ГУАП

КАФЕДРА № 43

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| профессор |  |  |  | Ю.А. Скобцов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

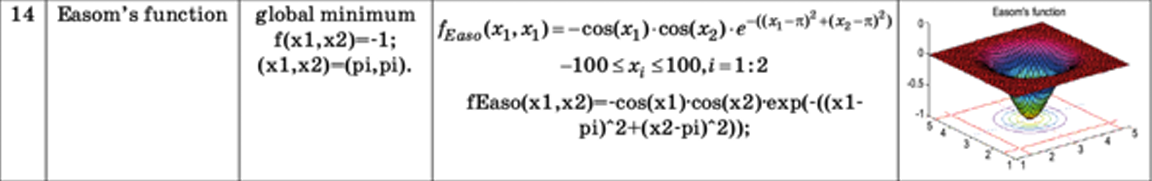
|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №6 |
|  |
| по курсу: ЭВОЛЮЦИОННЫЕ МЕТОДЫ ПРОЕКТИРОВАНИЯ ПРОГРАММНО-ИНФОРМАЦИОННЫХ СИСТЕМ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134 |  |  |  | Д. В. Самарин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

**Вариант 14.**

****

# 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.

1. Инициализация частиц:

Каждая частица получает случайную начальную позицию и скорость. Позиции выбираются из равномерного распределения в пределах диапазона ([0, 2pi]), а скорости из диапазона [-1, 1]. Эти параметры задают начальное состояние всех частиц в пространстве поиска.

2. Оценка значений функции:

Для каждой частицы вычисляется значение функции Эасома (fEaso), которая характеризуется глобальным минимумом в точке (x1, x2) = (pi, pi) со значением (-1). Значения функции используются для определения текущего качества положения частиц.

3. Когнитивная и социальная составляющие:

Скорость каждой частицы обновляется с учетом:

- Когнитивной компоненты: учитывает лучшую позицию самой частицы на текущий момент.

- Социальной компоненты: учитывает лучшую позицию, найденную всей группой частиц.

Оба компонента стремятся направить частицы к глобальному минимуму.

4. Обновление позиций:

После обновления скоростей частицы перемещаются в новое положение. Их координаты корректируются так, чтобы оставаться в пределах допустимого диапазона [0, 2pi].

5. Обновление лучших результатов:

Новые позиции оцениваются по функции Эасома. Если новое значение функции в позиции частицы оказывается лучше её текущего личного лучшего результата, то личный результат обновляется. Также обновляется глобальный лучший результат, если найдено новое глобально лучшее значение.

6. Визуализация процесса:

На каждом шаге отображаются текущие позиции частиц на 3D-графике, демонстрируя их перемещение к минимуму функции.

7. Завершение:

После завершения заданного числа итераций выводится найденная лучшая позиция (x1, x2) и значение функции в этой точке. Также визуально отмечаются найденный минимум и известный глобальный минимум.

# Программа

First\_task

import numpy as np

import matplotlib.pyplot as plt

import time

# Определение функции Эасома

# Функция Эасома имеет известный минимум в точке (π, π) со значением -1

def easom\_function(position):

    x1, x2 = position

    return -np.cos(x1) \* np.cos(x2) \* np.exp(-((x1 - np.pi)\*\*2 + (x2 - np.pi)\*\*2))

# Параметры алгоритма PSO

num\_particles = 100            # Число частиц в популяции

num\_iterations = 100           # Общее число итераций алгоритма

inertia\_weight = 0.5           # Коэффициент инерции для регулировки "тяжести" скорости

cognitive\_coeff = 1.5          # Когнитивный коэффициент (влияние личного опыта частицы)

social\_coeff = 1.5             # Социальный коэффициент (влияние группы на частицу)

lower\_bound, upper\_bound = 0, 2 \* np.pi  # Границы области поиска (x1 и x2)

# Инициализация позиций и скоростей частиц

positions = np.random.uniform(lower\_bound, upper\_bound, (num\_particles, 2))  # Случайные начальные позиции

velocities = np.random.uniform(-1, 1, (num\_particles, 2))  # Случайные начальные скорости

# Лучшая позиция для каждой частицы (личный опыт)

personal\_best\_positions = positions.copy()

personal\_best\_scores = np.array([easom\_function(pos) for pos in positions])  # Значения функции для личных лучших позиций

# Глобальная лучшая позиция (опыт всей группы)

global\_best\_position = positions[np.argmin(personal\_best\_scores)]

global\_best\_score = np.min(personal\_best\_scores)

# Создание сетки для отображения поверхности функции

x\_vals = np.linspace(lower\_bound, upper\_bound, 200)

y\_vals = np.linspace(lower\_bound, upper\_bound, 200)

x\_grid, y\_grid = np.meshgrid(x\_vals, y\_vals)

z\_grid = easom\_function([x\_grid, y\_grid])

# Настройка графика

fig = plt.figure(figsize=(12, 8))

ax = fig.add\_subplot(111, projection='3d')

# Построение поверхности функции Эасома

surface = ax.plot\_surface(x\_grid, y\_grid, z\_grid, cmap='viridis', edgecolor='none')

ax.set\_xlim([lower\_bound, upper\_bound])

ax.set\_ylim([lower\_bound, upper\_bound])

ax.set\_zlim([-1.5, 0.5])

ax.view\_init(elev=30, azim=240)  # Установка угла обзора

ax.set\_title('PSO оптимизация функции Эасома', fontsize=16)

ax.set\_xlabel('x1', fontsize=14)

ax.set\_ylabel('x2', fontsize=14)

ax.set\_zlabel('f(x1, x2)', fontsize=14)

# Основной цикл алгоритма PSO

for iter\_num in range(num\_iterations):

    print(f"Итерация {iter\_num + 1}/{num\_iterations}")  # Вывод текущей итерации

    for particle\_idx in range(num\_particles):

        # Расчёт когнитивной и социальной составляющих для обновления скорости

        cognitive\_term = cognitive\_coeff \* np.random.rand() \* (personal\_best\_positions[particle\_idx] - positions[particle\_idx])

        social\_term = social\_coeff \* np.random.rand() \* (global\_best\_position - positions[particle\_idx])

        # Обновление скорости частицы

        velocities[particle\_idx] = (inertia\_weight \* velocities[particle\_idx] +

                                    cognitive\_term +

                                    social\_term)

        # Обновление позиции частицы и ограничение в пределах области поиска

        positions[particle\_idx] += velocities[particle\_idx]

        positions[particle\_idx] = np.clip(positions[particle\_idx], lower\_bound, upper\_bound)

        # Вычисление значения функции для новой позиции

        current\_fitness = easom\_function(positions[particle\_idx])

        # Если новое значение лучше личного, обновляем личный лучший результат

        if current\_fitness < personal\_best\_scores[particle\_idx]:

            personal\_best\_scores[particle\_idx] = current\_fitness

            personal\_best\_positions[particle\_idx] = positions[particle\_idx]

    # Обновление глобального лучшего результата

    best\_particle\_idx = np.argmin(personal\_best\_scores)

    if personal\_best\_scores[best\_particle\_idx] < global\_best\_score:

        global\_best\_score = personal\_best\_scores[best\_particle\_idx]

        global\_best\_position = personal\_best\_positions[best\_particle\_idx]

    # Вывод информации о текущем лучшем результате

    print(f"Лучший результат на итерации {iter\_num + 1}: f({global\_best\_position[0]:.6f}, {global\_best\_position[1]:.6f}) = {global\_best\_score:.6f}")

    # Визуализация текущего состояния частиц на графике

    ax.scatter(positions[:, 0], positions[:, 1], [easom\_function(pos) for pos in positions], color='blue', alpha=0.3)

    plt.pause(0.1)  # Пауза для обновления визуализации

# Отображение финального результата на графике

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()

# Вывод результатов оптимизации

print(f'Найденный минимум: x1 = {global\_best\_position[0]:.6f}, x2 = {global\_best\_position[1]:.6f}')

print(f'Значение функции в найденной точке: {global\_best\_score:.6f}')

print(f'Глобальный минимум: f(x1, x2) = -1 при x1 = pi, x2 = pi')

Second\_task

import numpy as np

import matplotlib.pyplot as plt

import time

from tabulate import tabulate

# 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)

    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("\n\033[1mРезультаты для n = {}:\033[0m".format(n))

    print("\033[94m\033[1mЛучшее найденное решение:\033[0m", global\_best\_position)

    print("\033[94m\033[1mЗначение функции в этой точке:\033[0m", f"{global\_best\_score:.6f}")

    print("\033[94m\033[1mВремя выполнения:\033[0m", f"{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)

# Табличный вывод результатов

headers = ["Размерность n", "Лучшее решение", "Значение функции", "Время выполнения (сек)"]

table\_data = [

    [n, np.array2string(results[n][0], precision=4, separator=','), f"{results[n][1]:.6f}", f"{results[n][2]:.4f}"]

    for n in results

]

print("\033[1m\nСводная таблица результатов:\033[0m")

print(tabulate(table\_data, headers=headers, tablefmt="grid"))

# Визуализация времени выполнения

dims = list(results.keys())

times = [results[n][2] for n in dims]

plt.figure(figsize=(8, 5))

plt.bar(dims, times, color='skyblue')

plt.xlabel('Размерность n', fontsize=12)

plt.ylabel('Время выполнения (сек)', fontsize=12)

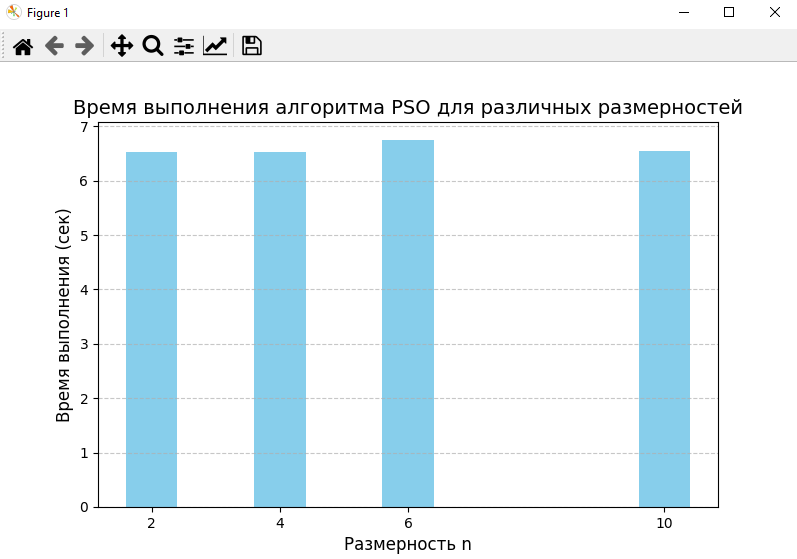
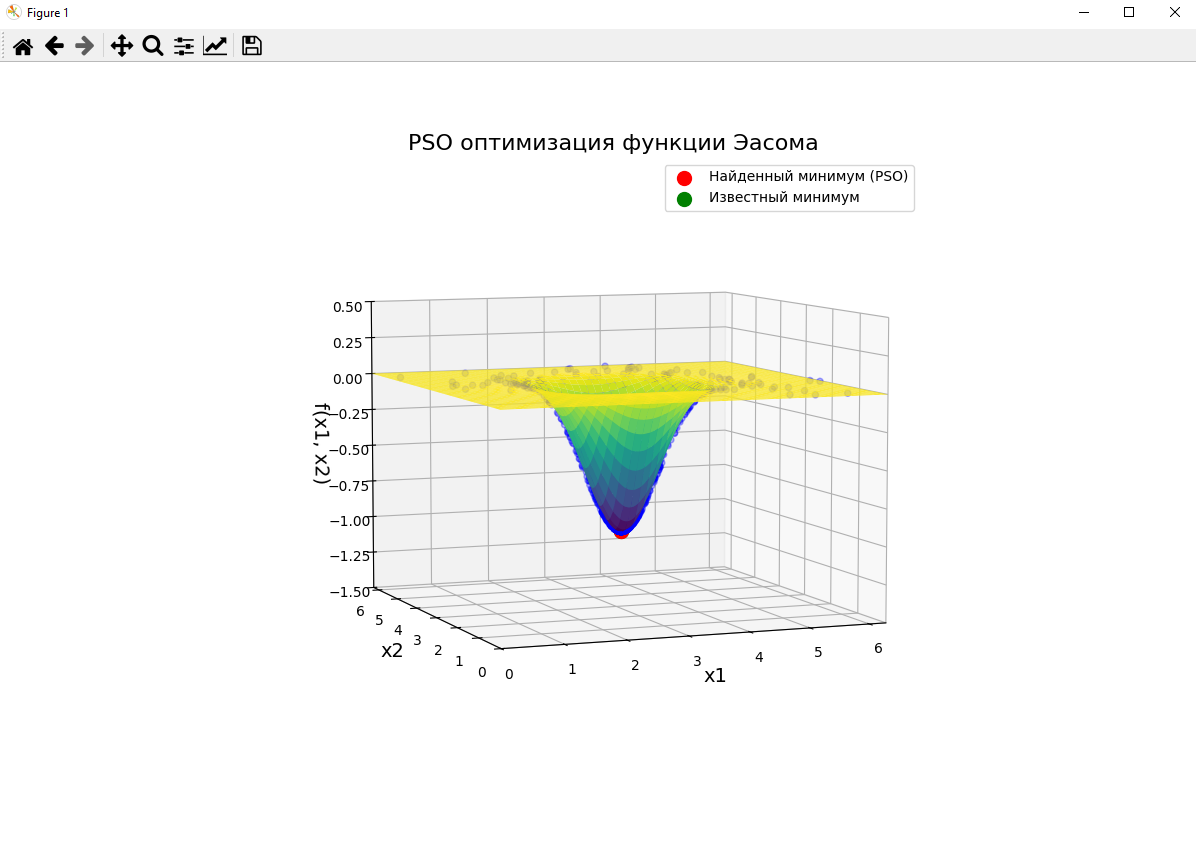
plt.title('Время выполнения алгоритма PSO для различных размерностей', fontsize=14)

plt.xticks(dims)

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.show()

# Результат работы программы



**Результат исследований:**

|  |
| --- |
|  |

# Вывод

В ходе выполнения работы была реализована оптимизация многомерной функции с использованием метода ройного интеллекта (PSO). Графические результаты показали успешность данного подхода в поиске глобального минимума, подтверждая его эффективность и возможность применения для решения задач оптимизации в многомерных пространствах.