

ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

профессор

должность, уч. степень, звание

подпись, дата

Ю.А. Скобцов

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №7

Оптимизация функций многих переменных с помощью роевых алгоритмов

По дисциплине: Эволюционные методы проектирования программно-
информационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.

4134к

подпись, дата

Костяков Н.А.

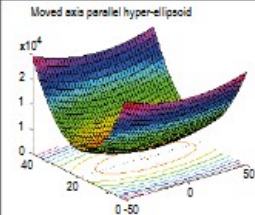
инициалы, фамилия

Санкт-Петербург
2024

Цель работы:

оптимизация функций многих переменных методом роевого интеллекта. Графическое отображение результатов оптимизации.

Вариант 4:

№ вв.	Название	Оптимум	Вид функции	График функции
4	Moved axis parallel hyper-ellipsoid function	global minimum $f(x)=0$; $x(i)=5*i$, $i=1:n$	$f_{1c}(x) = \sum_{i=1}^n 5i \cdot x_i^2$ $-5.12 \leq x_i \leq 5.12$ $f1c(x)=\text{sum}(5*i*x(i)^2),$ $i=1:n;$	

Задание:

1. Разработать программу, использующую РА для нахождения оптимума функции согласно таблице вариантов, приведенной в приложении А. Для всех Benchmark-ов оптимумом является минимум. Программу выполнить на встроенном языке пакета Python.
2. Для $n=2$ вывести на экран график данной функции с указанием найденного экстремума, точек популяции. Для вывода графиков использовать стандартные возможности пакета Python. Предусмотреть возможность пошагового просмотра процесса поиска решения.
3. Исследовать зависимость времени поиска, числа поколений (генераций), точности нахождения решения от основных параметров генетического алгоритма:
 - i. число особей в популяции
 - ii. вероятность мутации.
 - iii. 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.

Выполнение:

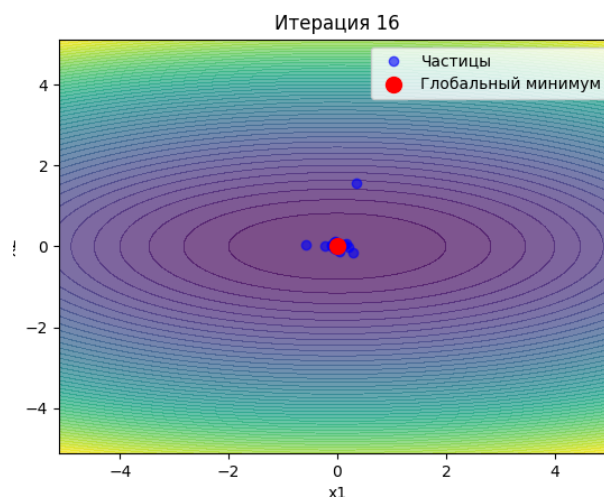
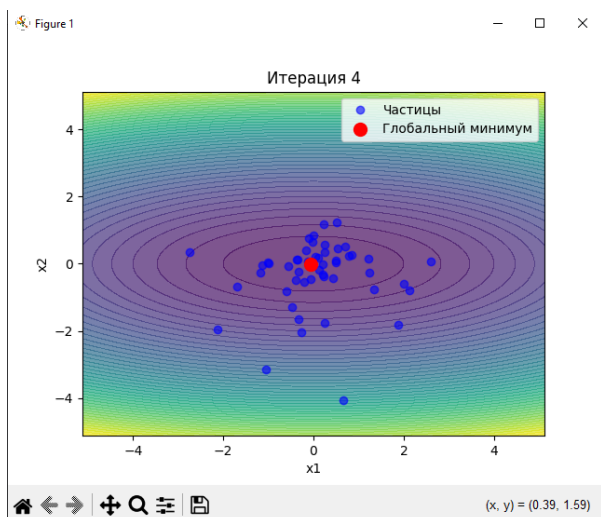
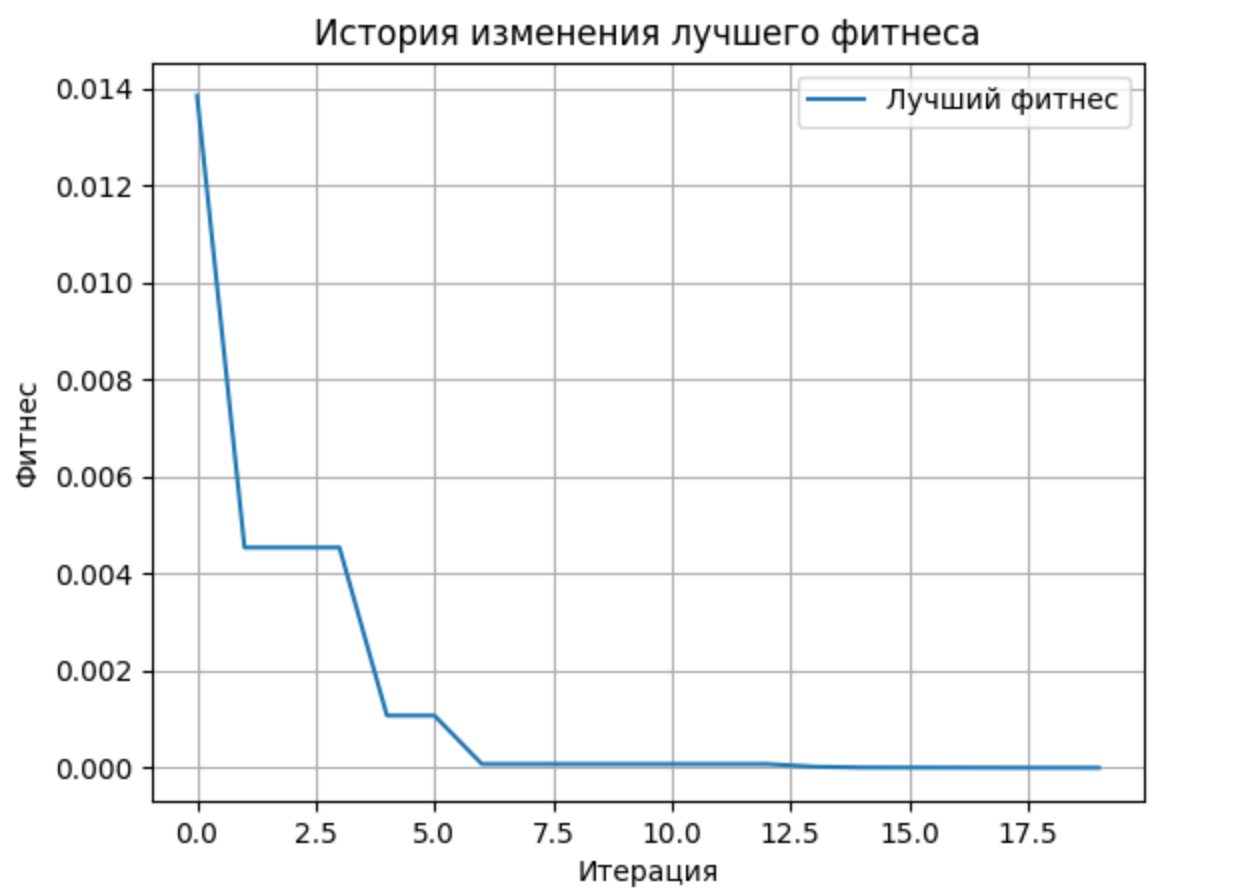


Figure 2



(x, y) = (6.26, 0.00694)

```

● Найденное решение: [-1.29047875e-03 -4.29983766e-06]
  Значение фитнес-функции: 1.6654463478068589e-06
  Время выполнения: 3.58 секунд
  Оптимизация для n=3
  Найденное решение: [-0.00170119  0.00262594 -0.00171334]
  Значение фитнес-функции: 7.65583142103053e-05
  Время выполнения: 0.02 секунд
  Оптимизация для n=5
  Найденное решение: [-7.74739575e-03 -2.07195354e-02 -3.73271298e-05  3.48024250e-03
    -6.44989287e-04]
  Значение фитнес-функции: 0.002838361988922031
  Время выполнения: 0.01 секунд
  Оптимизация для n=10
  Найденное решение: [ 0.24155045 -0.09178162 -0.22962773 -0.18587925  0.04818185 -0.00913888
    0.01344079  0.12623201  0.12987222 -0.04072673]
  Значение фитнес-функции: 2.639728257490284
  Время выполнения: 0.02 секунд

Сравнение времени выполнения:
n=3: Время=0.02 секунд, Фитнес=0.000077
n=5: Время=0.01 секунд, Фитнес=0.002838
n=10: Время=0.02 секунд, Фитнес=2.639728
○ PS D:\Vyzovskoe3-4\7 сем\ЭМПИС\лаб7>

```

Выводы:

В результате проведенной работы была успешно реализована оптимизация многопараметрической функции методом роевого интеллекта (PSO). Полученные графические результаты наглядно продемонстрировали эффективность данного метода в нахождении глобального минимума, что подтверждает его применимость для решения задач оптимизации в многомерных пространствах.

Листинг

```
import numpy as np
import matplotlib.pyplot as plt
import time

# Заданная функция оптимизации (пример: функция Растригина)
def fitness_function(f):
    return sum((5 * i + 1) * (f[i] ** 2) for i in range(len(f)))

# Параметры PSO
def pso_optimization(
    n=2,                # Размерность задачи
    population_size=50,  # Размер роя
    max_iterations=20,   # Максимальное количество итераций
    w=0.5,              # Коэффициент инерции
    c1=1.5,             # Когнитивный коэффициент
    c2=1.5,             # Социальный коэффициент
    x_min=-5.12,        # Нижняя граница поиска
    x_max=5.12          # Верхняя граница поиска
):
    # Инициализация
    particles = np.random.uniform(x_min, x_max, (population_size, n))
    velocities = np.random.uniform(-1, 1, (population_size, n))

    personal_best_positions = np.copy(particles)
    personal_best_scores = np.array([fitness_function(p) for p in particles])

    global_best_position = particles[np.argmin(personal_best_scores)]
    global_best_score = np.min(personal_best_scores)

    # История изменений
    best_scores_history = []
    start_time = time.time()

    for iteration in range(max_iterations):
        for i in range(population_size):
            # Обновление скорости и позиции
            cognitive_component = c1 * np.random.rand(n) * (personal_best_positions[i] -
particles[i])
            social_component = c2 * np.random.rand(n) * (global_best_position -
particles[i])
            velocities[i] = w * velocities[i] + cognitive_component + social_component
            particles[i] += velocities[i]

            # Ограничение в пределах поиска
            particles[i] = np.clip(particles[i], x_min, x_max)

            # Оценка новой позиции
            fitness = fitness_function(particles[i])
            if fitness < personal_best_scores[i]:
```

```

        personal_best_scores[i] = fitness
        personal_best_positions[i] = particles[i]

    # Обновление глобального лучшего результата
    current_best_index = np.argmin(personal_best_scores)
    if personal_best_scores[current_best_index] < global_best_score:
        global_best_score = personal_best_scores[current_best_index]
        global_best_position = personal_best_positions[current_best_index]

    best_scores_history.append(global_best_score)

    # Визуализация для n=2
    if n == 2:
        x1, x2 = np.linspace(x_min, x_max, 200), np.linspace(x_min, x_max, 200)
        X1, X2 = np.meshgrid(x1, x2)
        Z = np.array([[fitness_function([x, y]) for x, y in zip(row_x, row_y)] for
row_x, row_y in zip(X1, X2)])

        plt.cla()
        plt.contourf(X1, X2, Z, cmap='viridis', levels=50, alpha=0.7)
        plt.scatter(particles[:, 0], particles[:, 1], color='blue', label='Частицы',
alpha=0.6)
        plt.scatter(global_best_position[0], global_best_position[1], color='red',
label='Глобальный минимум', s=100)
        plt.title(f'Итерация {iteration + 1}')
        plt.xlabel('x1')
        plt.ylabel('x2')
        plt.legend()
        plt.pause(0.1)

    execution_time = time.time() - start_time

    return global_best_position, global_best_score, best_scores_history,
execution_time

# Выполнение алгоритма для n=2
best_pos, best_score, scores_history, exec_time = pso_optimization(n=2)

# Вывод результата
print(f"Найденное решение: {best_pos}")
print(f"Значение фитнес-функции: {best_score}")
print(f"Время выполнения: {exec_time:.2f} секунд")

# Построение графика изменения глобального лучшего результата
plt.figure()
plt.plot(scores_history, label='Лучший фитнес')
plt.xlabel('Итерация')
plt.ylabel('Фитнес')
plt.title('История изменения лучшего фитнеса')
plt.legend()

```

```
plt.grid()
plt.show()

# Повтор для n=3, n=5, n=10
results = []
for n_dim in [3, 5, 10]:
    print(f"Оптимизация для n={n_dim}")
    best_pos, best_score, scores_history, exec_time = pso_optimization(n=n_dim)
    results.append((n_dim, best_pos, best_score, exec_time))
    print(f"Найденное решение: {best_pos}")
    print(f"Значение фитнес-функции: {best_score}")
    print(f"Время выполнения: {exec_time:.2f} секунд")

# Сравнение результатов
print("\nCравнение времени выполнения:")
for result in results:
    print(f"n={result[0]}: Время={result[3]:.2f} секунд, Фитнес={result[2]:.6f}")
```