
КАФЕДРА

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
РУКОВОДИТЕЛЬ

должность, уч. степень, звание

подпись, дата

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. №

подпись, дата

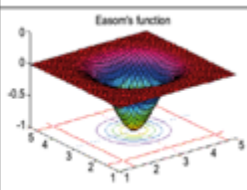
инициалы, фамилия

Санкт-Петербург 2024

Цель работы:

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

Вариант:

14	Easom's function	global minimum $f(x_1, x_2) = -1$; $(x_1, x_2) = (\pi, \pi)$.	$f_{Easo}(x_1, x_2) = -\cos(x_1) \cdot \cos(x_2) \cdot e^{-((x_1 - \pi)^2 + (x_2 - \pi)^2)}$ $-100 \leq x_i \leq 100, i = 1:2$ $fEaso(x_1, x_2) = -\cos(x_1) \cdot \cos(x_2) \cdot \exp(-((x_1 - \pi)^2 + (x_2 - \pi)^2));$	
----	------------------	---	---	---

Easom's function

$fEaso(x_1, x_2) = -\cos(x_1) \cdot \cos(x_2) \cdot \exp(-((x_1 - \pi)^2 + (x_2 - \pi)^2));$

$-100 \leq x_i \leq 100$

$i = 1:2$

global minimum

$f(x_1, x_2) = -1$

$(x_1, x_2) = (\pi, \pi)$

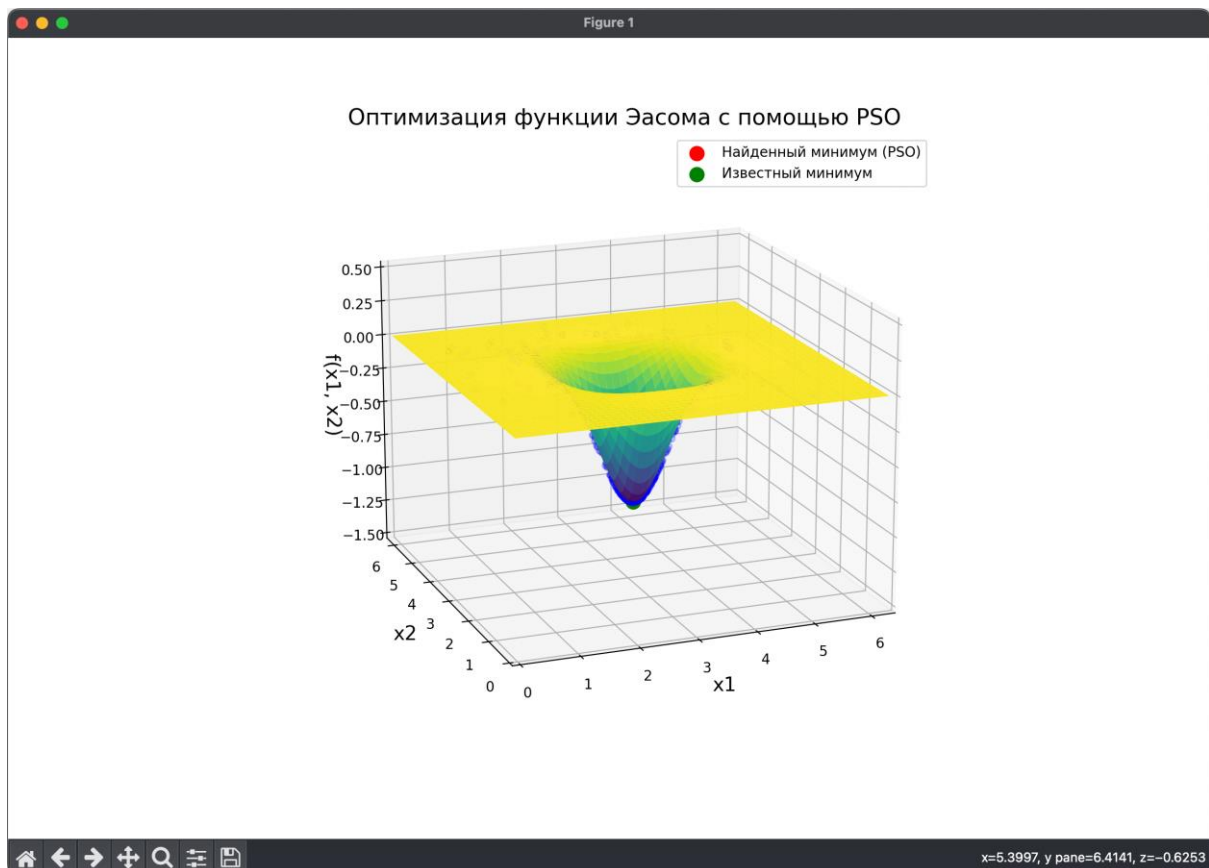
```
-np.cos(x[:, 0]) * np.cos(x[:, 1]) * np.cos(x[:, 2]) * \ np.exp(-((x[:, 0] - np.pi) ** 2 + (x[:, 1] - np.pi) ** 2 + (x[:, 2] - np.pi) ** 2))
```

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:

1. **Инициализация частиц:** Каждая частица начинает с случайной позиции и случайной скорости в заданном диапазоне для x_1 и x_2 (от 0 до 2π).
2. **Оценка значений функции:** Используется функция Эасона ($fEaso$), которая имеет глобальный минимум в точке $(x_1, x_2) = (\pi, \pi)$ со значением -1. Частицы оцениваются по этому критерию.
3. **Когнитивная и социальная составляющие:** Каждая частица обновляет свою скорость, используя собственную лучшую позицию и позицию глобального минимума (найденного всеми частицами), чтобы сближаться к наилучшей найденной точке.
4. **Обновление позиций:** После обновления скорости частицы перемещаются, а их позиции корректируются, чтобы оставаться в пределах допустимого диапазона.
5. **Обновление лучших результатов:** Позиции и значения функции у каждой частицы проверяются, чтобы обновить их личный и глобальный лучший результаты.



Лучшее найденное решение (PSO): $x_1 = 3.141593$, $x_2 = 3.141593$

Значение функции в этой точке (PSO): -1.000000

Известный оптимум: $f(x_1, x_2) = -1$ при $(x_1, x_2) = (\pi, \pi)$

```
population_size = 100      # Количество частиц
max_iterations = 100       # Максимальное количество итераций
w = 0.5                  # Коэффициент инерции
c1 = 1.5                  # Коэффициент когнитивной компоненты (личный опыт)
c2 = 1.5                  # Коэффициент социальной компоненты (опыт группы)
```

```
PROBLEMS OUTPUT ... Code
[Running] /opt/anaconda3/bin/python -u "/Users/andrey/Documents/SUAI/4.1/ЭМППИС/7/emppis7/main23510.py"
Для n = 2:
Лучшее найденное решение: [3.14159266 3.14159265]
Значение функции в этой точке: -1.000000
Время выполнения: 4.6235 секунд

Для n = 4:
Лучшее найденное решение: [3.14159266 3.14159265 3.14159265 3.14159265]
Значение функции в этой точке: -1.000000
Время выполнения: 4.6333 секунд

Для n = 6:
Лучшее найденное решение: [3.14159265 3.14159266 3.14159265 3.14159265 3.14159265 3.14159265]
Значение функции в этой точке: -1.000000
Время выполнения: 4.6572 секунд

Для n = 10:
Лучшее найденное решение: [3.14159265 3.14159265 3.14159266 3.14159266 3.14159266 3.14159265 3.14159265 3.14159266 3.14159266 3.14159266]
Значение функции в этой точке: -1.000000
Время выполнения: 4.6690 секунд

[Done] exited with code=0 in 19.243 seconds
```

Параметры PSO

```
population_size = 500      # Количество частиц
max_iterations = 500       # Максимальное количество итераций
w = 0.5                   # Коэффициент инерции
c1 = 1.5                  # Коэффициент когнитивной компоненты (личный опыт)
c2 = 1.5                  # Коэффициент социальной компоненты (опыт группы)
```

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.

Код программы:

Main:

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

# Функция Эасома
def fEaso(x):
    return -np.cos(x[0]) * np.cos(x[1]) * np.exp(-((x[0] - np.pi) ** 2 + (x[1] - np.pi) ** 2))

# Параметры PSO
population_size = 100      # Количество частиц
max_iterations = 100      # Максимальное количество итераций
w = 0.5                    # Коэффициент инерции
c1 = 1.5                   # Коэффициент когнитивной компоненты (личный опыт)
c2 = 1.5                   # Коэффициент социальной компоненты (опыт группы)
x_min, x_max = 0, 2 * np.pi  # Диапазон для x1 и x2

# Инициализация частиц
particles = np.random.uniform(x_min, x_max, (population_size, 2))
velocities = np.random.uniform(-1, 1, (population_size, 2))

# Инициализация лучшей позиции для каждой частицы и глобальной лучшей позиции
personal_best_positions = np.copy(particles)
personal_best_scores = np.array([fEaso(p) for p in particles])
global_best_position = particles[np.argmin(personal_best_scores)]
global_best_score = np.min(personal_best_scores)

# Построение графика
x1 = np.linspace(x_min, x_max, 200)
x2 = np.linspace(x_min, x_max, 200)
x1, x2 = np.meshgrid(x1, x2)
z = fEaso([x1, x2])

fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Перерисовка поверхности функции
surf = ax.plot_surface(x1, x2, z, cmap='viridis', edgecolor='none')

# Установка пределов осей и стиля
ax.set_xlim([x_min, x_max])
ax.set_ylim([x_min, x_max])
ax.set_zlim([-1.5, 0.5])
ax.view_init(elev=30, azimuth=240)
ax.set_title('Оптимизация функции Эасома с помощью PSO', fontsize=16)
ax.set_xlabel('x1', fontsize=14)
ax.set_ylabel('x2', fontsize=14)
ax.set_zlabel('f(x1, x2)', fontsize=14)
```

```

# Основной цикл PSO
for iteration in range(max_iterations):
    # Обновление позиций и скоростей частиц
    for i in range(population_size):
        # Когнитивная и социальная компоненты
        cognitive_component = c1 * np.random.rand() * (personal_best_positions[i] - particles[i])
        social_component = c2 * np.random.rand() * (global_best_position - particles[i])

        # Обновление скорости
        velocities[i] = w * velocities[i] + cognitive_component + social_component
        particles[i] += velocities[i] # Обновление позиции

        # Ограничение позиций частиц
        particles[i] = np.clip(particles[i], x_min, x_max)

        # Оценка новой позиции
        fitness = fEaso(particles[i])

        # Обновление лучшего личного результата
        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]

    # Визуализация текущих позиций частиц
    ax.scatter(particles[:, 0], particles[:, 1], [fEaso(p) for p in particles], color='blue', alpha=0.2)

    plt.pause(0.1)

# Отображение найденного экстремума
ax.scatter(global_best_position[0], global_best_position[1], global_best_score, color='red', s=100, label='Найденный минимум (PSO)')
ax.scatter(np.pi, np.pi, -1, color='green', s=100, label='Известный минимум')

ax.legend(loc='upper right')
plt.show()

print(f'Лучшее найденное решение (PSO): x1 = {global_best_position[0]:.6f}, x2 = {global_best_position[1]:.6f}')
print(f'Значение функции в этой точке (PSO): {global_best_score:.6f}')
print(f'Известный оптимум: f(x1, x2) = -1 при (x1, x2) = (pi, pi)')

```

main24610:

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

# Параметры PSO
population_size = 500      # Количество частиц
max_iterations = 500      # Максимальное количество итераций
w = 0.5                   # Коэффициент инерции
c1 = 1.5                  # Коэффициент когнитивной компоненты (личный опыт)
c2 = 1.5                  # Коэффициент социальной компоненты (опыт группы)
x_min, x_max = 0, 2 * np.pi  # Диапазон для переменных

# Функция Эасома для n-мерного случая
def fEaso(x):
    return -np.prod(np.cos(x)) * np.exp(-np.sum((x - np.pi) ** 2))

# Добавление функции для улучшения инициализации частиц
def initialize_particles(n):
    return np.random.uniform(x_min, x_max, (population_size, n))

def run_pso(n):
    # Инициализация частиц и их скоростей
    particles = initialize_particles(n) # Используйте новую инициализацию
    velocities = np.random.uniform(-1, 1, (population_size, n))

    # Инициализация лучших позиций
    personal_best_positions = np.copy(particles)
    personal_best_scores = np.array([fEaso(p) for p in particles])
    global_best_position = particles[np.argmin(personal_best_scores)]
    global_best_score = np.min(personal_best_scores)

    start_time = time.time()

    # Основной цикл PSO
    for iteration in range(max_iterations):
        for i in range(population_size):
            cognitive_component = c1 * np.random.rand() * (personal_best_positions[i] - particles[i])
            social_component = c2 * np.random.rand() * (global_best_position - particles[i])
            velocities[i] = w * velocities[i] + cognitive_component + social_component
            particles[i] += velocities[i]
            particles[i] = np.clip(particles[i], x_min, x_max)

            fitness = fEaso(particles[i])
            if fitness < personal_best_scores[i]:
                personal_best_scores[i] = fitness
                personal_best_positions[i] = particles[i]

        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]

execution_time = time.time() - start_time

print(f'Для n = {n}:')
print(f'Лучшее найденное решение: {global_best_position}')
print(f'Значение функции в этой точке: {global_best_score:.6f}')
print(f'Время выполнения: {execution_time:.4f} секунд\n")
return global_best_position, global_best_score, execution_time

# Запуск для различных значений n
results = {}
for n in [2, 4, 6, 10]:
    results[n] = run_pso(n)

```