

ГУАП

КАФЕДРА № 43

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ  
ПРЕПОДАВАТЕЛЬ

д-р техн. наук, профессор  
\_\_\_\_\_  
должность, уч. степень, звание

\_\_\_\_\_  
подпись, дата

Скобцов Ю.А.  
\_\_\_\_\_  
инициалы, фамилия

## ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

### **Оптимизация функций многих переменных с помощью роевых алгоритмов**

по курсу: Эволюционные методы проектирования программно-информационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №

4132

\_\_\_\_\_  
подпись, дата

Н.И. Карпов  
\_\_\_\_\_  
инициалы, фамилия

Санкт-Петербург 2024

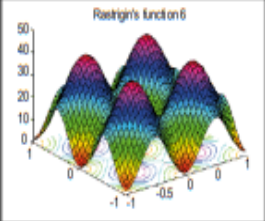
## СОДЕРЖАНИЕ

1	Индивидуальное задание .....	3
2	Краткие теоретические сведения .....	3
3	Результаты выполнения работы.....	4
3.1	Исследование решений при $n=2$ .....	4
3.2	Исследование решений при $n>2$ .....	5
3.3	Исследование решений при разных параметрах.....	6
3.4	Листинг программы .....	7
4	Вывод .....	10

## 1 Индивидуальное задание

Вариант 6:

Необходимо найти минимум многомерной функции с использованием эволюционной стратегии.

6	Rastrigin's function 6	global minimum $f(x)=0; x(i)=0,$ $i=1:n.$	$f_6(x) = 10 \cdot n + \sum_{i=1}^{n-1} (x_i^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i))$ $-5.12 \leq x_i \leq 5.12$ $f_6(x) = 10 \cdot n + \sum (x(i)^2 - 10 \cdot \cos(2 \cdot \pi \cdot x(i))),$ $i=1:n;$	
---	------------------------	---	--	---

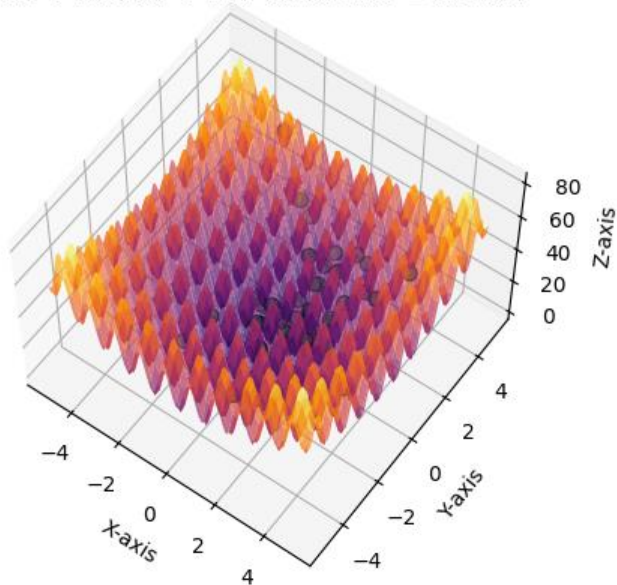
## 2 Краткие теоретические сведения

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.

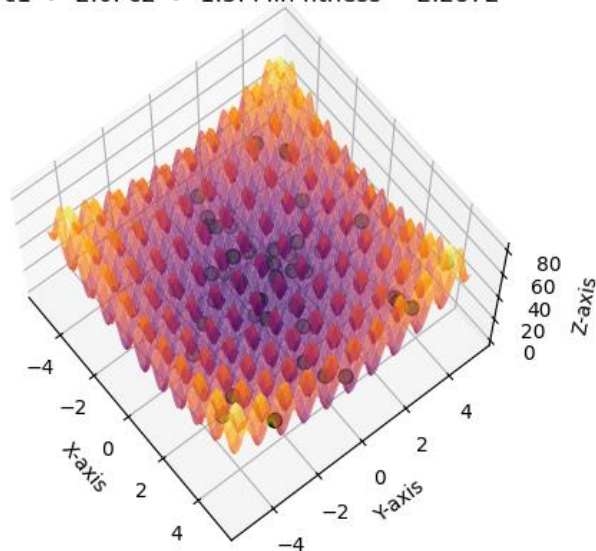
### 3 Результаты выполнения работы

#### 3.1 Исследование решений при $n=2$

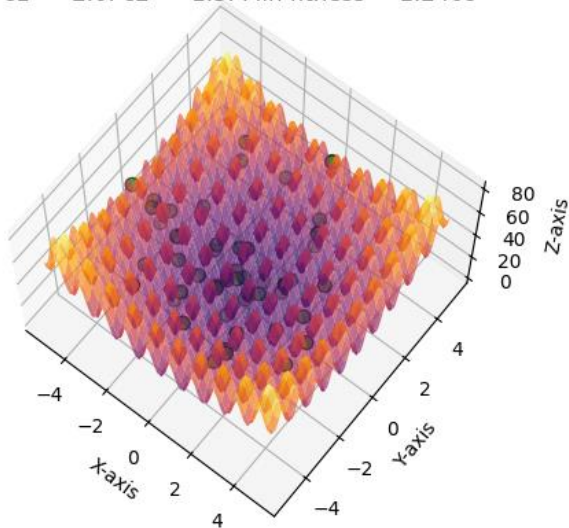
Generation 1. Population size is 50.  
c1 -> 2.0. c2 -> 1.5. Min fitness = 2.3042



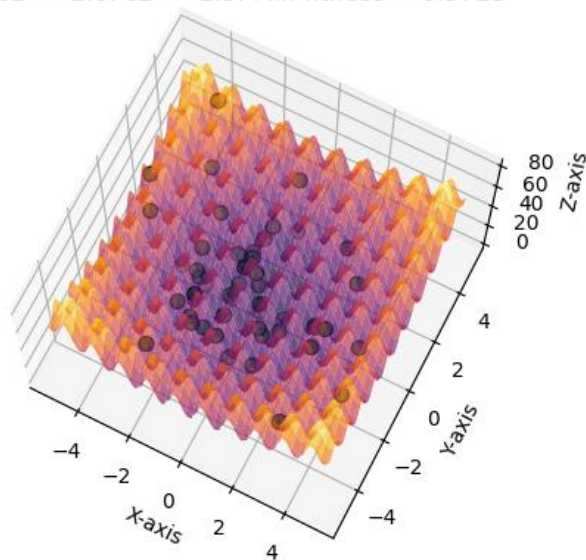
Generation 2. Population size is 50.  
c1 -> 2.0. c2 -> 1.5. Min fitness = 2.2872



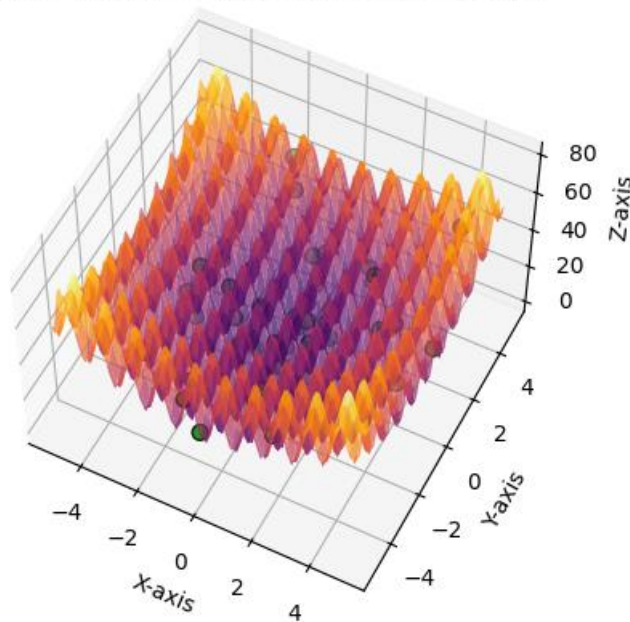
Generation 3. Population size is 50.  
c1 -> 2.0. c2 -> 1.5. Min fitness = 1.2409



Generation 6. Population size is 50.  
c1 -> 2.0. c2 -> 1.5. Min fitness = 0.5713



Generation 7. Population size is 50.  
c1 -> 2.0. c2 -> 1.5. Min fitness = 0.1060



Generation [8/10]	Best solution = 0.023	Current min fitness = 0.595	Mean: 25.550
Generation [9/10]	Best solution = 0.023	Current min fitness = 0.160	Mean: 26.132
Generation [10/10]	Best solution = 0.023	Current min fitness = 1.033	Mean: 26.254

```
=====
Required time: 32.78s. Found answer: 0.022589. Required generations: 10.
n = 2, c1 = 2.0, c2 = 1.5
PS F:\Coding\py\evolutionary-programming\algorithmPS0> []
```

Как можно увидеть из графиков, частицы крайне медленно сходятся к оптимальному значению и остаются раскиданными по локальным минимумам.

Не смотря на это, полученный ответ является весьма неплохим (0.02, по сравнению с 0.10 и 0.05 при использовании генетических алгоритмов и эволюционной стратегии соответственно). Однако такой результат часто зависит от случайности (например, выбранной скорости и начальных точек).

### 3.2 Исследование решений при $n > 2$

При  $n = 3$

Generation [49/50]	Best solution = 0.001	Current min fitness = 0.007	Mean: 16.686
Generation [50/50]	Best solution = 0.001	Current min fitness = 0.004	Mean: 14.625

```
=====
Required time: 0.62s. Found answer: 0.001125. Required generations: 50.
n = 3, c1 = 2.0, c2 = 1.5
```

Т.е. на поиск решения ушло 0,62 секунд и 50 поколений.

При  $n = 10$

Generation [49/50]	Best solution = 22.721	Current min fitness = 24.799	Mean: 88.513
Generation [50/50]	Best solution = 22.721	Current min fitness = 21.522	Mean: 84.631

```
=====
Required time: 0.72s. Found answer: 21.521645. Required generations: 50.
n = 10, c1 = 2.0, c2 = 1.5
```

Т.е. на поиск решения ушло 0,72 секунд и 50 поколений.

Точность решения понижается с увеличением количества измерений, скорость значений меняется незначительно. Для компенсации уменьшения точности можно увеличить размер популяции, количество поколений.

На данных примерах этот алгоритм оказался на порядок точнее, но он может требовать большего числа поколений.

### 3.3 Исследование решений при разных параметрах

В ходе работы было установлено, что размер популяции положительно влияет на точность, но негативно влияет на скорость вычислений:

Размер популяции 1000:

Generation [49/50]	Best solution = 1.850	Current min fitness = 4.296	Mean: 48.761
Generation [50/50]	Best solution = 1.850	Current min fitness = 1.727	Mean: 46.716

```
=====
Required time: 3.22s. Found answer: 1.726603. Required generations: 50.
n = 5, c1 = 2.0, c2 = 1.5
```

Размер популяции 100:

Generation [49/50]	Best solution = 7.349	Current min fitness = 7.243	Mean: 40.011
Generation [50/50]	Best solution = 7.243	Current min fitness = 7.265	Mean: 34.228

```
=====
Required time: 0.30s. Found answer: 7.243027. Required generations: 50.
n = 5, c1 = 2.0, c2 = 1.5
```

### 3.4 Листинг программы

```
import numpy as np
import genalg

POPULATION_SIZE = 100
MAX_GENERATIONS = 50
C1 = 2.
C2 = 1.5
W = 0.9
N = 5

LEFT_EDGE, RIGHT_EDGE = -5.12, 5.12

# Function is Rastrigin's function 6
def fitness_function(x):
    return 10 * len(x) + np.sum(np.power(x, 2) - 10 * np.cos(2 * np.pi *
x))

if __name__ == '__main__':
    pso_optimizer = genalg.AlgorithmPSO(fitness_function, N, POPULA-
TION_SIZE, MAX_GENERATIONS,
                                        LEFT_EDGE, RIGHT_EDGE, C1, C2, W)

    pso_optimizer.start()

import matplotlib.pyplot as plt
import numpy as np

from time import time

class Particle:
    """Particle - solution of the task"""
    def __init__(self, n, left_edge, right_edge, p_id):
        self.p_id = p_id
        self.position = np.random.uniform(left_edge, right_edge, n)
        self.velocity = np.random.uniform(-1, 1, n)
        self.best_position = self.position.copy()
        self.best_value = float('inf')
        self.value = float('inf')

class AlgorithmPSO:
    """
```

```

    Solves task of finding maximum of target function (fitness function)
    using genetic algorithm.
    :param fitness_function: target function
    :param n: number of dimensions
    :param population_size: size of the population
    :param max_generations: number of max available generations
    :param c1: acceleration coefficient for cognitive component
    :param c2: acceleration coefficient for social component
    """
    def __init__(self, fitness_function, n: int, population_size: int,
max_generations: int, left_edge: float,
                right_edge: float, c1=2., c2=2., w=0.9):
        self.fitness_function = fitness_function
        self.population_size = population_size
        self.max_generations = max_generations
        self.left_edge = left_edge
        self.right_edge = right_edge
        self.n = n
        self.c1 = c1
        self.c2 = c2
        self.w = w

        # Particles (array of population_size solutions).
        self.particles = [Particle(n, left_edge, right_edge, i) for i in
range(population_size)]

        # The global best particle.
        self.best_particle = None

        # The global best value.
        self.best_value = float('inf')

        self.fitness_values = np.zeros(shape=(self.population_size, ))

    def start(self):
        generation = 0 # generations counter

        start_time = time()

        while generation < self.max_generations:
            for particle in self.particles:
                # 1. Evaluate fitness.
                particle.value = self.fitness_function(particle.position)

                # 2. Update the personal best.
                if particle.value < particle.best_value:
                    particle.best_value = particle.value
                    particle.best_position = particle.position.copy()

                # 3. Update the global best.
                if particle.value < self.best_value:
                    self.best_value = particle.value
                    self.best_particle = particle.position.copy()

            for particle in self.particles:
                # 4. Update velocities and particles.
                # Get random coefficients for randomisation steps
                r1 = np.random.uniform(0, 1, size=(self.n, ))
                r2 = np.random.uniform(0, 1, size=(self.n, ))

                # Count cognitive and social components.

```



```

cle.position)        cognitive = self.c1 * r1 * (particle.best_position - parti-
cle.position)        social = self.c2 * r2 * (self.best_particle - parti-

                        # Update velocity and position.
particle.velocity = self.w * particle.velocity + cognitive
+ social

                        # Ограничение скорости
max_velocity = (self.right_edge - self.left_edge) / 2
particle.velocity = np.clip(particle.velocity, -
max_velocity, max_velocity)

particle.position = particle.position + particle.velocity

                        # Clip position if it is out of bounds.
particle.position = np.clip(particle.position,
self.left_edge, self.right_edge)
self.fitness_values[particle.p_id] =
self.fitness_function(particle.position)

self.w = max(0.4, self.w * 0.99)

print(f'Generation [{generation +
1}/{self.max_generations}]\t\t' +
      f'Best solution = {self.best_value:.3f}\t\t Current min
fitness = {min(self.fitness_values):.3f}\t\t ' +
      f'Mean: {(sum(self.fitness_values) /
self.population_size):.3f}\n')

if self.n == 2:
    self.draw_plot(generation + 1)

generation += 1

end_time = time()

# Update the best solution in last iteration.
for particle in self.particles:
    particle.value = self.fitness_function(particle.position)

    if particle.value < particle.best_value:
        particle.best_value = particle.value
        particle.best_position = particle.position.copy()

    if particle.value < self.best_value:
        self.best_value = particle.value
        self.best_particle = particle.position.copy()

print('\n', '=' * 100)
print(f'Required time: {end_time - start_time:.2f}s. Found answer:
{self.best_value:4f}. ',
      f'Required generations: {generation}.\n',
      f'n = {self.n}, c1 = {self.c1}, c2 = {self.c2}')

def draw_plot(self, generation):
    dots_n = 100

    x = np.linspace(self.left_edge, self.right_edge, dots_n)
    Z = np.zeros(shape=(dots_n, dots_n))

```

```

for i in range(dots_n):
    for j in range(dots_n):
        Z[i][j] = self.fitness_function(np.array([x[i], x[j]]))

# Plot the surface
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
X, Y = np.meshgrid(x, x)
ax.plot_surface(X, Y, Z, cmap='inferno', alpha=0.7)

x_dots = np.zeros(shape=(self.population_size, ))
y_dots = np.zeros(shape=(self.population_size, ))
z_dots = np.zeros(shape=(self.population_size, ))

for i in range(self.population_size):
    x_dots[i] = self.particles[i].position[0]
    y_dots[i] = self.particles[i].position[1]
    z_dots[i] = self.fitness_values[i]

ax.scatter3D(x_dots, y_dots, z_dots, color='green', marker='o',
s=50, edgecolor='black')

ax.set_xlabel("X-axis")
ax.set_ylabel("Y-axis")
ax.set_zlabel("Z-axis")
ax.set_title(f"Generation {generation}. Population size is
{self.population_size}.\n" +
            f"c1 -> {self.c1}. c2 -> {self.c2}. " +
            f"Min fitness = {min(self.fitness_values):.4f}")

plt.show()

```

## 4 Вывод

В ходе работы была создана программа для решения оптимизационной задачи с использованием роевого алгоритма. Полученное решение является весьма эффективным и превосходит по скорости и точности решения, рассмотренные в предыдущих работах.