НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ “КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп’ютерних систем

Лабораторна робота №5

з дисципліни “Алгоритми та методи обчислень”

Тема: “ **СЕРЕДНЬОКВАДРАТИЧНЕ НАБЛИЖЕННЯ ФУНКЦІЇ**”

Варіант 4

Виконала: студентка III курсу

КВ-72 Дорош Карина

Перевірив: Зорін Ю. М.

Київ 2019

Завдання для лабораторної роботи

1. Написати програму побудови узагальненого многочлена. Вибір варіанта базису, способу розв'язання нормальної системи та формули інтегрування здійснюється наступним чином. У двійковому поданні 100 номера залікової книжки, взятого за модулем 8,

X = 1 – многочлени Чебишева,

Y = 0 – схема єдиного поділу,

Z = 0 – інтегрування коефіцієнтів нормальної системи за узагальненою формулою Сімпсона

2. За допомогою програми визначити степінь узагальненого многочлена, який забезпечує для функції на проміжку згідно варіанта (табл. 5.1) середньоквадратичне відхилення не гірше ніж $O(10^{-2})$.

3. За допомогою Advanced Grapher побудувати графіки функції, що апроксимується, та узагальненого многочлена степеня визначеного в п.2.

Код програми

Main.cpp

```
#include "header.h"

int main() {

	double** matrix;
	double delta;
	double* matrix_roots;

	ofstream out_file("graph.csv");

	matrix = CreateMatrix(maxn);
	for (int i = 1; i < maxn; i++) {
		matrix_roots = GaussElimination(matrix, i);
		delta = getDelta(i + 1, matrix_roots);
		cout << "Delta = " << delta << " m = " << i + 1  << endl;
		if (delta < eps) {
			double h = (b - a) / 100.0;
			double x = a;
			cout << "\nLeast Squares Deviation = " << delta << " for m = " << i
<< endl;
			for (int j = 0; j <= maxn; j++) {
				x = a + j * h;
```

```
                        cout << "X = "<< setprecision(2) << x << "  Pm  = " <<
setprecision(3) << Pm(x, i + 1, matrix_roots) << endl;
                        out_file << setprecision(3) << x << "," << setprecision(5) <<
Pm(x, i + 1, matrix_roots) << endl;
                    }
                    break;
                }
        }
        system("pause");
        return 0;
}
```

Header.h

```
#pragma once
#include <iostream>
#include <cmath>
#include <iomanip>
#include <fstream>
#define a 1
#define b 10

#define eps 1e-2
#define maxn 100
using namespace std;

double func(double x);

void PrintMatrix(double** matrix, int n);
double Req(double x, int n);

double Pm(double x, int m, double* matrix_roots);

double PowReq(double x, int i, int j);

double SimpsoMethod(double h, int n, int i, int j);

double Integral(int i, int j);

double** CreateMatrix(int n);
double getDelta(int m, double* matrix_roots);
double* GaussElimination(double** initial_matrix, int n);
```

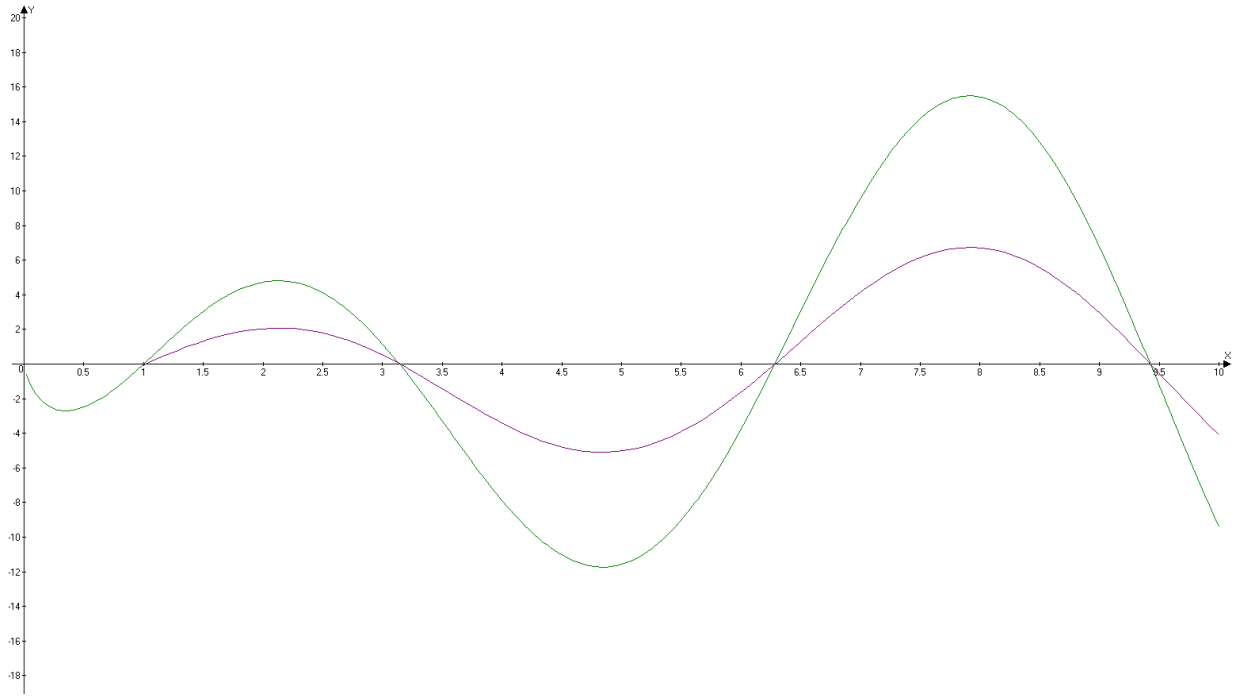
func.h

```
#include "header.h"

double func(double x) {
        return 7.5 * log10(x) * sin(x);
}

void PrintMatrix(double** matrix, int n) {
        for (int i = 0; i < n; i++){
                for (int j = 0; j < n + 1; j++)
                        cout << matrix[i][j] << " ";
                cout << endl;
        }
}
double Req(double x, int n) {
```

```
	if (n == 0) return 1;
	if (n == 1)return x;

	double Tn = 1;
	double Tn_1 = x;
	double buf = 0;
	for (int i = 1; i < n; i++){
		buf = 2.0 * x * Tn_1 - Tn; ;
		Tn = Tn_1;
		Tn_1 = buf;
	}
	return Tn_1;
}

double Pm(double x, int m, double *matrix_roots){
	double p = 0;
	for (int i = 0; i < m; i++)
		p += matrix_roots[i] * Req(x, i);
	return p;
}

double PowReq(double x, int i, int j) {
	if (j == maxn) return func(x) * Req(x, i);
	return Req(x, i) * Req(x, j);
}

double SimpsoMethod(double h, int n, int i, int j) {

	double odd = 0.0;
	double even = 0.0;
	for (int k = 1; k < n; k += 2)
		odd += PowReq(a + k * h, i, j);

	for (int k = 2; k < n - 1; k += 2)
		even += PowReq(a + k * h, i, j);

	return h / 3 * (PowReq(a, i, j) + PowReq(b, i, j) + 4 * odd + 2 * even);
}

double Integral(int i, int j) {
	int n = (b - a) / sqrt(sqrt(eps));
	if (n % 2 == 1) n++;
	double h = (b - a) / (double)n;
	double integral_1n = 0, integral_2n = 0;
	integral_1n = SimpsoMethod(h, n, i, j);
	n *= 2;
	h /= 2;
	integral_2n = SimpsoMethod(h, n, i, j); ;
	while (fabs((integral_1n - integral_2n) / integral_2n) > 15 * eps)
	{
		integral_1n = integral_2n;
		n *= 2;
		h /= 2;
		integral_2n = SimpsoMethod(h, n, i, j);
	}
	return integral_2n;
}
```

```
double** CreateMatrix(int n) {
	double** matrix = new double* [n];
	for (int i = 0; i < n; i++) {
		matrix[i] = new double[n + 1];
		for (int j = 0; j < n + 1; j++)
			matrix[i][j] = Integral(i, j);
	}
	return matrix;
}

double getDelta(int m, double* matrix_roots) {
	double h = (b - a) / 100.0;
	double sum = 0;
	for (double x = a; x <= b; x += h)
		sum += pow((func(x) - Pm(x, m, matrix_roots)), 2);
	return sqrt(sum / 101);
}

double* GaussElimination(double** initial_matrix, int n) {
	double** matrix = new double* [n];
	double * roots = new double[n + 1];

	for (int i = 0; i < n; i++) {
		matrix[i] = new double[n + 1];
		for (int j = 0; j < n; j++)
			matrix[i][j] = initial_matrix[i][j];
		matrix[i][n] = initial_matrix[i][maxn];
	}
	for (int i = 0; i <= n; i++)
		roots[i] = 0;

	for (int i = 0; i < n; i++)
		for (int j = 0; j < n + 1; j++)
			matrix[i][j] = matrix[i][j];
	for (int i = 0; i < n; i++) {
		double div = matrix[i][i];
		if (0 != div) {
			for (int j = 0; j < n + 1; j++) {
				matrix[i][j] /= div;
			}
			for (int z = i + 1; z < n; z++) {
				double mult = matrix[z][i];
				for (int j = 0; j < n + 1; j++) {
					matrix[z][j] -= matrix[i][j] * mult;
				}
			}
		}
	}
	for (int i = n - 1; i >= 0; i--) {
		double tmp = matrix[i][n];
		for (int j = n - 1; j >= i + 1; j--) {
			tmp -= matrix[i][j] * roots[j];
		}
		roots[i] = tmp;
	}
	return roots;
}
```

Графік



Зелений – графік функції, що апроксимується
Фіолетовий – узагальнений многочлен