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 №4.

Генетическое программирование.

по курсу: Эволюционные методы проектирования программно-информационных систем

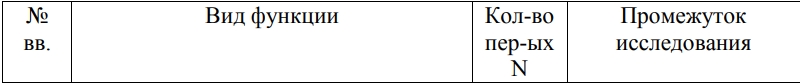
РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. 4136                                                                                Бобрович Н. С.

                                                                         подпись, дата                      инициалы, фамилия

Санкт-Петербург 2024

1. **Задание:**

Вариант 3:  


Скриншот 31-10-2024 124602

1. Разработать эволюционный алгоритм, реализующий ГП для нахождения заданной по варианту функции. 

- Структура для представления программы – древовидное представление.

- Терминальное множество: переменные х1, х2, х3, …, хn, и константы в соответствии с заданием по варианту.

- Функциональное множество: +, -, \*, /, abs(), sin(), cos(), exp(), возведение в степень,

- Фитнесс-функция – мера близости между реальными значениями выхода и требуемыми.

1. Представить графически найденное решение на каждой итерации.

3. Сравнить найденное решение с представленным в условии задачи.

1. **Краткие теоретические сведения:**

В генетическом программировании (ГП) в качестве особи выступает программа, представленная в определенном формате, которая решает некоторую задачу. Часто это выполняется с использованием обучающих данных и индуктивного вывода. ГП очень близко к машинному обучению и поэтому в качестве фитнесс-функции как правило выступают функции ошибки. ГП работает с генетическим материалом переменной длины, что требует нестандартной формы представления генома и соответствующих генетических операторов.

Программы составляются из переменных, констант и функций, которые связаны некоторыми синтаксическими правилами. Поэтому определяется терминальное множество, содержащее константы и переменные, и функциональное множество, которое состоит, прежде всего, из операторов и необходимых элементарных функций (exp(x), sin(x) и т.п.). Следует отметить, что терминалы и функции играют различную роль. Терминалы обеспечивают входные значения в систему (программу), в то время как функции используются при обработке значений внутри системы. Термины «функции» и «терминалы» взяты из древовидного представления, и соответствуют узлам древовидных (или графоподобных) структур.

1. **Программа и результаты выполнения индивидуального задания с комментариями и выводами:**

**Код программы на ЯП Python:**

import random

from math import sin, cos, exp, fabs

import numpy as np

def compare\_float\_and\_complex(value, other\_value):

value\_real = abs(value)

other\_value\_real = abs(other\_value)

if value\_real < other\_value\_real:

return -1

elif value\_real > other\_value\_real:

return 1

elif value\_real == other\_value\_real:

return 0

else:

return 0

# Определение функциональных узлов

def add(a, b):

return a + b

def sub(a, b):

return a - b

def mul(a, b):

return a \* b

def div(a, b):

if b == 0:

return 9999 # большое число вместо деления на ноль

else:

return a / b

def power(a, b):

return a \*\* b

def abs\_val(a):

return np.sqrt(np.abs(a))

def sin\_val(a):

return sin(a)

def cos\_val(a):

return cos(a)

def exp\_val(a):

return exp(a)

func\_set = [add, sub, mul, div, power, abs\_val, sin\_val, cos\_val, exp\_val]

# Терминальные узлы

term\_set = ['x']

class TreeNode:

def \_\_init\_\_(self, data=None, left=None, right=None):

self.data = data

self.left = left

self.right = right

def evaluate(self, x):

if isinstance(self.data, str):

return x[0] # Возвращаем первое значение из списка x

elif callable(self.data):

if self.data in [abs\_val, sin\_val, cos\_val, exp\_val]:

if self.left is not None:

return self.data(self.left.evaluate(x))

else:

raise ValueError("Left child is missing!")

else:

if self.left is not None and self.right is not None:

return self.data(self.left.evaluate(x), self.right.evaluate(x))

else:

raise ValueError("One of the children is missing!")

def \_\_repr\_\_(self):

if isinstance(self.data, str):

return self.data

elif callable(self.data):

name = self.data.\_\_name\_\_

if self.data in [abs\_val, sin\_val, cos\_val, exp\_val]:

return f"{name}({self.left})"

else:

return f"({self.left} {name} {self.right})"

def generate\_random\_tree(max\_depth, func\_set, term\_set, depth=0):

if depth >= max\_depth or random.random() < 0.5:

return TreeNode(random.choice(term\_set))

else:

func = random.choice(func\_set)

if func in [abs\_val, sin\_val, cos\_val, exp\_val]:

return TreeNode(func, generate\_random\_tree(max\_depth, func\_set, term\_set, depth+1))

else:

left = generate\_random\_tree(max\_depth, func\_set, term\_set, depth+1)

right = generate\_random\_tree(max\_depth, func\_set, term\_set, depth+1)

return TreeNode(func, left, right)

def fitness(tree, target\_function, X):

total\_error = 0

for x in X:

try:

predicted\_value = tree.evaluate(x)

actual\_value = target\_function(x)

error = (predicted\_value - actual\_value)\*\*2

total\_error += error

except Exception as e:

total\_error += 1000000 # большая ошибка за исключение

return total\_error

def tournament\_selection(population, k=3):

best = None

for \_ in range(k):

candidate = random.choice(population)

if not best or compare\_float\_and\_complex(fitness(candidate, target\_function, X), fitness(best, target\_function, X)):

best = candidate

return best

def crossover(parent1, parent2):

if random.random() < 0.7:

subtree1 = random.choice([parent1.left, parent1.right])

subtree2 = random.choice([parent2.left, parent2.right])

# Проверяем, что выбранные узлы не являются пустыми

if subtree1 is not None and subtree2 is not None:

new\_parent1 = TreeNode(parent1.data, parent1.left, parent1.right)

new\_parent2 = TreeNode(parent2.data, parent2.left, parent2.right)

if subtree1 == parent1.left:

new\_parent1.left = subtree2

else:

new\_parent1.right = subtree2

if subtree2 == parent2.left:

new\_parent2.left = subtree1

else:

new\_parent2.right = subtree1

return new\_parent1, new\_parent2

return parent1, parent2

def mutate(tree, func\_set, term\_set, max\_depth):

if random.random() < 0.05:

node = tree

while True:

if node.left is None and node.right is None:

break

next\_node = random.choice([node.left, node.right]) if node.right else node.left

node = next\_node

if random.random() < 0.5:

node.data = random.choice(term\_set)

else:

node.data = random.choice(func\_set)

if node.data in [abs\_val, sin\_val, cos\_val, exp\_val]:

node.left = generate\_random\_tree(max\_depth, func\_set, term\_set)

node.right = None

else:

node.left = generate\_random\_tree(max\_depth, func\_set, term\_set)

node.right = generate\_random\_tree(max\_depth, func\_set, term\_set)

return tree

def evolve(population, target\_function, X, generations=50, max\_depth=5):

best\_fitness = []

for gen in range(generations):

new\_population = []

for \_ in range(len(population)):

parent1 = tournament\_selection(population)

parent2 = tournament\_selection(population)

child1, child2 = crossover(parent1, parent2)

mutated\_child1 = mutate(child1, func\_set, term\_set, max\_depth)

mutated\_child2 = mutate(child2, func\_set, term\_set, max\_depth)

new\_population.append(mutated\_child1)

new\_population.append(mutated\_child2)

population = new\_population

best\_individual = min(population, key=lambda ind: fitness(ind, target\_function, X))

best\_fitness.append(fitness(best\_individual, target\_function, X))

print(f"Generation {gen}: Best Fitness = {best\_fitness[-1]} | Program: {best\_individual}")

return best\_individual, best\_fitness

# Функция для тестирования

def target\_function(x):

result = 0

for i in range(1, len(x)+1):

result += sum(val\*\*2 for val in x[:i])

return result

# Генерация начальных значений x

N = 8

X = [[random.uniform(-5.536, 65.536)] for \_ in range(N)]

# Начальная популяция

population\_size = 20

max\_depth = 5

initial\_population = [generate\_random\_tree(max\_depth, func\_set, term\_set) for \_ in range(population\_size)]

# Запуск эволюции

best\_solution, best\_fitness = evolve(initial\_population, target\_function, X, generations=30, max\_depth=max\_depth)

plt.figure(figsize=(10, 6))

plt.plot(range(len(best\_fitness)), best\_fitness, label="Fitness")

plt.xlabel("Generations")

plt.ylabel("Fitness Value")

plt.title("Evolution of Fitness over Generations")

plt.legend()

plt.show()

print("Best Solution:", best\_solution)

print("Fitness History:", best\_fitness)

**Результат выполнения программы:**  
Generation 0: Best Fitness = 44638856.58734117 | Program: x

Generation 1: Best Fitness = 44638856.58734117 | Program: x

Generation 2: Best Fitness = 44638856.58734117 | Program: x

Generation 3: Best Fitness = 8486745.044886313 | Program: ((x mul x) mul sin\_val(exp\_val(x)))

Generation 4: Best Fitness = 8000000 | Program: cos\_val(sin\_val(exp\_val(exp\_val(x))))

Generation 5: Best Fitness = 5320882.805079838 | Program: ((x sub ((cos\_val(x) add exp\_val(exp\_val(abs\_val(x)))) sub x)) div (x sub ((cos\_val(x) add exp\_val(exp\_val(abs\_val(x)))) sub x)))

Generation 6: Best Fitness = 5320882.805079838 | Program: ((x sub ((cos\_val(((x div x) power (x sub exp\_val(x)))) add exp\_val(exp\_val(abs\_val(x)))) sub ((x add exp\_val(x)) div (x power cos\_val(x))))) div (x sub ((cos\_val(((x div x) power (x sub exp\_val(x)))) add exp\_val(exp\_val(abs\_val(x)))) sub ((x add exp\_val(x)) div (x power cos\_val(x))))))

Generation 7: Best Fitness = 7224893.708691113 | Program: (((((sin\_val(exp\_val(cos\_val(x))) mul cos\_val(x)) sub cos\_val((x mul ((x mul x) add x)))) mul (x add ((x add x) power cos\_val(cos\_val(x))))) add exp\_val(exp\_val(cos\_val((cos\_val(((x mul x) add x)) div (x power x)))))) sub (sin\_val((cos\_val(cos\_val(exp\_val((((((((x sub x) add exp\_val(x)) div sin\_val((x div sin\_val(abs\_val(exp\_val((x add x))))))) div (x add (x mul (cos\_val((exp\_val(x) power x)) div exp\_val(sin\_val(x)))))) sub x) sub (x sub x)) div (sin\_val(cos\_val(((x add x) mul (sin\_val(abs\_val(sin\_val(x))) power x)))) power x))))) power abs\_val((x mul (((exp\_val(x) add x) add x) add abs\_val(sin\_val(x))))))) div (abs\_val(sin\_val(sin\_val(sin\_val(sin\_val(cos\_val(x)))))) sub (cos\_val(((x div x) add x)) add cos\_val((x div exp\_val((x mul sin\_val(((x power cos\_val(x)) sub abs\_val(cos\_val(x))))))))))))

Generation 8: Best Fitness = 8000000 | Program: (((abs\_val((sin\_val(((x add (x sub x)) power x)) power exp\_val(((x power ((x add (x add (sin\_val((x power x)) div (sin\_val(x) div x)))) sub sin\_val(x))) sub sin\_val(x))))) add cos\_val(cos\_val(((((x power x) power cos\_val(x)) div (x sub (sin\_val((x sub (x add x))) mul (x sub x)))) sub x)))) mul abs\_val((sin\_val((exp\_val(x) power (((x sub x) div (cos\_val((x power x)) add x)) mul cos\_val((x mul (x add sin\_val(x))))))) sub (abs\_val((exp\_val((cos\_val(((x add exp\_val((x div sin\_val(x)))) mul (sin\_val((x div x)) power (x mul x)))) power (abs\_val(x) mul (x div (x div x))))) power x)) mul exp\_val(cos\_val(exp\_val(((exp\_val(((x power x) mul x)) sub (x sub (x power x))) add exp\_val((x mul x)))))))))) mul (exp\_val(((((x mul (((x sub (x mul cos\_val((x power x)))) add abs\_val(x)) power x)) sub x) div (cos\_val((exp\_val(x) sub exp\_val(cos\_val(abs\_val(abs\_val(cos\_val((sin\_val((abs\_val(x) mul abs\_val(x))) div x)))))))) div sin\_val((((x mul (((x div (x add exp\_val((x mul x)))) sub x) sub sin\_val(sin\_val(exp\_val((cos\_val(x) div (x div x))))))) power x) div ((((x mul x) sub x) div cos\_val((x div sin\_val(abs\_val(x))))) add x))))) power exp\_val((abs\_val((((x power ((x sub (abs\_val(x) div x)) div ((x sub abs\_val(x)) mul x))) add (cos\_val(cos\_val(x)) add ((x power x) sub exp\_val(x)))) power (cos\_val(x) power cos\_val(abs\_val(cos\_val(((sin\_val((x div x)) add ((x sub (x power (x div x))) mul (x mul x))) mul x))))))) power ((((sin\_val(x) sub abs\_val(x)) div x) add exp\_val(abs\_val((x power x)))) power (exp\_val(x) sub sin\_val(((x div x) sub ((x div (x add (x mul exp\_val((abs\_val(x) add x))))) power x))))))))) sub exp\_val(((((((x sub x) add ((x mul (x sub x)) div x)) power sin\_val((((abs\_val(x) power x) sub x) power (((x add x) div (x mul cos\_val(x))) add abs\_val(x))))) div (abs\_val(sin\_val((x power x))) div (abs\_val(x) sub exp\_val(abs\_val(x))))) div cos\_val((((exp\_val(exp\_val(exp\_val((x mul x)))) power (x mul x)) div ((cos\_val(cos\_val(x)) mul x) mul (abs\_val((x mul (x mul x))) add x))) sub abs\_val(abs\_val(cos\_val(exp\_val(exp\_val((sin\_val((x sub x)) div abs\_val(((x add x) add abs\_val(x)))))))))))) add abs\_val((exp\_val(cos\_val(abs\_val(((exp\_val((x mul abs\_val(x))) add ((x div (x sub x)) sub (x div (x mul x)))) mul sin\_val((((x sub x) sub (x sub (sin\_val((x power (((x add x) mul (x div x)) power x))) mul (((x add exp\_val(abs\_val(x))) power ((exp\_val(x) mul (x sub x)) div (sin\_val(x) mul x))) sub (((exp\_val(x) add (x mul x)) div x) div cos\_val((x div x))))))) mul (x div x))))))) add sin\_val(sin\_val(((exp\_val(abs\_val(sin\_val((x div x)))) power exp\_val((sin\_val((x power cos\_val((sin\_val(cos\_val((abs\_val(sin\_val(sin\_val((x sub x)))) mul x))) div x)))) div abs\_val(x)))) div (x add ((sin\_val(x) add cos\_val((x power (x div x)))) sub sin\_val((x sub (abs\_val(cos\_val(x)) sub sin\_val(x)))))))))))))))Generation 9: Best Fitness = 8000000 | Program: ((cos\_val(sin\_val(((exp\_val(exp\_val(cos\_val(sin\_val(cos\_val((x mul (cos\_val(((cos\_val(x) div exp\_val((x add (sin\_val(x) power x)))) power x)) sub x))))))) power exp\_val((cos\_val(((x mul ((x sub x) sub (x power x))) power (x add abs\_val(x)))) power x))) div (cos\_val(sin\_val((cos\_val(abs\_val(x)) power (cos\_val(cos\_val(exp\_val(cos\_val(x)))) power (x power ((x sub cos\_val(cos\_val(x))) add x)))))) sub ((x div (cos\_val(((exp\_val(x) power (cos\_val(x) div (x power x))) add abs\_val(abs\_val(abs\_val(x))))) add x)) power (((((x mul ((cos\_val(x) power x) sub (x sub sin\_val(x)))) add x) div (exp\_val(x) div x)) div ((x sub ((x add x) add exp\_val(x))) power x)) power ((cos\_val(cos\_val((sin\_val(x) sub x))) sub (x power sin\_val(x))) sub x))))))) sub ((cos\_val((((((x div x) div ((x add x) mul x)) mul abs\_val(x)) div abs\_val(((exp\_val(exp\_val(x)) power x) div x))) power (x sub exp\_val(abs\_val(((x power x) div ((exp\_val(sin\_val(x)) mul x) add cos\_val(abs\_val(x))))))))) add exp\_val(exp\_val(abs\_val(((x power (x add cos\_val(abs\_val(sin\_val(x))))) sub ((exp\_val((exp\_val((x add x)) mul x)) div (((x power x) power (sin\_val((x sub x)) add ((x mul x) div (x sub x)))) add x)) mul ((abs\_val(abs\_val(((exp\_val(x) mul (x mul x)) div (exp\_val(x) sub x)))) div cos\_val(x)) add x))))))) sub (((x div sin\_val((exp\_val(x) mul x))) add exp\_val(cos\_val(exp\_val(sin\_val(x))))) div (abs\_val(exp\_val(exp\_val(((x div (x add cos\_val(exp\_val(x)))) sub x)))) power cos\_val((x add ((sin\_val(abs\_val((x power x))) add x) power ((x mul exp\_val(x)) div (exp\_val(x) sub x))))))))) mul sin\_val(exp\_val(exp\_val(sin\_val((abs\_val(cos\_val(((abs\_val(abs\_val((((x sub ((cos\_val((x mul x)) mul (x add x)) sub x)) div x) sub abs\_val(cos\_val(x))))) add cos\_val(exp\_val((((cos\_val(x) add x) div (sin\_val(x) mul (sin\_val((exp\_val(x) sub sin\_val(x))) power (abs\_val((x power x)) add x)))) div (x power sin\_val(sin\_val(x))))))) div exp\_val(cos\_val(cos\_val(cos\_val((((((x mul x) power x) div x) power (exp\_val(cos\_val(cos\_val(x))) sub sin\_val(x))) power exp\_val(exp\_val(((abs\_val(x) div (x sub ((x power (x mul (x add x))) mul x))) add (x power sin\_val((abs\_val(x) add sin\_val(x))))))))))))))) mul sin\_val((exp\_val(sin\_val(((cos\_val(x) sub ((x sub x) power cos\_val((cos\_val(x) add (x mul x))))) sub (sin\_val(x) power (abs\_val((x power exp\_val(x))) mul cos\_val(x)))))) sub cos\_val(cos\_val(cos\_val(exp\_val(sin\_val(cos\_val((sin\_val(exp\_val(cos\_val((abs\_val(x) sub abs\_val(sin\_val(x)))))) div ((abs\_val(x) div x) mul sin\_val((sin\_val((((((x mul x) power x) div (x add (x div x))) mul (x mul x)) mul (x power x))) sub x))))))))))))))))))

1. **Письменный ответ на контрольный вопрос по варианту:**
2. Опишите древовидное представление.

Древовидное представление данных — это способ организации информации, при котором элементы связаны между собой иерархическими отношениями, напоминающими структуру дерева. В такой структуре каждый элемент называется узлом, а связи между узлами называются ребрами или ветвями. Узлы могут иметь дочерние узлы, которые, в свою очередь, также могут иметь свои дочерние элементы, и так далее.

Основные компоненты древовидной структуры:

Корень: Это верхний узел дерева, который не имеет родительского узла. От корня начинается вся структура дерева.

Узел: Каждый элемент дерева, имеющий связь с другими элементами через ребра. Узлы бывают двух типов:

Внутренний узел: Имеет хотя бы одного потомка (дочернего узла).

Листовой узел (или лист): Узел без потомков.

Ребро: Связь между двумя узлами. Ребро всегда соединяет родителя с одним из его детей.

Уровень: Глубина расположения узла относительно корня. Уровень корня обычно считается нулевым уровнем.

Высота дерева: Максимальная длина пути от корня до самого глубокого листа.

Поддерево: Часть дерева, которая сама является деревом, начиная с некоторого узла.

Примеры использования древовидных структур

Файловая система компьютера: Папки и файлы организованы в виде дерева, где каждая папка может содержать другие папки и файлы.

Дерево решений: Используется для моделирования различных вариантов выбора и их последствий. Например, дерево игры в шахматы, где каждый ход игрока — это ветвление.

Бинарные деревья поиска: Специальный вид деревьев, где у каждого узла максимум два ребенка (левый и правый), и данные упорядочены таким образом, чтобы поиск был эффективным.

Преимущества древовидного представления

Эффективность поиска и вставки элементов благодаря строгому порядку узлов.

Простое визуальное представление сложных структур данных.

Возможность эффективного хранения и обработки иерархических данных.

Недостатки

Сложная реализация некоторых операций, таких как удаление узлов или балансировка дерева.

Ограниченная гибкость при необходимости изменять структуру дерева.