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 №1.

Простой генетический алгоритм.

по курсу: Эволюционные методы проектирования программно-информационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. 4136                                                                                П. В. Иноземцева

                                                                         подпись, дата                      инициалы, фамилия

Санкт-Петербург 2024

1. **Задание:**

Вариант Вид функции Промежуток нахождения решения

3 sin(x)/x^2 x e [3.1,20]

1. Разработать простой генетический алгоритм для нахождения оптимума заданной по варианту функции одной переменной (таб. 1.1). Вид экстремума:   
   Максимум
2. Исследовать зависимость времени поиска, числа поколений (генераций), точности нахождения решения от основных параметров генетического алгоритма:

- число особей в популяции

- вероятность кроссинговера, мутации.

1. Вывести на экран график данной функции с указанием найденного экстремума для каждого поколения.

4. Сравнить найденное решение с действительным

1. **Краткие теоретические сведение:**

Простой генетический алгоритм (ПГА) является базовым вариантом генетических алгоритмов и включает в себя следующие основные элементы:

1. Кодирование решения:

- Решение задачи представляется в виде особи, закодированной в виде строки из генов (хромосома).

- Гены могут быть бинарными, целочисленными, действительными или другими типами данных.

2. Формирование начальной популяции:

- Создается начальная популяция особей, состоящая из случайно сгенерированных хромосом.

3. Оценка пригодности:

- Для каждой особи в популяции вычисляется значение функции приспособленности (fitness function), которая определяет качество решения.

4. Отбор родителей:

- Из текущей популяции с помощью оператора отбора (например, рулеточный отбор, турнирный отбор) выбираются особи-родители для скрещивания.

5. Скрещивание (кроссовер):

- Выбранные родители подвергаются кроссоверу, в результате которого создаются новые особи-потомки.

- Наиболее распространенные методы кроссовера - одноточечный, двухточечный, равномерный.

6. Мутация:

- С определенной вероятностью к некоторым потомкам применяется оператор мутации, который вносит случайные изменения в их гены.

- Мутация помогает поддерживать разнообразие генетического материала и избежать преждевременной сходимости.

7. Формирование новой популяции:

- Создаются новые особи путем скрещивания и мутации.

- Размер новой популяции обычно равен размеру предыдущей популяции.

8. Проверка условий завершения:

- Если условие останова (например, достижение заданного числа поколений или удовлетворительного решения) выполнено, то алгоритм завершается.

- Иначе происходит переход к шагу 3 (оценка пригодности) для новой популяции.

Простой генетический алгоритм является основой для более сложных модификаций и расширений генетических алгоритмов, которые могут быть применены для решения разнообразных оптимизационных задач.

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

#include <iostream>

#include <vector>

#include <cmath>

#include <algorithm>

#include <random>

#include <locale>

#include <codecvt>

// Целевая функция

double objective\_function(double x) {

if (x == 0.0) {

return -std::numeric\_limits<double>::infinity();

}

return std::sin(x) / (x \* x);

}

// Турнирный отбор

int tournament\_selection(std::vector<double>& population, std::vector<double>& fitness) {

// Реализация турнирного отбора

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> dis(0, static\_cast<int>(population.size() - 1));

int tournament\_size = 3;

int winner\_index = 0;

double winner\_fitness = -std::numeric\_limits<double>::infinity();

for (int i = 0; i < tournament\_size; i++) {

int participant\_index = dis(gen);

double participant\_fitness = fitness[participant\_index];

if (participant\_fitness > winner\_fitness) {

winner\_index = participant\_index;

winner\_fitness = participant\_fitness;

}

}

return winner\_index;

}

// Одноточечный кроссовер

void one\_point\_crossover(double& child1, double& child2, double parent1, double parent2) {

// Реализация одноточечного кроссовера

double crossover\_point = static\_cast<double>(rand()) / RAND\_MAX;

child1 = crossover\_point \* parent1 + (1 - crossover\_point) \* parent2;

child2 = crossover\_point \* parent2 + (1 - crossover\_point) \* parent1;

}

// Мутация

void mutation(double& individual, double mutation\_rate) {

// Реализация мутации

if (static\_cast<double>(rand()) / RAND\_MAX < mutation\_rate) {

individual += (static\_cast<double>(rand()) / RAND\_MAX - 0.5) \* 2.0;

}

}

int main() {

std::locale::global(std::locale("ru\_RU.UTF-8"));

std::wstring\_convert<std::codecvt\_utf8<wchar\_t>> converter;

// Определение параметров ГА

int population\_size = 100;

double crossover\_rate = 0.8;

double mutation\_rate = 0.1;

int max\_generations = 1000;

// Инициализация популяции

std::vector<double> population(population\_size);

std::vector<double> fitness(population\_size);

// Основной цикл ГА

for (int generation = 0; generation < max\_generations; generation++) {

// Вычисление значений функции приспособленности

for (int i = 0; i < population\_size; i++) {

population[i] = 3.1 + static\_cast<double>(rand()) / RAND\_MAX \* (20.0 - 3.1);

fitness[i] = objective\_function(population[i]);

}

// Отбор родителей

std::vector<double> parents(population\_size);

for (int i = 0; i < population\_size; i++) {

parents[i] = population[tournament\_selection(population, fitness)];

}

// Скрещивание и мутация

for (int i = 0; i < population\_size; i += 2) {

double child1 = parents[i];

double child2 = parents[i + 1];

one\_point\_crossover(child1, child2, parents[i], parents[i + 1]);

mutation(child1, mutation\_rate);

mutation(child2, mutation\_rate);

population[i] = child1;

population[i + 1] = child2;

}

}

// Вывод результата

double best\_x = \*std::max\_element(population.begin(), population.end(), [&](double x1, double x2) {

return objective\_function(x1) < objective\_function(x2);

});

double best\_f = objective\_function(best\_x);

std::wcout << L"Найденный оптимум: x = " << best\_x << L", f(x) = " << best\_f << std::endl;

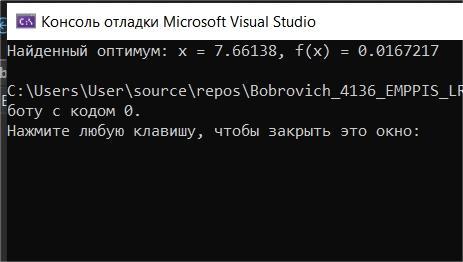
// Построение графика

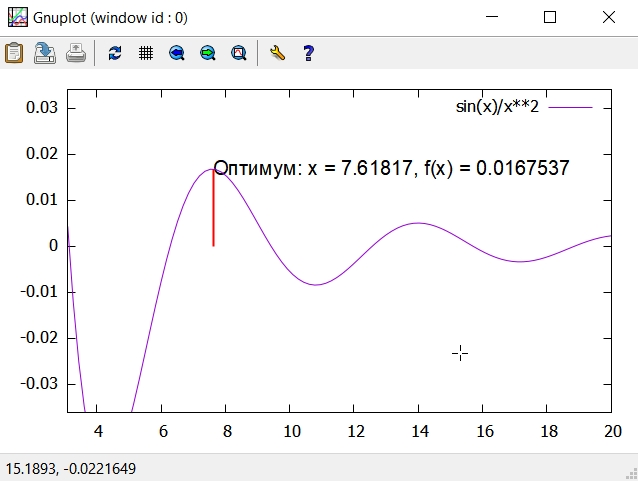
// Реализуйте построение графика функции с отмеченным найденным экстремумом

return 0;

}

Результат выполнения функции:





1. **Письменный ответ на контрольный вопрос по варианту:**
2. Какую роль в ГА играет оператор репродукции (ОР)?

Оператор репродукции (ОР) играет ключевую роль в генетических алгоритмах (ГА). Основными функциями оператора репродукции являются:

1. Отбор родителей:

- ОР отвечает за выбор особей-родителей из текущей популяции для создания новых потомков.

- Методы отбора могут включать рулеточный отбор, турнирный отбор, ранговый отбор и другие.

2. Размножение:

- ОР производит скрещивание выбранных родителей для создания новых особей-потомков.

- Методы скрещивания могут включать одноточечное, двухточечное или равномерное скрещивание.

3. Поддержание разнообразия популяции:

- ОР помогает сохранять разнообразие генетического материала в популяции, предотвращая преждевременную сходимость к локальному оптимуму.

- Это достигается за счет применения операторов, таких как мутация, которые вносят случайные изменения в геном особи.

Таким образом, оператор репродукции играет центральную роль в генетических алгоритмах, обеспечивая механизм для отбора, размножения и поддержания разнообразия в популяции, что в конечном итоге способствует эффективному поиску глобального оптимального решения.

Грамотный выбор и настройка оператора репродукции являются важными факторами, влияющими на производительность и сходимость генетического алгоритма.