

ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

профессор

должность, уч. степень, звание

подпись, дата

Ю.А. Скобцов

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

Оптимизация многомерных функций с помощью ГА

по дисциплине: Эволюционные методы проектирования программно-информационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.

4134к

подпись, дата

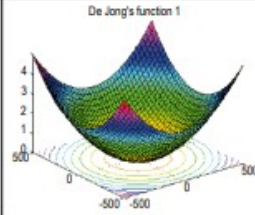

Столяров Н.С.

инициалы, фамилия

Санкт-Петербург
2024

Цель работы: модификация представления хромосомы и операторов рекомбинации ГА для оптимизации многомерных функций. Графическое отображение результатов оптимизации

Индивидуальные задания на лабораторную работу №2

№ вв.	Название	Оптимум	Вид функции	График функции
1	De Jong's function 1	global minimum $f(x)=0; x(i)=0,$ $i=1:n.$	$f_1(x) = \sum_{i=1}^n x_i^2$ $-5.12 \leq x_i \leq 5.12$ $f1(x)=\sum(x(i)^2),$ $i=1:n;$	 <p>De Jong's function 1</p>
2	Axis parallel	global minimum	n	 <p>Axis parallel hyper-ellipsoid</p>

Задание:

1. Создать программу, использующую ГА 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.
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. Повторить нахождение решения с использованием стандартного Genetic Algorithm toolbox. Сравнить полученные результаты.
4. Исследовать зависимость времени поиска, числа поколений (генераций), точности нахождения решения от основных параметров

генетического алгоритма:

- число особей в популяции
- вероятность кроссинговера, мутации.

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

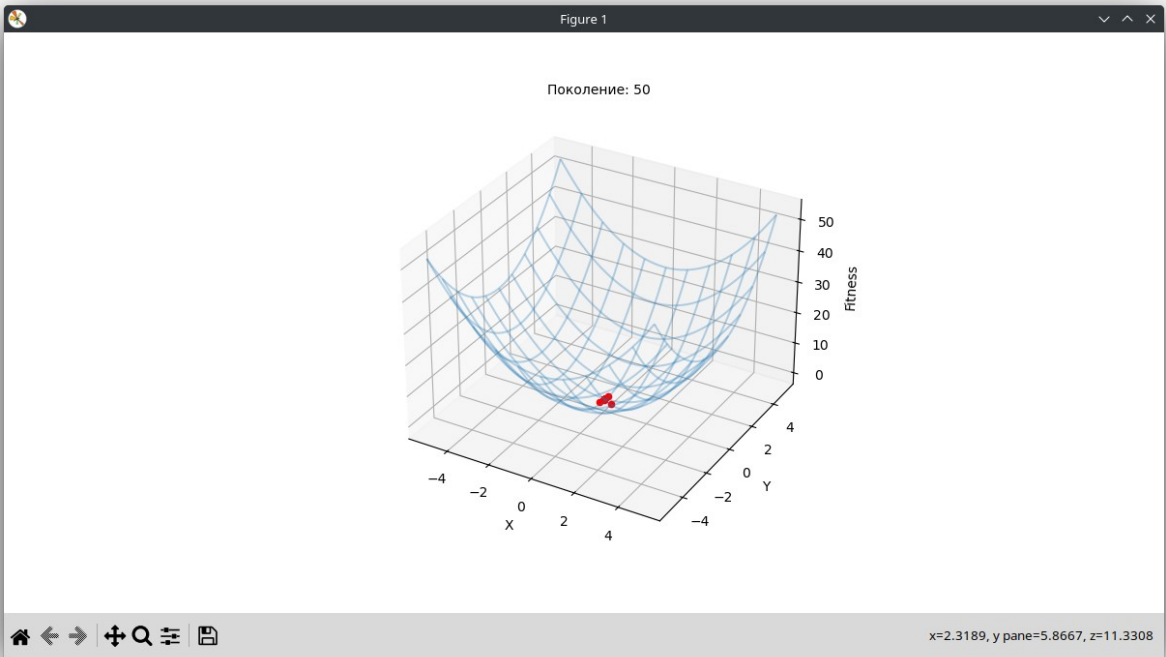
5. Повторить процесс поиска решения для $n=3$, сравнить результаты, скорость работы программы.

Выполнение:

1: Реализация генетического алгоритма для оптимизации функции

1. Представление хромосомы: для начала рассматривается одномерная функция, где каждая хромосома будет представлять значение переменной x из диапазона $[-10, 10]$.
2. Генерация начальной популяции: инициализируется популяция случайных значений x из указанного диапазона. Количество особей в популяции задается параметром N .
3. Фитнес-функция: для каждой хромосомы рассчитывается значение целевой функции $f(x)$, где функция оценивает минимизацию. Чем меньше значение функции, тем выше присваивается “фитнес”.
4. Операторы рекомбинации и мутации: используются классические операторы кроссинговера (двойной точечный кроссинговер) и мутации (случайное изменение значений хромосом).
5. Оператор отбора: применяется метод отбора по турниру, чтобы выбрать лучшие особи для создания следующего поколения.
- 3
6. Критерий остановки: Алгоритм прекращает выполнение, если на протяжении

заданного количества поколений не наблюдается улучшения результата.



- Количество поколений: Умеренная вероятность мутации может способствовать снижению числа поколений, так как она помогает избежать преждевременной сходимости к локальным минимумам, позволяя находить более качественные решения.
- Точность нахождения решения: Оптимальные значения вероятности мутации обеспечивают разнообразие популяции и помогают избежать застоя в поиске, что в свою очередь увеличивает шансы нахождения глобального минимума.

Критерий остановки

Критерий остановки вычислений в данном алгоритме реализован через два параметра:

1. Максимальное количество поколений (например, 100 эпох) — это основной параметр, который ограничивает время выполнения алгоритма и предотвращает бесконечный цикл.
2. Количество поколений без улучшения (стагнация) — алгоритм останавливается, если не происходит улучшения решения за заданное число поколений. Это позволяет избежать ненужных вычислений, если алгоритм не показывает прогресса.

- Количество поколений: Умеренная вероятность мутации может способствовать снижению числа поколений, так как она помогает избежать преждевременной сходимости к локальным минимумам, позволяя находить более качественные решения.
- Точность нахождения решения: Оптимальные значения вероятности мутации обеспечивают разнообразие популяции и помогают избежать застоя в поиске, что в свою очередь увеличивает шансы нахождения глобального минимума.

Критерий остановки

Критерий остановки вычислений в данном алгоритме реализован через два параметра:

1. Максимальное количество поколений (например, 100 эпох) — это основной параметр, который ограничивает время выполнения алгоритма и предотвращает бесконечный цикл.
2. Количество поколений без улучшения (стагнация) — алгоритм останавливается, если не происходит улучшения решения за заданное число поколений. Это позволяет избежать ненужных вычислений, если алгоритм не показывает прогресса.
5. Повторить процесс поиска решения для $n=3$, сравнить результаты, скорость работы программы.

Значения при $n=2$:

Результаты схождения для трех генов

Сходимость по сравнению с 2 генами сильно дольше

Причины различий:

1. Размерность пространства:

- При увеличении размерности (с $n = 2$ до $n = 3$) сложность оптимизации возрастает, поскольку пространство решений становится больше.

2. Условия остановки:

- Остановка встроенного алгоритма может быть связана с его настройками (например, FunctionTolerance). Если изменение значений функции становится меньше заданного порога, это может привести к преждевременной остановке.

3. Генерация решения:

- Различные методы отбора, кроссинговера и мутации могут влиять на скорость сходимости алгоритма.

4. Свойства функции:

- Если функция более “плоская” в одной размерности, это может сделать поиск более трудным. В этом случае более низкие значения могут потребовать больше итераций, что влияет на скорость и точность нахождения оптимума.

Причины различий:

- Остановка встроенного алгоритма может быть связана с его настройками (например, FunctionTolerance). Если изменение значений функции становится меньше заданного порога, это может привести к преждевременной остановке.

3. Генерация решения:

- Различные методы отбора, кроссинговера и мутации могут влиять на скорость сходимости алгоритма.

4. Свойства функции:

- Если функция более “плоская” в одной размерности, это может сделать поиск более трудным. В этом случае более низкие значения могут потребовать больше итераций, что влияет на скорость и точность нахождения оптимума.

Выводы:

В ходе выполнения задания была успешно разработана программа на языке python, использующая генетический алгоритм для нахождения оптимума заданной функции.

Листинг

```
import random
import time
import matplotlib.pyplot as plt
from matplotlib import animation
import numpy as np

# Константы генетического алгоритма
POPULATION_SIZE = 50 # количество индивидуумов в популяции
MAX_GENERATIONS = 50 # максимальное количество поколений
P_CROSSOVER = 0.9 # вероятность скрещивания
P_MUTATION = 0.1 # вероятность мутации
N_VECTOR = 2 # размерность задачи (количество генов в хромосоме)
LIMIT_VALUE_TOP = 5.12 # верхняя граница области поиска
LIMIT_VALUE_DOWN = -5.12 # нижняя граница области поиска
RANDOM_SEED = 42 # для воспроизводимости
random.seed(RANDOM_SEED)

class Individual(list):
    """Класс для представления индивидуумов популяции"""
    def __init__(self, *args):
        super().__init__(*args)
        # print(args)
        self.value = 0

# Функция приспособленности (De Jong's function 1)
def fitness_function(f):
    return sum(x**2 for x in f)

# Создание индивидуумов и популяции
def individual_creator():
    return Individual([random.uniform(LIMIT_VALUE_DOWN, LIMIT_VALUE_TOP)
for _ in range(N_VECTOR)])

def population_creator(n=0):
    return [individual_creator() for _ in range(n)]
```

```

# Начальная популяция
population = population_creator(POPULATION_SIZE)
fitness_values = list(map(fitness_function, population))
for individual, fitness_value in zip(population, fitness_values):
    individual.value = fitness_value

# Для статистики
min_fitness_values = []
mean_fitness_values = []

# Сортировка начальной популяции по значению приспособленности
population.sort(key=lambda ind: ind.value)

def clone(individual):
    """Клонирование индивидуума"""
    clone_ind = Individual(individual[:])
    clone_ind.value = individual.value
    return clone_ind

def selection(population, n=POPULATION_SIZE):
    """Турнирная селекция"""
    offspring = []
    for _ in range(n):
        participants = random.sample(population, 4)
        winner = min(participants, key=lambda ind: ind.value)
        offspring.append(winner)
    return offspring

def crossover(parent1, parent2):
    """Одноточечное скрещивание"""
    point = random.randint(1, len(parent1) - 1)
    parent1[point:], parent2[point:] = parent2[point:], parent1[point:]

def mutate(individual, mutation_prob=1.0 / N_VECTOR):
    """Мутация гена"""
    for i in range(len(individual)):
        if random.random() < mutation_prob:
            individual[i] += random.uniform(-0.5, 0.5)
            individual[i] = max(min(individual[i], LIMIT_VALUE_TOP),
LIMIT_VALUE_DOWN)

# Анимация процесса оптимизации
generation_counter = 0

def animate(frame):
    global generation_counter, population

    ax.clear()
    ax.plot_wireframe(X, Y, Z, rstride=10, cstride=10, alpha=0.3)

```



```

ax.set_title(f"Поколение: {generation_counter}", fontsize=10)
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Fitness")

# Отображение текущих индивидов популяции
for ind in population:
    ax.scatter(ind[0], ind[1], ind.value, color='red', s=20)

# Вывод значений x и y для каждого индивидуума
if generation_counter % 5 == 0: # Выводить каждые 5 поколений
    print(f"Поколение {generation_counter}:")
    for ind in population:
        # print(ind)
        # print(f" Индивидуум: x={ind[0]:.6f}, y={ind[1]:.6f},
Fitness={ind.value:.2f}")
        print(f" Индивидуум: x={ind[0]:.6f}, y={ind[1]:.6f},
z={ind.value:.6f}")

# Селекция, кроссовер и мутация
offspring = selection(population)
offspring = list(map(clone, offspring))

for child1, child2 in zip(offspring[::2], offspring[1::2]):
    if random.random() < P_CROSSOVER:
        crossover(child1, child2)

for mutant in offspring:
    if random.random() < P_MUTATION:
        mutate(mutant)

# Вычисление новой функции приспособленности
fresh_fitness_values = list(map(fitness_function, offspring))
for ind, fitness_value in zip(offspring, fresh_fitness_values):
    ind.value = fitness_value

# Обновление популяции
population[:] = offspring

# Сбор статистики
fitness_values = [ind.value for ind in population]
std_fitness = np.std(fitness_values)
min_fitness = min(fitness_values)
mean_fitness = sum(fitness_values) / len(fitness_values)
min_fitness_values.append(min_fitness)
mean_fitness_values.append(mean_fitness)

if generation_counter % 5 == 0:
    print(f"Минимальная приспособле нность: {min_fitness:.6f}\nСредняя
приспособленность: {mean_fitness:.6f}\nСтандартное отклонение:

```

```
{std_fitness:.6f}"))

    generation_counter += 1

# Создание сетки для визуализации функции
X = np.linspace(LIMIT_VALUE_DOWN, LIMIT_VALUE_TOP, 100)
Y = np.linspace(LIMIT_VALUE_DOWN, LIMIT_VALUE_TOP, 100)
X, Y = np.meshgrid(X, Y)
Z = X**2 + Y**2

# Настройка графиков
fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 1, 1, projection='3d')

ani = animation.FuncAnimation(fig, animate, frames=MAX_GENERATIONS,
interval=50, repeat=False)
plt.show()
```