Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення
Дисципліна: Алгоритми та структури даних

Лабораторна робота №7

Тема: «ОПТИМІЗАЦІЯ ФУНКЦІЇ ІЗ ЗАСТОСУВАННЯМ ГЕНЕТИЧНИХ АЛГОРИТМІВ»

Виконав: ст. гр. КН-24
Куріщенко П. В.
Перевірив: викладач
Константинова Л. В.

Кропивницький 2026

Мета: Написати програму оптимізації функції з використанням генетичного алгоритму.

Варіант - 14

Завдання: Вибрати функцію й діапазон відповідно до варіанта. Побудувати графік функції. Написати програму знаходження максимуму й мінімуму функції на заданому діапазоні. Проаналізувати отримані результати.

№	Функція
14.	Y(x)=-x^cos(5*x), x=[0...10]

Повний лістинг програми міститься в git репозиторії:
https://github.com/movavok/Algorithms-labs/tree/main/lab_7

Лістинг *geneticalgorithm.h*:

```
#ifndef GENETICALGORITHM_H
#define GENETICALGORITHM_H

#include <cmath>
#include <string>
#include <vector>
#include <cstdlib>
#include <ctime>

using namespace std;

class GeneticAlgorithm
{
public:
    GeneticAlgorithm(double xmin, double xmax,
                     int popSize = 50, int gens = 100,
                     double mutRate = 0.01, int chromLen = 16);

    double getXmin() { return m_Xmin; }
    double getXmax() { return m_Xmax; }

    double function(double x);

    double findMin();
    double findMax();

private:
    double m_Xmin, m_Xmax;
```

```
    int m_populationSize;
    int m_generations;
    double m_mutationRate;
    int m_chromosomeLength;

    double findExtreme(bool isMax);

    double random(double min, double max);
    string encodeChromosome(double value);
    double decodeChromosome(const string& binChromosome);
    pair<string, string> crossoverChromosomes(const string& parent1, const
string& parent2);
    void mutateChromosome(string& chromosome);
};

#endif // GENETICALGORITHM_H
```

Лістинг *geneticalgorithm.cpp*:

```
#include "geneticalgorithm.h"

GeneticAlgorithm::GeneticAlgorithm(double xmin, double xmax, int popSize, int
gens, double mutRate, int chromLen)
    : m_Xmin(xmin), m_Xmax(xmax)
    , m_populationSize(popSize), m_generations(gens)
    , m_mutationRate(mutRate), m_chromosomeLength(chromLen)
{
    srand(static_cast<unsigned>(time(nullptr)));
}

double GeneticAlgorithm::function(double x) { return -pow(x, cos(5 * x)); }

double GeneticAlgorithm::findMin() { return findExtreme(false); }

double GeneticAlgorithm::findMax() { return findExtreme(true); }

double GeneticAlgorithm::findExtreme(bool isMax) {
    vector<string> population;
    for(int count = 0; count < m_populationSize; ++count) {
        double randomValue = random(m_Xmin, m_Xmax);
        population.push_back(encodeChromosome(randomValue));
    }

    string bestChromosome;
    double bestFitness = isMax ? std::numeric_limits<double>::lowest()
                               : std::numeric_limits<double>::max();

    for(int generation = 0; generation < m_generations; ++generation) {
        vector<double> fitnessValues;
        for(string& chromo : population) {
            double value = decodeChromosome(chromo);
            double fitness = function(value);
            fitnessValues.push_back(fitness);

            if((isMax && fitness > bestFitness) || (!isMax && fitness <
bestFitness)) {
                bestFitness = fitness;
                bestChromosome = chromo;
```

```
            }
        }

        vector<string> newPopulation;
        while(newPopulation.size() < population.size()) {
            int parentIdx1 = rand() % population.size();
            int parentIdx2 = rand() % population.size();

            auto [child1, child2] =
crossoverChromosomes(population[parentIdx1], population[parentIdx2]);

            mutateChromosome(child1);
            mutateChromosome(child2);

            newPopulation.push_back(child1);
            if(newPopulation.size() < population.size())
                newPopulation.push_back(child2);
        }

        population = newPopulation;
    }

    return decodeChromosome(bestChromosome);
}

double GeneticAlgorithm::random(double min, double max) {
    return min + (max - min) * (static_cast<double>(rand()) / RAND_MAX);
}

string GeneticAlgorithm::encodeChromosome(double value) {
    int maxBinary = (1 << m_chromosomeLength) - 1;
    int normValue = static_cast<int>((value - m_Xmin) / (m_Xmax - m_Xmin) *
maxBinary);

    string binLine;
    for(int bitIdx = m_chromosomeLength - 1; bitIdx >= 0; --bitIdx)
        binLine += ((normValue >> bitIdx) & 1) ? '1' : '0';

    return binLine;
}

double GeneticAlgorithm::decodeChromosome(const string &binChromosome) {
    int decimalValue = 0;
    int length = binChromosome.size();

    for (int index = 0; index < length; ++index)
        if (binChromosome[index] == '1')
            decimalValue += 1 << (length - 1 - index);

    double maxBinary = (1 << length) - 1;
    double realValue = m_Xmin + (decimalValue / maxBinary) * (m_Xmax -
m_Xmin);
    return realValue;
}

pair<string, string> GeneticAlgorithm::crossoverChromosomes(const string&
parent1, const string& parent2) {
    int length = parent1.size();
    int crossoverPoint = random(1, length - 1);
```

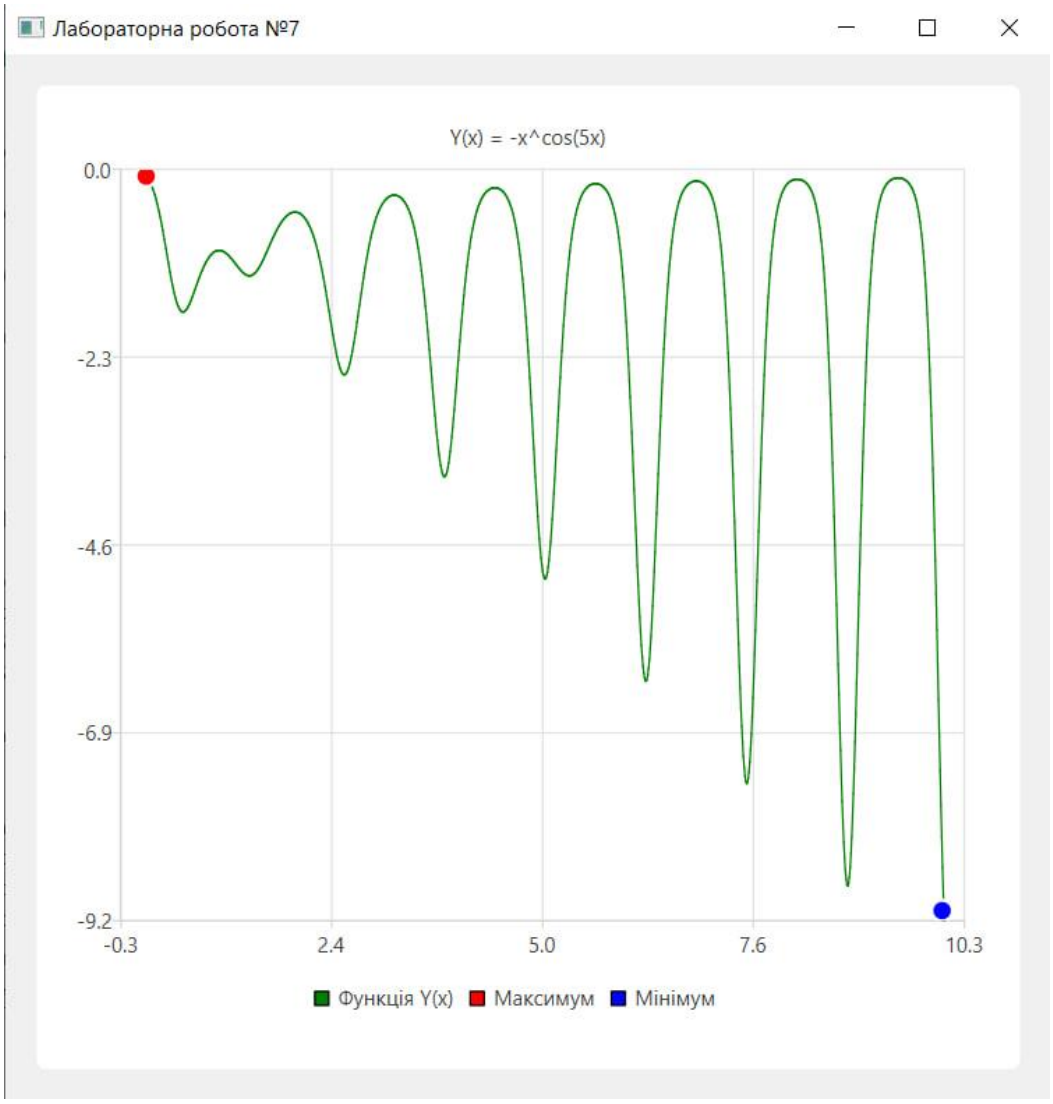
```
    string child1 = parent1.substr(0, crossoverPoint) +
parent2.substr(crossoverPoint);
    string child2 = parent2.substr(0, crossoverPoint) +
parent1.substr(crossoverPoint);

    return {child1, child2};
}

void GeneticAlgorithm::mutateChromosome(string& chromosome) {
    int length = chromosome.size();

    for (int index = 0; index < length; ++index) {
        double randomValue = random(0.0, 1.0);
        if (randomValue < m_mutationRate)
            chromosome[index] = (chromosome[index] == '0') ? '1' : '0';
    }
}
```

Результат виконання:



Висновки і аналіз роботи:

1. Було реалізовано програму для оптимізації функції з використанням генетичного алгоритму.
2. Генетичний алгоритм дозволяє знаходити максимум та мінімум функції, моделюючи процес еволюції популяції.
3. Використання бінарного кодування хромосом, операцій кросовера та мутації забезпечує різноманітність поколінь та пошук оптимальних рішень.
4. Візуалізація графіка функції разом із точками максимального і мінімального значення допомагає наочно оцінити результати оптимізації.
5. Лабораторна робота закріпила знання з оптимізаційних методів і роботи з Qt Charts для графічного відображення даних.