­­­­­­­­­

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКО”

Факультет прикладної математики

Кафедра системного програмування і спеціальних комп’ютерних систем

**Лабораторна робота №5**

З дисципліни «Алгоритми та методи обчислень»

«СЕРЕДНЬОКВАДРАТИЧНЕ НАБЛИЖЕННЯ ФУНКЦІЙ»

**Виконав:**

**студент III-го курсу**

**групи КВ-41**

**Горпинич-Радуженко Іван**

**Київ 2016**

**Варіант 6:**

**Завдання для лабораторної роботи**

1. 1.Написати програму побудови узагальненого многочлена для заданої функції, яку необхідно апроксимувати (табл. 5.1). Вибір (варіант) базису, способу розв’язання нормальної системи та формули інтегрування здійснюється наступним чином. У двійковому поданні XYZ номера залікової книжки, взятого за модулем 8,   
X = 0 – многочлени Лежандра, X = 1 – многочлени Чебишева,   
Y = 0 – схема єдиного поділу, Y = 1 – схема з вибором головного елемента,   
Z = 0 – інтегрування коефіцієнтів нормальної системи за узагальненою формулою Сімпсона,   
Z = 1 – інтегрування коефіцієнтів нормальної системи за узагальненою формулою трапецій.  
2.За допомогою програми визначити степінь узагальненого многочлена, який забезпечує для заданої функції на заданому проміжку середньоквадратичне відхилення не гірше ніж O(10-2).  
3.За допомогою Advanced Grapher побудувати графік заданої функції та графік узагальненого многочлена степеня, визначеного в п.2.

**Текст програми:**

Integral.h

#pragma once

#include <math.h>

#include <iostream>

#include <fstream>

#include <functional>

using namespace std;

double I(double a, double b, int n, function<double(double)> f);

double integrate(double a, double b, double eps, function<double(double)> f);

integral.cpp

#include "main.h"

double integrate(double a, double b, double eps, function<double(double)> f) {

int i;

double h;

double sig1 = 0;

double sig2 = 0;

int n = (int)(1.0 / sqrt(eps));

double y0 = f(a), yn = f(b);

h = (b - a) / n;

for (i = 1; i < n; i++) {

if (i % 2 == 0)

sig1 += f(a + i\*h);

else

sig2 += f(a + i\*h);

}

double cur\_int, prev\_int = h / 3 \* (4 \* sig2 + 2 \* sig1 + y0 + yn), cur\_even = sig2 + sig1;

int cur\_n = 2 \* n;

do {

h = (b - a) / cur\_n;

double cur\_odd = 0, xi;

for (i = 1, xi = a + h; i < cur\_n; i += 2, xi += 2 \* h) {

cur\_odd += f(xi);

}

cur\_int = h / 3 \* (4 \* cur\_odd + 2 \* cur\_even + y0 + yn);

prev\_int = cur\_int;

cur\_even = cur\_odd + cur\_even;

cur\_n \*= 2;

} while (!(((abs((cur\_int - prev\_int) / cur\_int) / 15) < eps) || ((abs(cur\_int - prev\_int) / 15) < eps)));

return cur\_int;

}

methodMatrix.h

#pragma once

#include <math.h>

#include <iostream>

#include <fstream>

#include <functional>

using namespace std;

double\* gaussian\_elimination(double \*\*AM, int N);

methodMatrix.cpp

#include "main.h"

double\* gaussian\_elimination(double \*\*AM, int N)

{

int i, j, k, l;

double tmp;

double\* res = new double[N];

for (i = 0; i < N; i++)

{

tmp = AM[i][i];

for (j = i; j < N + 1; j++) AM[i][j] /= tmp;

for (k = i + 1; k < N; k++)

{

tmp = AM[k][i];

for (l = i; l < N + 1; l++) AM[k][l] -= AM[i][l] \* tmp;

}

}

for (i = N - 1; i >= 0; i--)

{

tmp = 0;

for (j = N - 1; j > i; j--) tmp += AM[i][j] \* res[j];

res[i] = AM[i][N] - tmp;

}

return res;

}

Main.h

#pragma once

#include <math.h>

#include <iostream>

#include <fstream>

#include <functional>

#include "methodMatrix.h"

#include "Integral.h"

using namespace std;

double lejandr(int n, double x);

double\* constA(double a, double b, double eps, int N, function<double(double)> f);

double value(double\*A, double x, int N);

main.cpp

#include "main.h"

double func(double x)

{

return 1/x - 0.1\*x\*x\*sin(2\*x);

}

double lejandr(int n, double x)

{

double Pn1, Pn = x, Pn\_1 = 1;

if (n == 0) return Pn\_1;

int i = 1;

while (i < n)

{

Pn1 = ((2 \* n + 1) / (n + 1))\*x\*Pn - (n / (n + 1))\*Pn\_1;

Pn\_1 = Pn;

Pn = Pn1;

++i;

}

return Pn;

}

double\* constA(double a, double b, double eps, int N, function<double(double)> f)

{

double \*\* A = new double\*[N + 1];

int i, j;

for (i = 0; i < N + 1; i++)

{

A[i] = new double[N + 2];

}

for (i = 0; i < N + 1; i++)

{

for (j = 0; j < N + 1; j++)

A[i][j] = integrate(a, b, eps, [&](double x) {return lejandr(i, x) \* lejandr(j, x); });

A[i][N + 1] = integrate(a, b, eps, [&](double x) {return lejandr(i, x) \* f(x); });

}

auto res = gaussian\_elimination(A, N + 1);

return res;

}

double value(double\*A, double x, int N)

{

double Pn1, Pn = x, Pn\_1 = 1, sum;

sum = Pn\_1\*A[0];

if (N == 0) return sum;

sum += Pn\*A[1];

int i = 1;

while (i < N + 1)

{

Pn1 = ((2 \* N + 1) / (N + 1))\*x\*Pn - (N / (N + 1))\*Pn\_1;

Pn\_1 = Pn;

Pn = Pn1;

sum += Pn\*A[i + 1];

++i;

}

return sum;

}

int main()

{

ofstream table("table.csv");

for (int n = 10; n < 100; n++)

{

double\* A = constA( 2, 11, 0.00001, n, func);

if (sqrt((integrate(2, 11, 0.00001, [&](double x) {return (func(x) - value(A,x,n)) \* (func(x) - value(A,x,n)); })) / 9) < 0.01)

{

for (double x = 2; x <= 11; x += 0.5)

{

cout << x << "; " << value(A,x,n) << "; "<<endl;

table << x << ";" << value(A,x,n) << ";"<<endl;

}

cout << endl;

cout << "Polin. order=" << n << endl;

break;

}

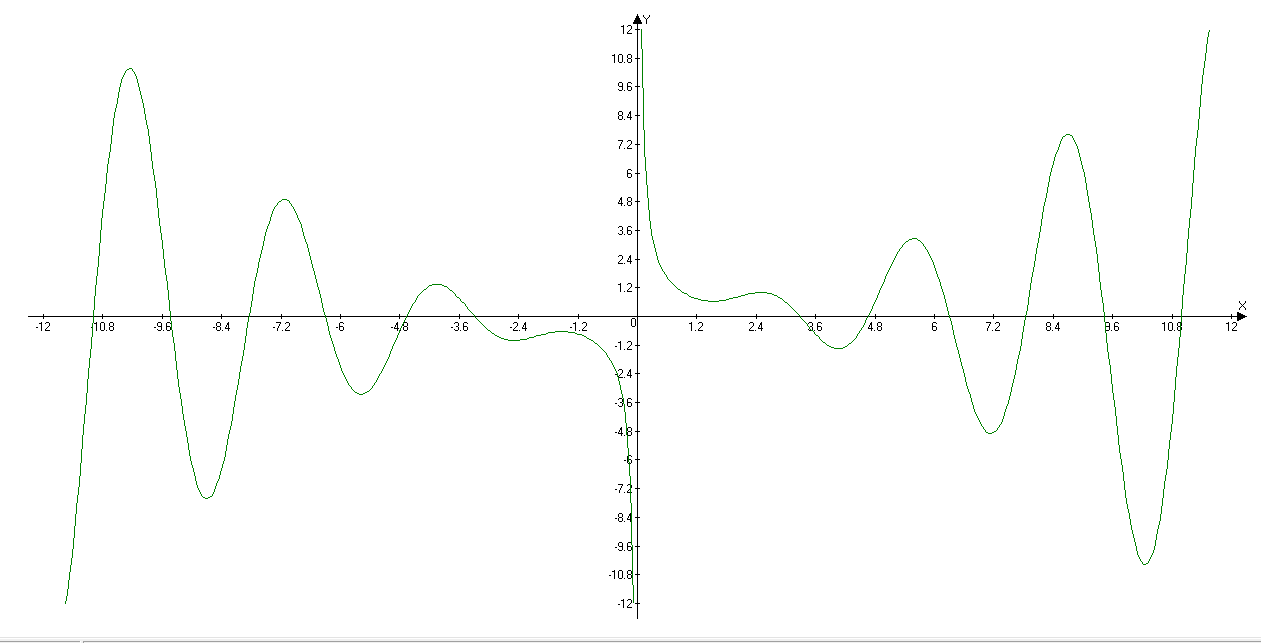
}

return 0;

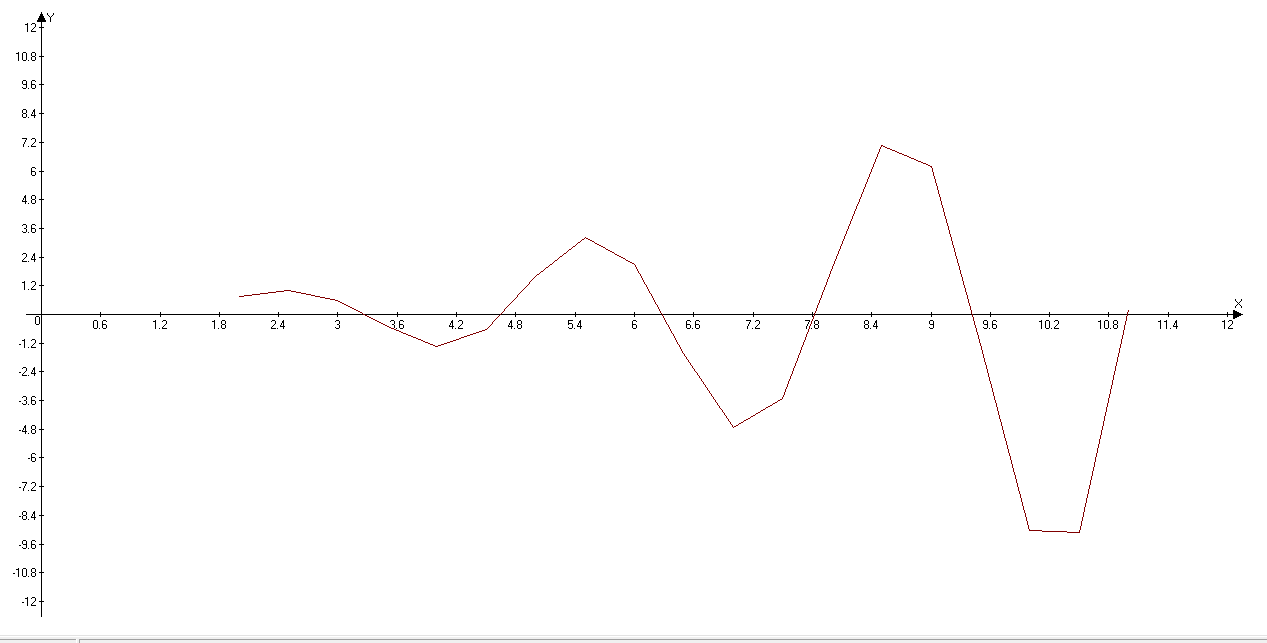
}

**Результати:**

Графік функції:



Графік узагальненого многочлена:



**Висновки:**

В ході виконання лабораторної роботи ми досліджували метод наближення функцій за допомогою методу найменших квадратів. Інтерполяція функції полягає в заміні заданої функції *f(x)* іншою функцією *Ln(x)* за умови, що функції *f(x) і* *Ln(x)* тотожні на заданій послідовності точок. Однак, подання функції за допомогою інтерполяційного многочлена не завжди є зручним: зі збільшенням кількості вузлів зростає його степінь, що не завжди приводить до поліпшення наближеного подання функції на заданому відрізку, або, скажімо, близькість ординат кривих *f(x)* і *Ln(x)* на заданому відрізку ще не гарантує близькості на ньому похідних *(x)* і  *(x)*, тобто малої розбіжності кутових коефіцієнтів дотичних до цих кривих. Такі міркування приводять до доцільності середньоквадратичного наближення функції. Для функції *f(x)*, заданої на відрізку [*a,b*], потрібно підібрати апроксимуючу функцію  таку, щоб значення інтеграла



було якнайменшим.

Метод показав гарні результати, даючи досить точне наближення до функції, що видно з графіків. Ця точність сильно залежить від степені узагальненого многочлена і, зачасту, чим більшу ми вимагаємо точність, тим більшою є степінь. Оскільки метод вимагає багато обчислень інтегралів, систем рівнянь, то його швидкодія не є великою, коли вимагається висока точність.