

---

КАФЕДРА

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ  
РУКОВОДИТЕЛЬ

---

должность, уч. степень, звание

---

подпись, дата

---

инициалы, фамилия

Отчет о лабораторной работе №5

Оптимизация многомерных функций с помощью эволюционной стратегии

По дисциплине: Эволюционные методы проектирования программно-  
информационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №

---

подпись, дата

---

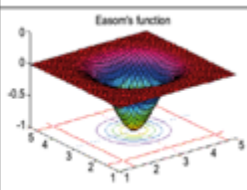
инициалы, фамилия

Санкт-Петербург 2024

### Цель работы:

Оптимизация функций многих переменных модификация методом эволюционной стратегии. 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. 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 - Python (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 - Python. 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.

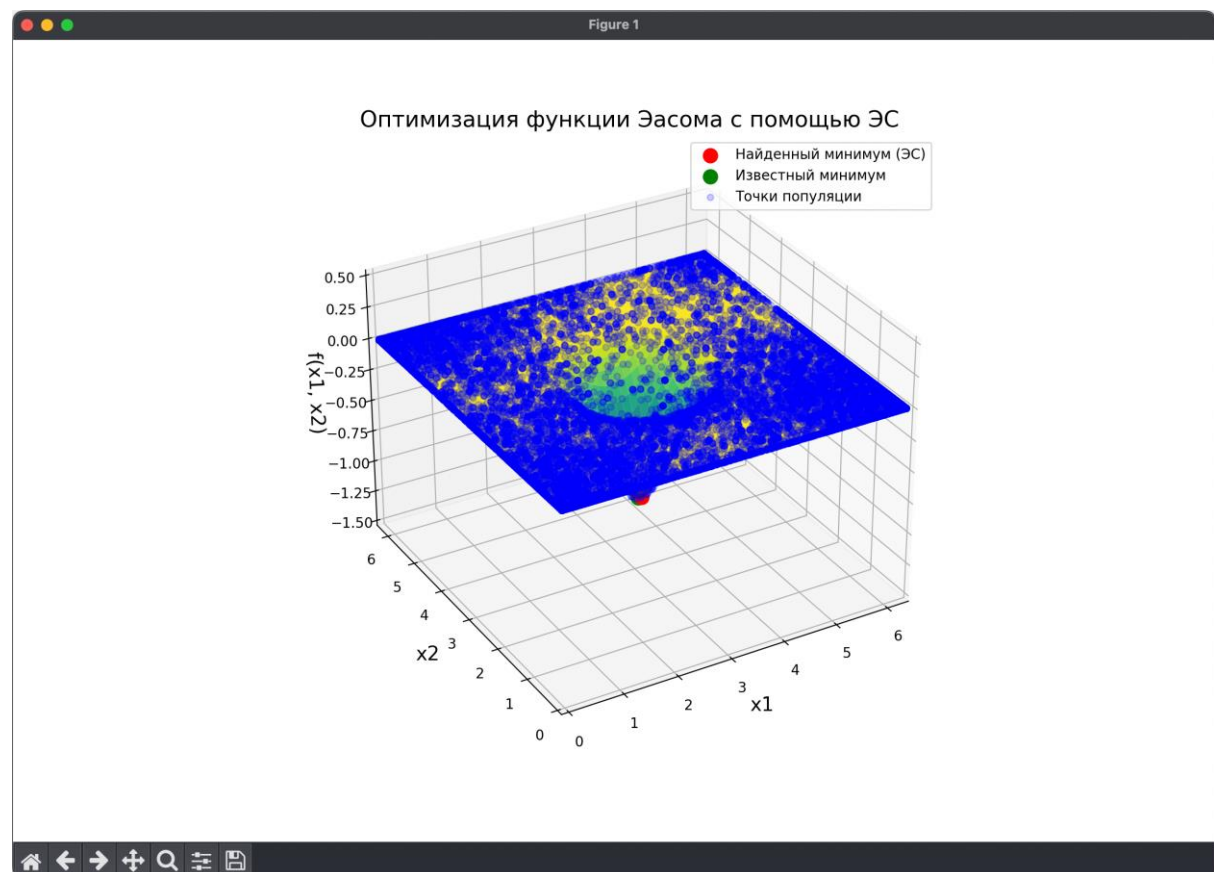
Критерий остановки вычислений – повторение лучшего результата заданное количество раз или достижение популяцией определенного возраста (например, 100 эпох).

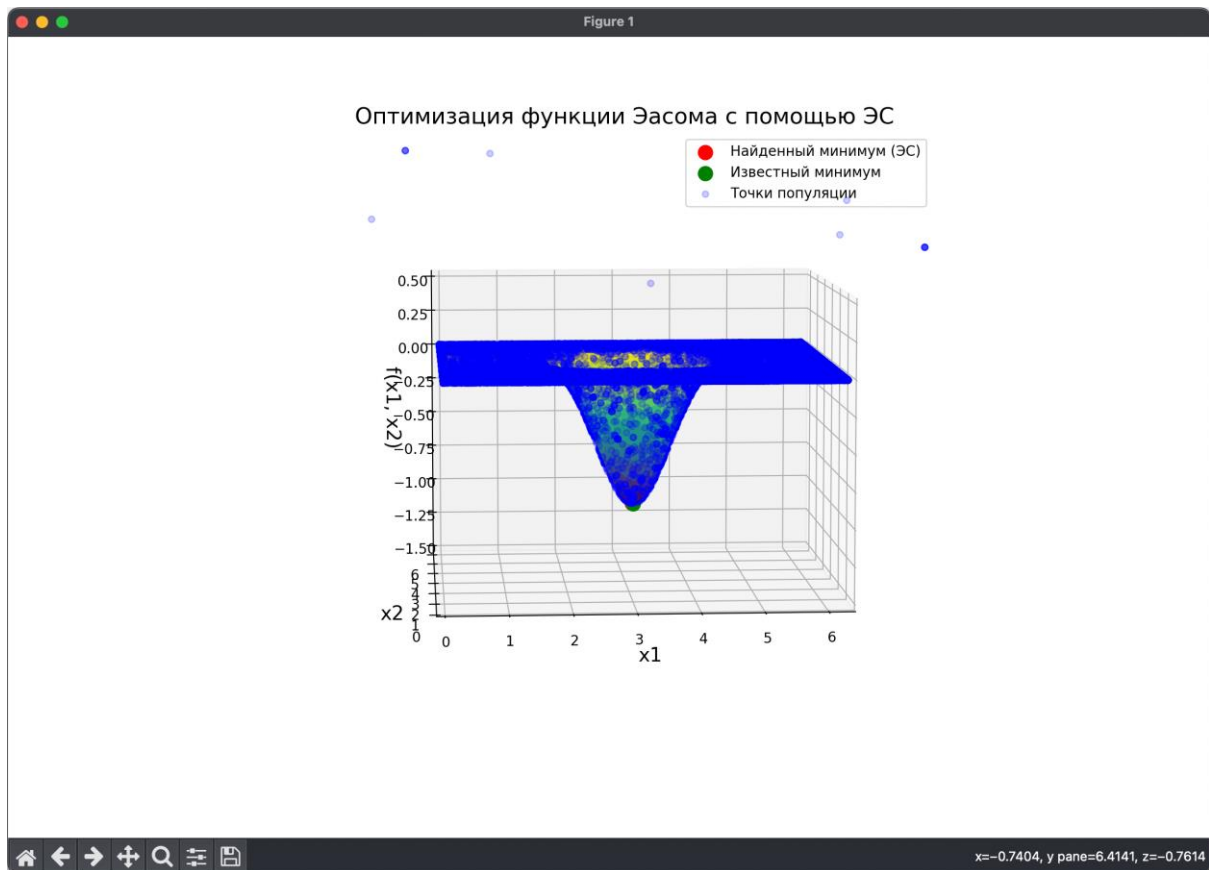
4. Повторить процесс поиска решения для  $n=3$ , сравнить результаты, скорость работы программы.

## Выполнение:

Для  $n = 2$ :

```
PROBLEMS OUTPUT ... Code
[Running] /opt/anaconda3/bin/python -u "/Users/andrey/Documents/SUAI/4.1/ЭМППИС/5/emppis5/main.py"
2024-10-26 18:59:19.706 python[32894:1419841] +[IMKClient subclass]: chose
IMKClient_Legacy
2024-10-26 18:59:19.706 python[32894:1419841] +[IMKInputSession subclass]: chose
IMKInputSession_Legacy
Остановка на поколении 249 из-за отсутствия улучшений за 100 поколений.
Лучшее найденное решение (ЭС):  $x_1 = 3.158508$ ,  $x_2 = 3.149290$ 
Значение функции в этой точке (ЭС):  $-0.999482$ 
Известный оптимум:  $f(x_1, x_2) = -1$  при  $(x_1, x_2) = (\pi, \pi)$ 
Время выполнения программы: 86.38 секунд
```





2024-10-26 19:11:29.888 python[33191:1431583] +[IMKClient subclass]: chose  
IMKClient\_Legacy

2024-10-26 19:11:29.888 python[33191:1431583] +[IMKInputSession subclass]: chose  
IMKInputSession\_Legacy

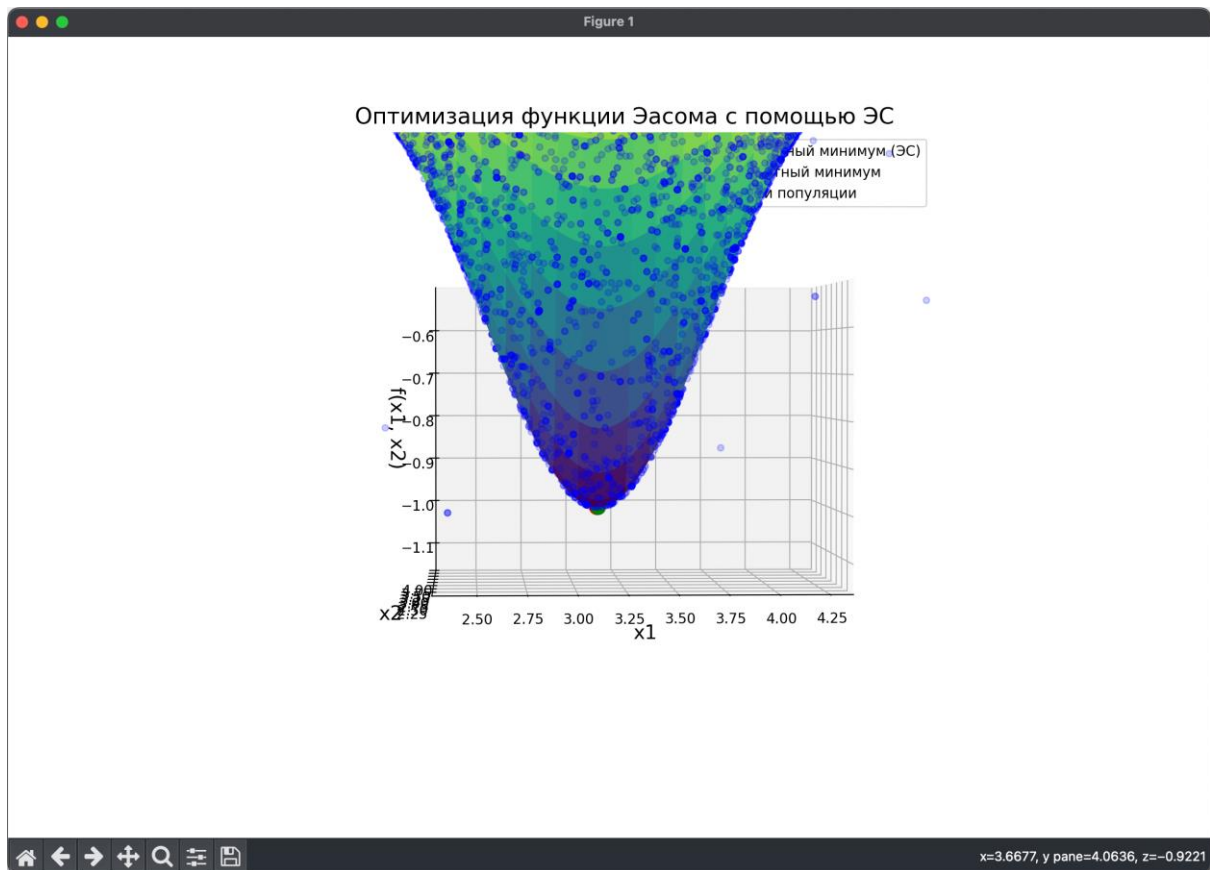
Лучшее найденное решение (ЭС):  $x_1 = 3.128355$ ,  $x_2 = 3.150050$

Значение функции в этой точке (ЭС): -0.999630

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

Время выполнения программы: 81.72 секунд

```
population_size = 300      # Размер популяции
max_generations = 200      # 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
```



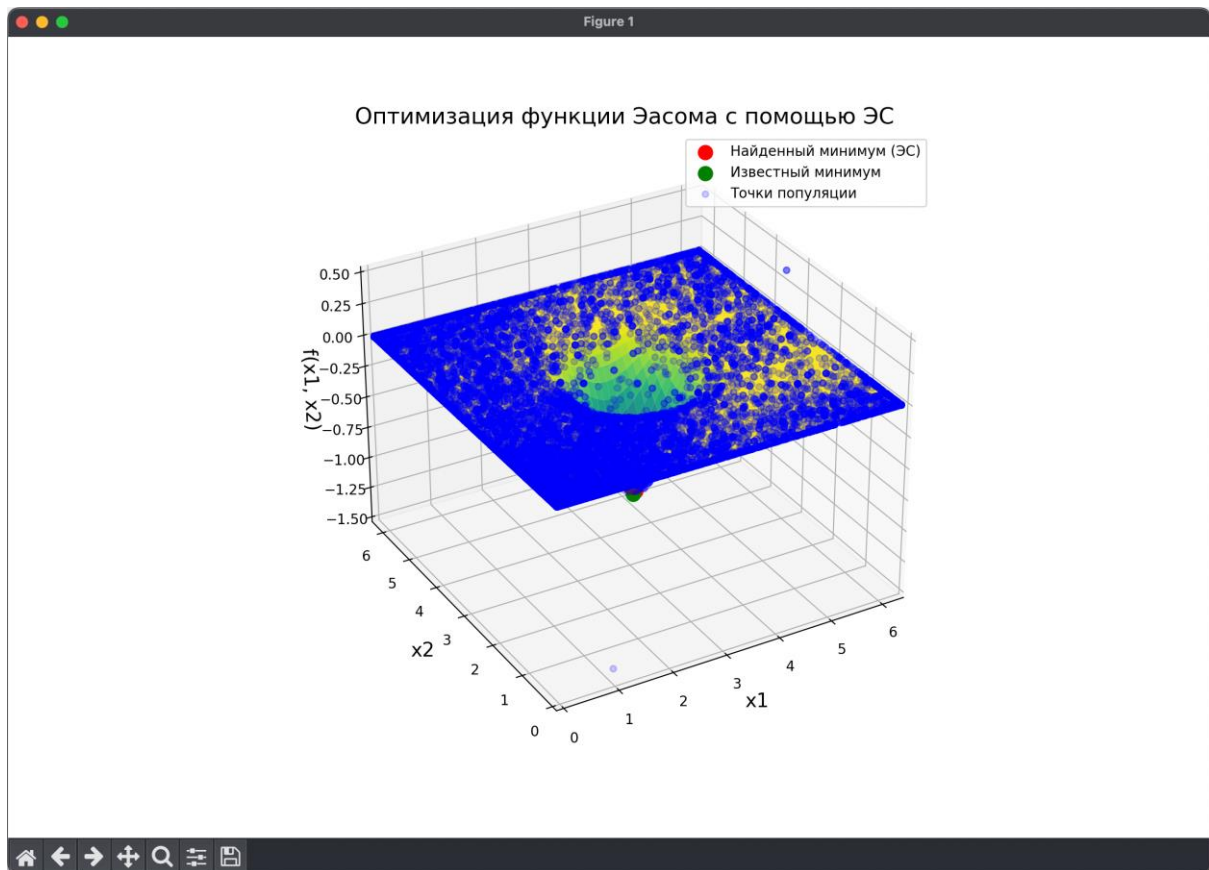
Остановка на поколении 276 из-за отсутствия улучшений за 100 поколений.

Лучшее найденное решение (ЭС):  $x_1 = 3.135188, x_2 = 3.157849$

Значение функции в этой точке (ЭС): -0.999542

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

Время выполнения программы: 114.03 секунд



Лучшее найденное решение (ЭС):  $x_1 = 3.209194$ ,  $x_2 = 3.171506$

Значение функции в этой точке (ЭС): -0.991835

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

Время выполнения программы: 27.32 секунд

```

population_size = 300      # Размер популяции
max_generations = 100     # 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

```

Для  $n = 3$ :

Лучшее найденное решение:  $x_1 = 5.225540$ ,  $x_2 = -2.509082$ ,  $x_3 = -1.976731$

Значение функции в этой точке:  $-0.999928$

Известный оптимум:  $f(x_1, x_2, x_3) = -1$

Время выполнения программы: 3.56 секунд

### **Выводы:**

В данной работе была реализована программа для оптимизации многомерной функции Эасома с использованием эволюционной стратегии. В результате экспериментов были получены оптимальные значения функции, визуализированные на графиках, а также проведено исследование влияния параметров алгоритма, таких как размер популяции и вероятность мутации, на время поиска и точность нахождения решения. Для трехмерного случая был проведен аналогичный анализ, что позволило сравнить эффективность алгоритма в зависимости от размерности задачи.

### **Код программы:**

```
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 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 0 до  $2\pi$ 
initial_population = np.random.uniform(-100, 100, (population_size, 2))

# Начало замера времени
start_time = time.time()
```

```

best_fitness_history = []
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  $2\pi$ 
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  $2\pi$ 
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, azimuth=240)
ax.set_title('Оптимизация функции Эасома с помощью ЭС', 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

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)

    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)
    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)

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_solu-
tion[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()
```

для  $n = 3$ , пример

```
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 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 0 до  $2\pi$ 
initial_population = np.random.uniform(-100, 100, (population_size, 2))

# Начало замера времени
start_time = time.time()

best_fitness_history = []
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  $2\pi$ 
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 границ осей от 0 до  $2\pi$ 
```

```

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, azimuth=240)
ax.set_title('Оптимизация функции Эасома с помощью ЭС', 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

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)

    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)
        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)

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_solu-
tion[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()

```