

ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

профессор

должность, уч. степень, звание

подпись, дата

Ю.А. Скобцов

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.

4134к

подпись, дата

Столяров Н.С.

инициалы, фамилия

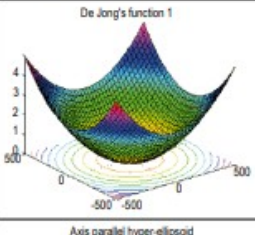

Санкт-Петербург
2024

Цель работы:

Оптимизация функций многих переменных модификация методом эволюционной стратегии. Графическое отображение результатов оптимизации

Вариант 1:

Индивидуальные задания на лабораторную работу №2

| № вв. | Название | Оптимум | Вид функции | График функции |
|----------|----------------------|---|---|--|
| 1 | De Jong's function 1 | global minimum $f(x)=0; x(i)=0,$ $i=1:n.$ | $f_1(x) = \sum_{i=1}^n x_i^2$ $-5.12 \leq x_i \leq 5.12$ $f_1(x) = \sum_{i=1}^n (x(i)^2),$ $i=1:n;$ |  <p>De Jong's function 1</p> |
| 2 | Axis parallel | global minimum | n |  <p>Axis parallel hyper-ellipsoid</p> |

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:

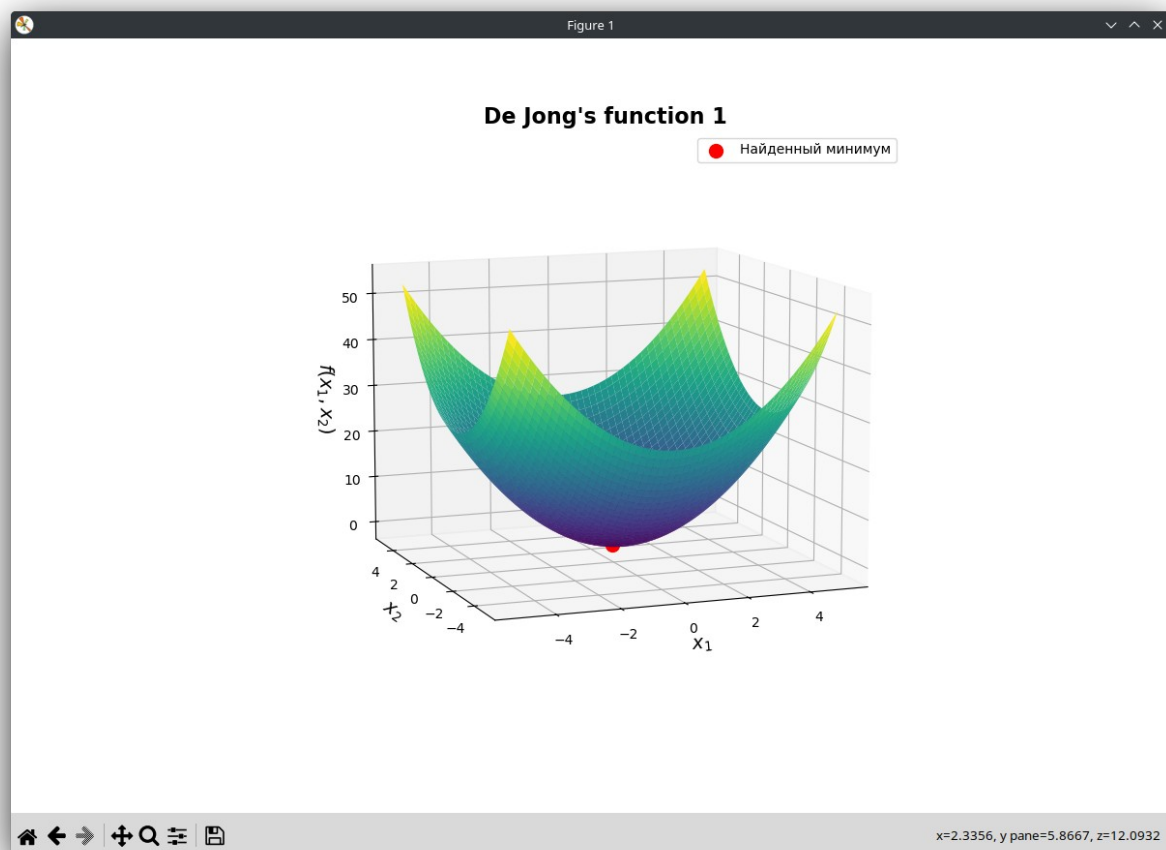
- число особей в популяции
- вероятность мутации.

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.

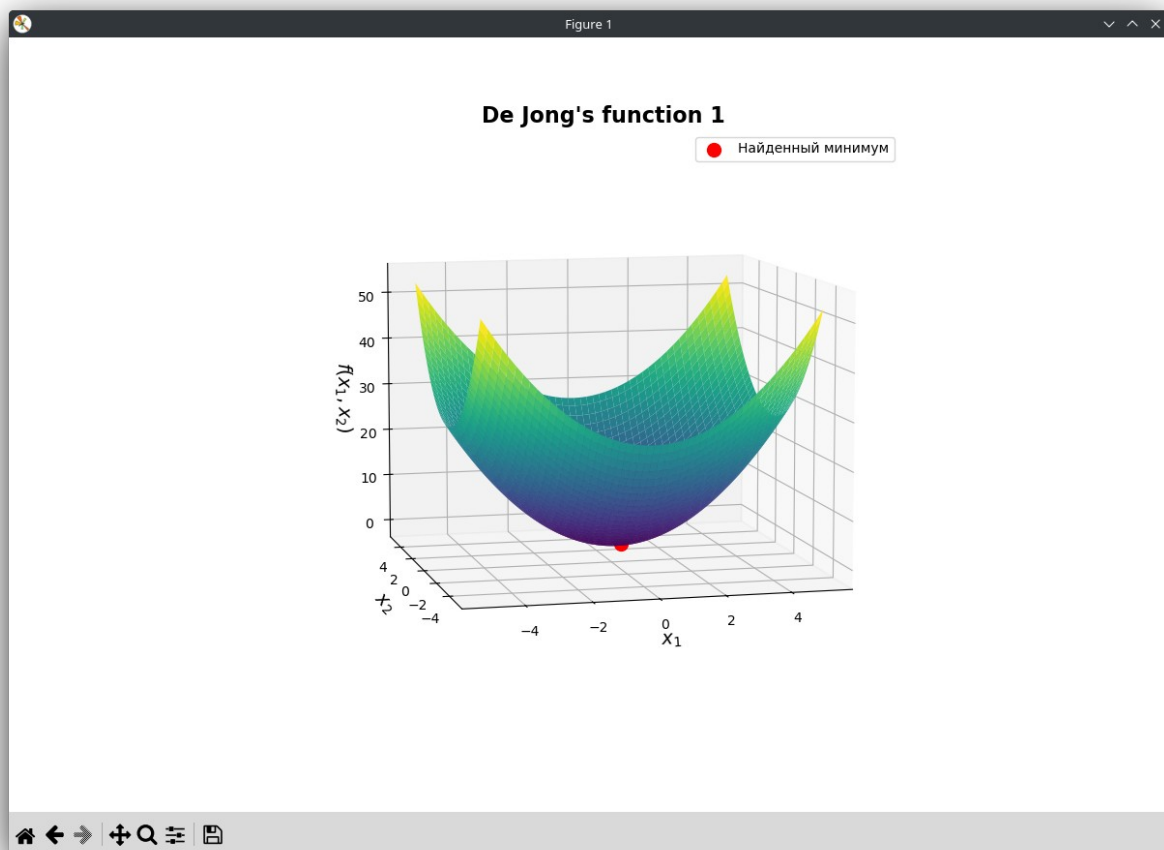
Выполнение:

Для $n = 2$:



```
stolar@stolar-NMH-WCX9:~/PROJECTS/Programming-GUAP/ЭМППИС/5$ python3 main_plot_2.py
Остановка на поколении 109 из-за отсутствия улучшений за 100 поколений.
РЕЗУЛЬТАТЫ ОПТИМИЗАЦИИ:
Функция: De Jong's (f1)
Глобальный минимум функции:  $f(x) = 0$  при  $x = [0, 0]$ 
Лучшее найденное решение (ЭС):  $x = [-0.03871218 \ -0.00156467]$ ,  $f(x) = 0.001501$ 
Количество поколений: 110
Время выполнения программы: 1.22 секунд
stolar@stolar-NMH-WCX9:~/PROJECTS/Programming-GUAP/ЭМППИС/5$
```

Для $n=3$:



```
stolar@stolar-NMH-WCX9:~/PROJECTS/Programming-GUAP/ЭМПИС/5$ python3 main_plot_2.py
Остановка на поколении 103 из-за отсутствия улучшений за 100 поколений.
РЕЗУЛЬТАТЫ ОПТИМИЗАЦИИ:
Функция: De Jong's (f1)
Глобальный минимум функции:  $f(x) = 0$  при  $x = [0, 0]$ 
Лучшее найденное решение (ЭС):  $x = [0.29534134 \ -0.03616651 \ 0.07360631]$ ,  $f(x) = 0.0939$ 
Количество поколений: 104
Время выполнения программы: 1.13 секунд
stolar@stolar-NMH-WCX9:~/PROJECTS/Programming-GUAP/ЭМПИС/5$
```

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.

Листинг

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

# Функция De Jong's (f1)
def fDeJong(x):
    return np.sum(x**2)

# Параметры эволюционной стратегии
population_size = 600      # Размер популяции
max_generations = 1000     # Максимальное количество поколений
mutation_probability = 0.5 # Вероятность мутации
mutation_sigma = 0.1       # Стандартное отклонение для мутации
no_improvement_limit = 100 # Лимит поколений без улучшений для остановки

# Диапазоны для оптимизации
x_min, x_max = -5.12, 5.12
dimensions = 2 # Размерность задачи

# Инициализация начальной популяции
initial_population = np.random.uniform(x_min, x_max, (population_size, dimensions))

# Начало замера времени
start_time = time.time()

best_fitness_history = []
best_solution = initial_population[0]
best_fitness = fDeJong(best_solution)

# Основной цикл эволюционной стратегии
no_improvement_count = 0

for generation in range(max_generations):
    # Оценка популяции
    fitness_values = np.array([fDeJong(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, dimensions)
        child = np.clip(child, x_min, x_max)
    else:
        child = parent
    new_population.append(child)

# print(np.array(new_population))
# print()

initial_population = np.array(new_population)

# Окончание замера времени
end_time = time.time()
execution_time = end_time - start_time

# Построение 3D-графика функции
x1 = np.linspace(x_min, x_max, 200)
x2 = np.linspace(x_min, x_max, 200)
x1, x2 = np.meshgrid(x1, x2)

# Вычисление значений функции для каждого элемента сетки
z = x1**2 + x2**2

fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(x1, x2, z, cmap='viridis', edgecolor='none')
ax.set_title("De Jong's function 1", fontsize=16, fontweight='bold')
ax.set_xlabel('$x_1$', fontsize=14)
ax.set_ylabel('$x_2$', fontsize=14)
ax.set_zlabel('$f(x_1, x_2)$', fontsize=14)
ax.view_init(elev=30, azimuth=240)

# Отображение найденного экстремума
ax.scatter(best_solution[0], best_solution[1], best_fitness,
          color='red', s=100, label='Найденный минимум')
ax.legend(loc='upper right')

```

```
# Вывод результатов в формате задания
print("РЕЗУЛЬТАТЫ ОПТИМИЗАЦИИ:")
print(f"Функция: De Jong's (f1)")
print(f"Глобальный минимум функции:  $f(x) = 0$  при  $x = [0, 0]$ ")
print(f"Лучшее найденное решение (ЭС):  $x = \{best\_solution\}$ ,  $f(x) = \{best\_fitness:.6f\}$ ")
print(f"Количество поколений: {generation + 1}")
print(f"Время выполнения программы: {execution_time:.2f} секунд")

plt.show()
```