МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №  43

ОТЧЁТ

ЗАЩИЩЁН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

### профессор                                   Скобцов Ю.А.

должность, уч. Степень, звание   подпись, дата           инициалы, фамилия

ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ №5.

Оптимизация многомерных функций с помощью эволюционной стратегии.

по курсу: Эволюционные методы проектирования программно-информационных систем

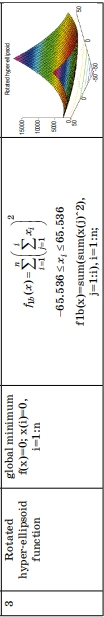
РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. 4136                                                                                Бобрович Н. С.

                                                                         подпись, дата                      инициалы, фамилия

Санкт-Петербург 2024

1. **Задание:**

Вариант 3:  
Скриншот 17-10-2024 230645

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.

1. **Листинг программы:**

import numpy as np

from matplotlib import pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

from matplotlib.animation import FuncAnimation

import time

# Функция Rotated Hyper-Ellisoid Function

def rotated\_hyper\_ellipsoid\_function(X):

n = X.shape[1]

result = 0

for i in range(n):

sum\_x = np.sum(X[:, :i+1], axis=1)

result += np.square(sum\_x)

return result

# Генетический алгоритм

class GeneticAlgorithm:

def \_\_init\_\_(self, population\_size, mutation\_probability, max\_generations, elite\_percentage, min\_value=-65.536, max\_value=65.536):

self.population\_size = population\_size

self.mutation\_probability = mutation\_probability

self.max\_generations = max\_generations

self.elite\_percentage = elite\_percentage

self.min\_value = min\_value

self.max\_value = max\_value

# Генерация начальной популяции

def generate\_initial\_population(self, n):

population = np.random.uniform(low=self.min\_value, high=self.max\_value, size=(self.population\_size, n))

return population

# Турнирный отбор

def tournament\_selection(self, population, fitness\_values):

new\_population = []

for \_ in range(self.population\_size):

candidate\_indices = np.random.choice(population.shape[0], size=2, replace=False)

candidates = population[candidate\_indices]

fitnesses = fitness\_values[candidate\_indices]

best\_candidate\_index = np.argmin(fitnesses)

new\_population.append(candidates[best\_candidate\_index])

return np.array(new\_population)

# Одноточечный кроссовер

def single\_point\_crossover(self, parent1, parent2):

crossover\_point = np.random.randint(1, len(parent1))

child1 = np.concatenate((parent1[:crossover\_point], parent2[crossover\_point:]))

child2 = np.concatenate((parent2[:crossover\_point], parent1[crossover\_point:]))

return child1, child2

# Мутация

def mutate(self, individual):

mutated\_individual = individual.copy()

for i in range(len(individual)):

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

mutated\_individual[i] = np.random.uniform(low=self.min\_value, high=self.max\_value)

return mutated\_individual

# Основная функция генетического алгоритма

def run(self, n):

population = self.generate\_initial\_population(n)

best\_fitness = float('inf')

best\_solution = None

generation\_count = 0

convergence\_counter = 0

fitness\_history = []

while generation\_count < self.max\_generations and convergence\_counter < 10:

fitness\_values = rotated\_hyper\_ellipsoid\_function(population)

# Найдем лучшее решение в текущей популяции

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

current\_best\_index = np.argmin(fitness\_values)

current\_best\_solution = population[current\_best\_index]

if current\_best\_fitness == best\_fitness:

convergence\_counter += 1

else:

convergence\_counter = 0

if current\_best\_fitness < best\_fitness:

best\_fitness = current\_best\_fitness

best\_solution = current\_best\_solution

# Сохраняем историю лучших решений

fitness\_history.append(best\_fitness)

# Отбор родителей методом турнира

parents = self.tournament\_selection(population, fitness\_values)

# Кроссовер и мутация

next\_population = []

for i in range(0, self.population\_size, 2):

parent1 = parents[i]

parent2 = parents[(i + 1) % self.population\_size]

child1, child2 = self.single\_point\_crossover(parent1, parent2)

next\_population.extend([child1, child2])

# Добавляем элитных особей

elite\_count = int(self.elite\_percentage \* self.population\_size)

elite\_indices = np.argsort(fitness\_values)[:elite\_count]

next\_population[-elite\_count:] = population[elite\_indices]

# Применяем мутацию к новым особям

next\_population = np.array([self.mutate(individual) for individual in next\_population])

population = next\_population

generation\_count += 1

return best\_solution, best\_fitness, fitness\_history, generation\_count

# Визуализация функции и найденного экстремума для n=2

def visualize\_function\_and\_extremum(n, solution, fitness\_history, generations):

fig = plt.figure(figsize=(12, 6))

ax1 = fig.add\_subplot(121, projection='3d' if n==2 else 'rectilinear')

ax2 = fig.add\_subplot(122)

# Построим поверхность функции

if n == 2:

x = y = np.arange(-65.536, 65.537, 0.25)

X, Y = np.meshgrid(x, y)

Z = rotated\_hyper\_ellipsoid\_function(np.stack((X.ravel(), Y.ravel()), axis=1)).reshape(X.shape)

ax1.plot\_surface(X, Y, Z, cmap='viridis', alpha=0.7)

# Отображаем точки популяции

population = np.random.uniform(low=-65.536, high=65.536, size=(50, 2))

z\_population = rotated\_hyper\_ellipsoid\_function(population).reshape(-1, 1)

ax1.scatter(population[:, 0], population[:, 1], z\_population, c='r', marker='o', s=20)

# Отображаем найденную точку минимума

ax1.scatter(solution[0], solution[1], fitness\_history[-1], c='g', marker='\*', s=200)

elif n == 3:

ax1.set\_title("График функции для n=3 недоступен")

# График изменения лучшей приспособленности

ax2.plot(range(generations), fitness\_history)

ax2.set\_xlabel('Поколения')

ax2.set\_ylabel('Лучшая приспособленность')

ax2.grid(True)

plt.show()

# Исследование влияния параметров

def investigate\_parameters():

population\_sizes = [10, 20, 30, 40, 50]

mutation\_probabilities = [0.01, 0.05, 0.1, 0.15, 0.2]

results = {}

for population\_size in population\_sizes:

for mutation\_probability in mutation\_probabilities:

print(f"Исследование для population\_size={population\_size}, mutation\_probability={mutation\_probability}")

start\_time = time.time()

ga = GeneticAlgorithm(population\_size=population\_size, mutation\_probability=mutation\_probability, max\_generations=100, elite\_percentage=0.1)

best\_solution, best\_fitness, fitness\_history, generations = ga.run(n=2)

end\_time = time.time()

execution\_time = end\_time - start\_time

results[(population\_size, mutation\_probability)] = {

'execution\_time': execution\_time,

'generations': generations,

'accuracy': best\_fitness

}

print(f"Результат: Время выполнения {execution\_time:.2f} сек., поколений {generations}, точность {best\_fitness:.8f}\n")

return results

if \_\_name\_\_ == "\_\_main\_\_":

# Выполнение для n=2

ga = GeneticAlgorithm(population\_size=50, mutation\_probability=0.1, max\_generations=100, elite\_percentage=0.1)

best\_solution, best\_fitness, fitness\_history, generations = ga.run(n=2)

visualize\_function\_and\_extremum(n=2, solution=best\_solution, fitness\_history=fitness\_history, generations=generations)

# Исследование параметров

results = investigate\_parameters()

# Выполнение для n=3

ga\_n3 = GeneticAlgorithm(population\_size=50, mutation\_probability=0.1, max\_generations=100, elite\_percentage=0.1)

start\_time\_n3 = time.time()

best\_solution\_n3, best\_fitness\_n3, fitness\_history\_n3, generations\_n3 = ga\_n3.run(n=3)

end\_time\_n3 = time.time()

execution\_time\_n3 = end\_time\_n3 - start\_time\_n3

print(f"\nРезультат для n=3:")

print(f"Время выполнения: {execution\_time\_n3:.2f} сек.")

print(f"Поколений: {generations\_n3}")

print(f"Точность: {best\_fitness\_n3:.8f}")

1. **Результат выполнения программы:**

Исследование для population\_size=10, mutation\_probability=0.01

Результат: Время выполнения 0.00 сек., поколений 11, точность 1146.70611691

Исследование для population\_size=10, mutation\_probability=0.05

Результат: Время выполнения 0.01 сек., поколений 19, точность 92.53883062

Исследование для population\_size=10, mutation\_probability=0.1

Результат: Время выполнения 0.01 сек., поколений 16, точность 10.12988118

Исследование для population\_size=10, mutation\_probability=0.15

Результат: Время выполнения 0.00 сек., поколений 16, точность 1.35709247

Исследование для population\_size=10, mutation\_probability=0.2

Результат: Время выполнения 0.01 сек., поколений 18, точность 41.34744066

Исследование для population\_size=20, mutation\_probability=0.01

Результат: Время выполнения 0.01 сек., поколений 13, точность 107.68561058

Исследование для population\_size=20, mutation\_probability=0.05

Результат: Время выполнения 0.01 сек., поколений 20, точность 16.30130337

Исследование для population\_size=20, mutation\_probability=0.1

Результат: Время выполнения 0.01 сек., поколений 24, точность 54.06141814

Исследование для population\_size=20, mutation\_probability=0.15

Результат: Время выполнения 0.02 сек., поколений 27, точность 5.60181970

Исследование для population\_size=20, mutation\_probability=0.2

Результат: Время выполнения 0.01 сек., поколений 12, точность 6.19254092

Исследование для population\_size=30, mutation\_probability=0.01

Результат: Время выполнения 0.01 сек., поколений 15, точность 26.07903708

Исследование для population\_size=30, mutation\_probability=0.05

Результат: Время выполнения 0.02 сек., поколений 18, точность 0.53474608

Исследование для population\_size=30, mutation\_probability=0.1

Результат: Время выполнения 0.01 сек., поколений 15, точность 73.07523401

Исследование для population\_size=30, mutation\_probability=0.15

Результат: Время выполнения 0.03 сек., поколений 34, точность 3.00056669

Исследование для population\_size=30, mutation\_probability=0.2

Результат: Время выполнения 0.08 сек., поколений 92, точность 0.03334988

Исследование для population\_size=40, mutation\_probability=0.01

Результат: Время выполнения 0.01 сек., поколений 13, точность 1.64068908

Исследование для population\_size=40, mutation\_probability=0.05

Результат: Время выполнения 0.02 сек., поколений 23, точность 5.38461378

Исследование для population\_size=40, mutation\_probability=0.1

Результат: Время выполнения 0.02 сек., поколений 17, точность 2.50863627

Исследование для population\_size=40, mutation\_probability=0.15

Результат: Время выполнения 0.04 сек., поколений 30, точность 1.00570572

Исследование для population\_size=40, mutation\_probability=0.2

Результат: Время выполнения 0.03 сек., поколений 29, точность 1.50668718

Исследование для population\_size=50, mutation\_probability=0.01

Результат: Время выполнения 0.02 сек., поколений 13, точность 1.69840705

Исследование для population\_size=50, mutation\_probability=0.05

Результат: Время выполнения 0.02 сек., поколений 17, точность 5.24748892

Исследование для population\_size=50, mutation\_probability=0.1

Результат: Время выполнения 0.03 сек., поколений 22, точность 3.90731902

Исследование для population\_size=50, mutation\_probability=0.15

Результат: Время выполнения 0.03 сек., поколений 25, точность 1.45608072

Исследование для population\_size=50, mutation\_probability=0.2

Результат: Время выполнения 0.03 сек., поколений 21, точность 0.55933323

Результат для n=3:

Время выполнения: 0.03 сек.

Поколений: 25

Точность: 3.76905395

