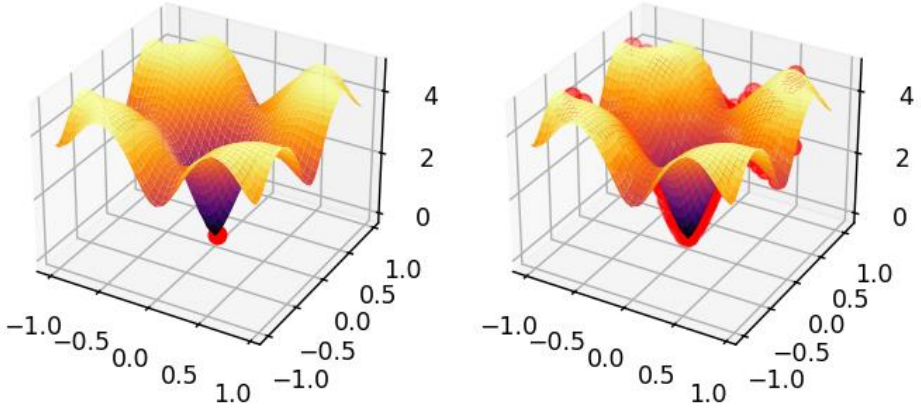

Рис. 2 Конечная популяция

```
Best particle at iteration 0:
Position:  [0.007524275566031191, 0.04702026382671387]
Function value:  0.19397014127686107
Best global value:  0.19397014127686107

Best particle at iteration 500:
Position:  [0.01819122635645648, -0.0020200331417350448]
Function value:  0.06066493447827925
Best global value:  0.19397014127686107
```

Рис. 3 Вывод программы

Количество переменных: 3

Количество генераций: 1000

Количество особей в популяции: 150

Коэффициент c1=3

Коэффициент c2=2

```
Best particle at iteration 0:
Position:  [-0.14531616769100042, 0.0589138626164325, 0.18167011375539688]
Function value:  1.3430398266914882
Best global value:  1.3430398266914882

Best particle at iteration 1000:
Position:  [-0.08649808593782554, -0.035085659089196045, 0.013905334271345882]
Function value:  0.3683145974804032
Best global value:  1.3430398266914882
```

Количество переменных: 3

Количество генераций: 500

Количество особей в популяции: 150

Коэффициент c1=3

Коэффициент c2=2

```
Best particle at iteration 0:
Position:  [-0.03878417906575038, -0.013944375017116206, -0.07378621322044498]
Function value:  0.31725384433360704
Best global value:  0.31725384433360704

Best particle at iteration 500:
Position:  [-0.03878417906575038, -0.013944375017116206, -0.07378621322044498]
Function value:  0.31725384433360704
Best global value:  0.31725384433360704
```

Количество переменных: 3

Количество генераций: 500

Количество особей в популяции: 300

Коэффициент c1=3

Коэффициент c2=2

```
Best particle at iteration 0:
Position:  [0.05395986491326621, -0.04705173649500671, 0.05565800483859129]
Function value:  0.35022484860733316
Best global value:  0.35022484860733316

Best particle at iteration 500:
Position:  [-0.031251840858841196, 0.0798154353155025, 0.015356894956526534]
Function value:  0.33008457324520935
Best global value:  0.35022484860733316
```

Количество переменных: 3

Количество генераций: 500

Количество особей в популяции: 300

Коэффициент c1=1.5

Коэффициент c2=1.5

```
Best particle at iteration 0:
Position:  [-0.07367194897310081, 0.02066724403254705, 0.016906699061650876]
Function value:  0.2861213565940166
Best global value:  0.2861213565940166

Best particle at iteration 500:
Position:  [0.007534085886122477, 0.0321468751774075, -0.002115869509718135]
Function value:  0.09570810040959943
Best global value:  0.2861213565940166
```

Количество переменных: 3

Количество генераций: 500

Количество особей в популяции: 300

Коэффициент c1=4

Коэффициент c2=4

```
Best particle at iteration 0:
Position:  [-0.19843271075843982, 0.04302067435912371, 0.055359666023179877]
Function value:  1.1009956405454777
Best global value:  1.1009956405454777

Best particle at iteration 500:
Position:  [0.05043412940029718, 0.00757786341580214, -0.06462806939669763]
Function value:  0.30644827621001314
Best global value:  1.1009956405454777
```

Количество переменных: 5

Количество генераций: 500

Количество особей в популяции: 300

Коэффициент c1=4

Коэффициент c2=4

```
Best particle at iteration 0:
Position:  [0.15094192949791552, 0.20883103695833416, 0.3924974231296543, 0.03847758740353213, -0.08121130001225474]
Function value:  2.0925654697171763
Best global value:  2.0925654697171763

Best particle at iteration 500:
Position:  [0.007315863265870637, 0.0028988471273381777, 0.21062907947035692, 0.010588437969001514, -0.049548697669916986]
Function value:  0.7890448724795402
Best global value:  2.0925654697171763
```

Количество переменных: 10

Количество генераций: 500

Количество особей в популяции: 300

Коэффициент c1=1.5

Коэффициент c2=1.5

```
Best particle at iteration 0:
Position:  [-0.1951808377891624, -0.00390031978973826, 0.9599864994766609,
 0.24970846983055028, 0.024814278850453952, 0.03886377078762693, 0.6585486831781189,
 -0.130032528892307, -0.08421557545093328, -0.5550767707781872]
Function value:  2.7991975505741356
Best global value:  2.7991975505741356

Best particle at iteration 500:
Position:  [0.19593989204993031, 0.03153751321666154, -0.10836430065223768,
 -0.005275283336312872, -0.07466265607698319, 0.004764877635574316, 0.15605384085851237,
 0.18842374503058423, -0.0032378049947434095, -0.8198270514352026]
Function value:  1.7272760890914651
Best global value:  2.7991975505741356
```

Выводы

При увеличении количества генераций можем заметить, что точность решения увеличилась, время выполнения увеличилось совсем незначительно.

При изменении коэффициентов время выполнения программы не изменилось.

Время выполнение программы увеличивается от увеличения количества переменных.

При увеличении количества переменных до 10 - время выполнения программы существенно увеличивается. Результат близок к необходимому.

В ходе выполнения лабораторной работы были изучены роевые алгоритмы.

Листинг программы

```
 import random
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from copy import deepcopy

def ackleys_path(x):
    n = len(x)
    sum_sq = sum([xi**2 for xi in x])
    sum_cos = sum([np.cos(2 * np.pi * xi) for xi in x])
    return -20 * np.exp(-0.2 * np.sqrt(sum_sq / n)) - np.exp(sum_cos / n) + 20 +
np.exp(1)

def initialization(quants, n_times, left_interv, right_interv):
    popul = []
    for k in range(quants):
        coordinates = [random.uniform(left_interv, right_interv) for _ in
range(n_times)]
        travel_speed = [0] * n_times
        cognitive_component = deepcopy(coordinates)
        popul.append([coordinates, travel_speed, cognitive_component])
    best = popul[0]
    for elem in popul:
        if ackleys_path(best[0]) > ackleys_path(elem[0]):
            best = deepcopy(elem)
    for elem in popul:
        elem.append(best[0])
    return popul

def print_results(popul, n_times, iterat, left_interv, right_interv):
    print("\nBest particle at iteration " + str(iterat) + ":")
    best = popul[0]
    for elem in popul:
        if ackleys_path(best[0]) > ackleys_path(elem[0]):
            best = elem
    print("Position: ", best[0])
    print("Function value: ", ackleys_path(best[0]))
    print("Best global value: ", ackleys_path(best[3]))
    if n_times == 2:
        x = np.arange(left_interv, right_interv, 0.05)
        y = np.arange(left_interv, right_interv, 0.05)
        x, y = np.meshgrid(x, y)
        z = ackleys_path([x, y])
        fig = plt.figure()
        ax1 = fig.add_subplot(121, projection='3d')
        ax1.plot_surface(x, y, z, rstride=1, cstride=1, cmap="inferno")
        ax1.scatter(best[0][0], best[0][1], ackleys_path(best[0]), color='red',
s=50, marker='o')
        ax1.set_title('Generation ' + str(iterat) + ': Function Surface')
        ax2 = fig.add_subplot(122, projection='3d')
        ax2.plot_surface(x, y, z, rstride=1, cstride=1, cmap="inferno")
        x_vals = [k[0][0] for k in popul]
        y_vals = [k[0][1] for k in popul]
        z_vals = [ackleys_path(k[0]) for k in popul]
        ax2.scatter(x_vals, y_vals, z_vals, color='red', s=50, marker='o')
        ax2.set_title('Generation ' + str(iterat) + ': Particles Positions')
        plt.show()

def speed_correction(elem, first_coef, second_coef):
    for k in range(len(elem[1])):
        elem[1][k] += first_coef * random.random() * (elem[2][k] - elem[0][k])
```

```
        elem[1][k] += second_coef * random.random() * (elem[3][k] - elem[0][k])
    # No speed limit correction for this problem

def position_correction(elem):
    for k in range(len(elem[0])):
        elem[0][k] += elem[1][k]


if __name__ == '__main__':
    n = 10
    iteration = 500
    quanta = 300
    c1 = 1.5
    c2 = 1.5
    left_interval = -1
    right_interval = 1
    generation = 0
    population = initialization(quanta, n, left_interval, right_interval)
    print_results(population, n, generation, left_interval, right_interval)
    while generation < iteration:
        for i in population:
            if ackleys_path(i[0]) < ackleys_path(i[2]):
                i[2] = deepcopy(i[0])
            if ackleys_path(i[2]) < ackleys_path(i[3]):
                for j in population:
                    j[3] = deepcopy(i[2])
        for quant in population:
            speed_correction(quant, c1, c2)
            position_correction(quant)
        generation += iteration
    print_results(population, n, generation, left_interval, right_interval)
```