ГУАП

КАФЕДРА № 43

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

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

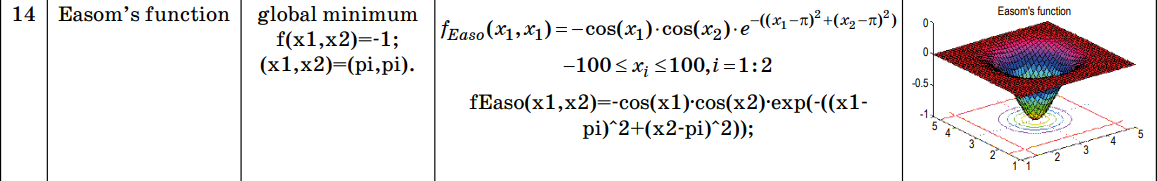
|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №5 |
|  |
| по курсу: ЭВОЛЮЦИОННЫЕ МЕТОДЫ ПРОЕКТИРОВАНИЯ ПРОГРАММНО-ИНФОРМАЦИОННЫХ СИСТЕМ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

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

Санкт-Петербург 2024

**Вариант 14.**

****

# Задание:

1. Создать программу, использующую ЭС для нахождения оптимума функции согласно таблице вариантов, приведенной в приложении А. Для всех Benchmark-ов оптимумом является минимум. Программу выполнить на встроенном языке пакета Matlab (или любом, доступным вам, языке программирования).

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: - число особей в популяции - вероятность мутации. 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, сравнить результаты, скорость работы программы.

# Программа

import numpy as np

import matplotlib.pyplot as plt

import time

# Функция Эасома

def fEaso(x):

    """

    Функция Эасома для двух переменных.

    :param x: Массив с двумя элементами [x1, x2]

    :return: Значение функции Эасома

    """

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

# Параметры эволюционной стратегии

population\_size = 300          # Размер популяции

max\_generations = 300           # Максимальное количество поколений

mutation\_probability = 0.5      # Вероятность мутации

mutation\_sigma = 0.5            # Стандартное отклонение для мутации

no\_improvement\_limit = 100       # Лимит поколений без улучшений для остановки

# Диапазоны для визуализации и ограничений популяции

x\_min\_vis, x\_max\_vis = 0, 2 \* np.pi

# Инициализация начальной популяции в диапазоне от -100 до 100

initial\_population = np.random.uniform(-100, 100, (population\_size, 2))

# Начало замера времени

start\_time = time.time()

best\_fitness\_history = []  # История лучших значений фитнеса

best\_solution = initial\_population[0]

best\_fitness = fEaso(best\_solution)

# Задаем известный экстремум

real\_extremum = np.array([np.pi, np.pi])

real\_extremum\_fitness = fEaso(real\_extremum)

# Построение 3D-графика функции Эасома в пределах от 0 до 2π

x1 = np.linspace(x\_min\_vis, x\_max\_vis, 200)

x2 = np.linspace(x\_min\_vis, x\_max\_vis, 200)

x1, x2 = np.meshgrid(x1, x2)

z = fEaso([x1, x2])

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

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

surf = ax.plot\_surface(x1, x2, z, cmap='viridis', edgecolor='none')

# Установ границы осей от 0 до 2π

ax.set\_xlim([x\_min\_vis, x\_max\_vis])

ax.set\_ylim([x\_min\_vis, x\_max\_vis])

ax.set\_zlim([-1.5, 0.5])  # Пределы для лучшей видимости углубления

ax.view\_init(elev=30, azim=240)

ax.set\_title('Оптимизация функции Эасома с помощью стратегии', fontsize=16)

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

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

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

# Основной цикл эволюционной стратегии

no\_improvement\_count = 0

text = ax.text2D(0.05, 0.95, '', transform=ax.transAxes, fontsize=12, verticalalignment='top')

for generation in range(max\_generations):

    # Оценка популяции

    fitness\_values = np.array([fEaso(ind) for ind in initial\_population])

    # Поиск лучшего решения

    current\_best\_fitness = np.min(fitness\_values)

    best\_idx = np.argmin(fitness\_values)

    # Обновление лучшего решения, если текущее решение лучше

    if current\_best\_fitness < best\_fitness:

        best\_fitness = current\_best\_fitness

        best\_solution = initial\_population[best\_idx]

        no\_improvement\_count = 0  # Сброс при улучшении

    else:

        no\_improvement\_count += 1  # Увеличиваем счетчик без улучшения

    best\_fitness\_history.append(best\_fitness)

    # Проверка условия остановки

    if no\_improvement\_count >= no\_improvement\_limit:

        print(f"Остановка на поколении {generation} из-за отсутствия улучшений за {no\_improvement\_limit} поколений.")

        break

    # Создание новой популяции

    new\_population = []

    for \_ in range(population\_size):

        # Выбор родителя случайным образом

        parent = initial\_population[np.random.choice(population\_size)]

        # Мутация с вероятностью

        if np.random.rand() < mutation\_probability:

            child = parent + np.random.normal(0, mutation\_sigma, 2)

            child = np.clip(child, x\_min\_vis, x\_max\_vis)  # Ограничиваем значения на допустимом диапазоне

        else:

            child = parent

        new\_population.append(child)

    initial\_population = np.array(new\_population)

    # Отображение текущей популяции

    ax.scatter(initial\_population[:, 0], initial\_population[:, 1], fEaso(initial\_population.T),

               color='blue', alpha=0.2)

    # Обновление текста на графике с текущей информацией

    text.set\_text(f'Поколение: {generation+1}/{max\_generations}\n'

                  f'Лучшее значение функции: {best\_fitness:.6f}\n'

                  f'Текущее время: {time.time() - start\_time:.2f} секунд')

    plt.pause(0.1)  # Пауза для пошагового просмотра

# Отображение найденного экстремума (ЭС)

ax.scatter(best\_solution[0], best\_solution[1], best\_fitness,

           color='red', s=100, label='Найденный минимум (ЭС)')

# Отображение реального экстремума

ax.scatter(real\_extremum[0], real\_extremum[1], real\_extremum\_fitness,

           color='green', s=100, label='Известный минимум')

# Отображение точек популяции с меткой

ax.scatter([], [], [], color='blue', alpha=0.2, label='Точки популяции')  # Пустая точка для легенды

# Добавление условных обозначений

ax.legend(loc='upper right')

# Окончание замера времени

end\_time = time.time()

execution\_time = end\_time - start\_time

# Вывод результатов

print(f'Лучшее найденное решение (ЭС): x1 = {best\_solution[0]:.6f}, x2 = {best\_solution[1]:.6f}')

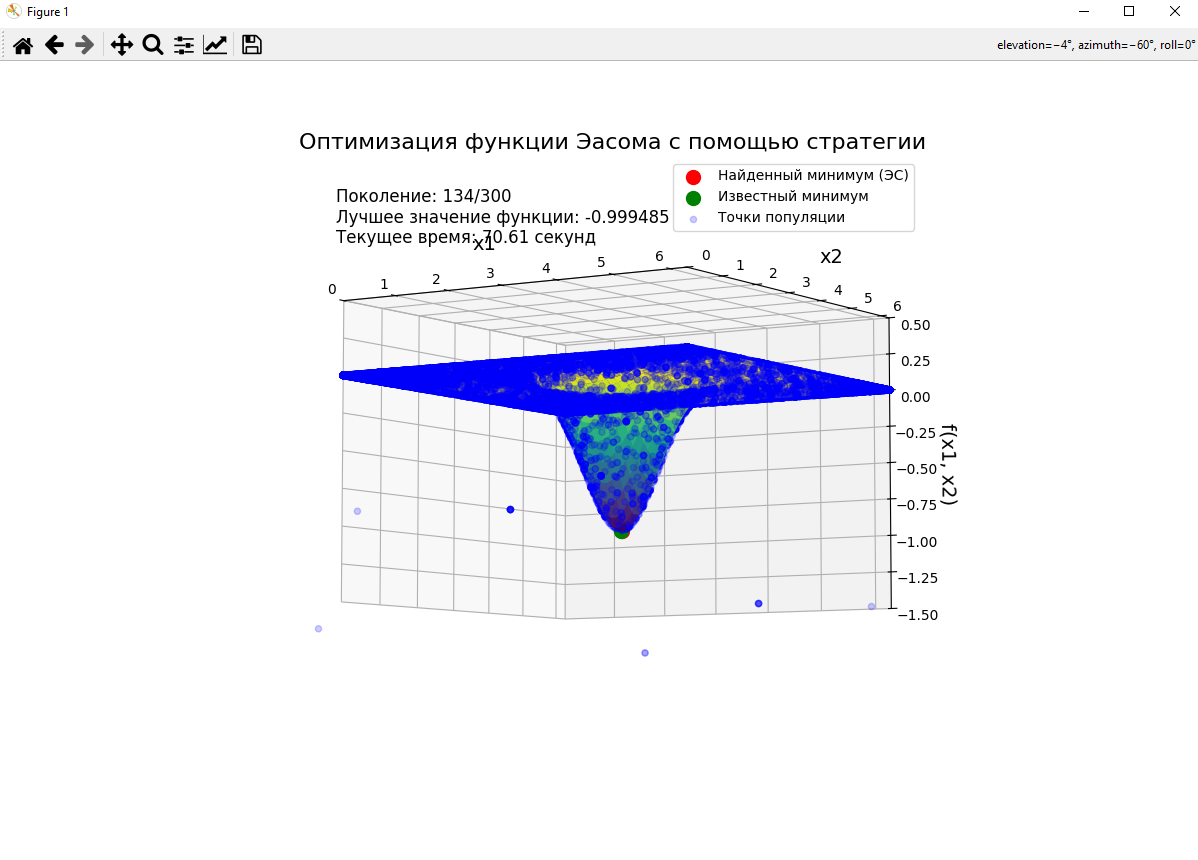
print(f'Значение функции в этой точке (ЭС): {best\_fitness:.6f}')

print(f'Известный оптимум: f(x1,x2) = -1 при (x1,x2) = (pi, pi)')

print(f'Время выполнения программы: {execution\_time:.2f} секунд')

plt.show()

# Результат работы программы



**Результат исследований:**

|  |
| --- |
| Остановка на поколении 134 из-за отсутствия улучшений за 100 поколений.  Лучшее найденное решение (ЭС): x1 = 3.143497, x2 = 3.160018  Значение функции в этой точке (ЭС): -0.999485  Известный оптимум: f(x1,x2) = -1 при (x1,x2) = (pi, pi)  Время выполнения программы: 71.55 секунд |

Результаты демонстрируют, что алгоритм смог приблизиться к оптимуму с высокой точностью, но не достиг его полностью.

# Вывод

Выполнив ЛР5, программа реализовала эволюционную стратегию для нахождения минимума функции. В процессе выполнения программа исследовала пространство решений, постепенно улучшая качество найденных решений. Алгоритм остановился на 134-м поколении из-за отсутствия улучшений за последние 100 поколений. В результате программа нашла приближенное решение: x1=3.143497, x2=3.160018 с функцией f(x1,x2)=−0.999485, что близко к теоретическому минимуму. Также было визуализировано изменение популяции на 3D-графике функции, а общее время выполнения составило 71.55 секунд.