НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ ТА СПЕЦІАЛІЗОВАНИХ КОМП’ЮТЕРНИХ СИСТЕМ

Лабораторна робота №5  
з дисципліни «Алгоритми та методи обчислення» на тему: «СЕРЕДНЬОКВАДРАТИЧНЕ НАБЛИЖЕННЯ ФУНКЦІЙ»

**Варіант 21**

Виконав:  
студент 3-го курсу,  
гр. КВ-41,  
Яковенко Максим

Київ – 2016

**Завдання для лабораторної роботи**

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.

**Варіант 121:**

121 mod 8=001; X=0, Y=0, Z=1.



**Текст програми**:

**Mainfunc.h**

#pragma once

#include <math.h>

#include <iostream>

#include <fstream>

#include <functional>

#include "Matrixsolv.h"

#include "Integral.h"

using namespace std;

double lejandr(int n, double x);

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

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

**Matrixsolv.h**

#pragma once

#include <math.h>

#include <iostream>

#include <fstream>

#include <functional>

using namespace std;

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

**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);

**Matrixsolv.cpp**

#include "mainfunc.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;

}

**Integral.cpp**

#include "mainfunc.h"

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

{

double h;

double sum = 0.0;

int i;

h = (b - a) / n;

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

{

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

}

return h\*(f(a) + f(b) + sum);

}

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

{

int n = (int)(1 / sqrt(eps));

double In = I(a, b, n, f);

double I2n = I(a, b, 2 \* n, f);

double Rn = fabs(In - I2n) / 3;

while (Rn > eps)

{

n \*= 2;

In = I(a, b, n, f);

I2n = I(a, b, 2 \* n, f);

Rn = fabs(In - I2n) / 3;

}

return I2n;

}

**main.cpp**

#include "mainfunc.h"

double func(double x)

{ return 10 \* sin(x)\*cos(x)\*cos(11 \* 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\* makeA(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 val(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 tbl("table.csv");

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

{

double\* A = makeA( -1, 1, 0.0001, n, func);

if (sqrt((integrate(-1, 1, 0.00001, [&](double x) {return (func(x) - val(A,x,n)) \* (func(x) - val(A,x,n)); })) / 2) < 0.01)

{

for (double x = -1; x <= 1; x += 0.02)

{

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

tbl << x << ";" << val(A,x,n) << ";"<<endl;

}

break;

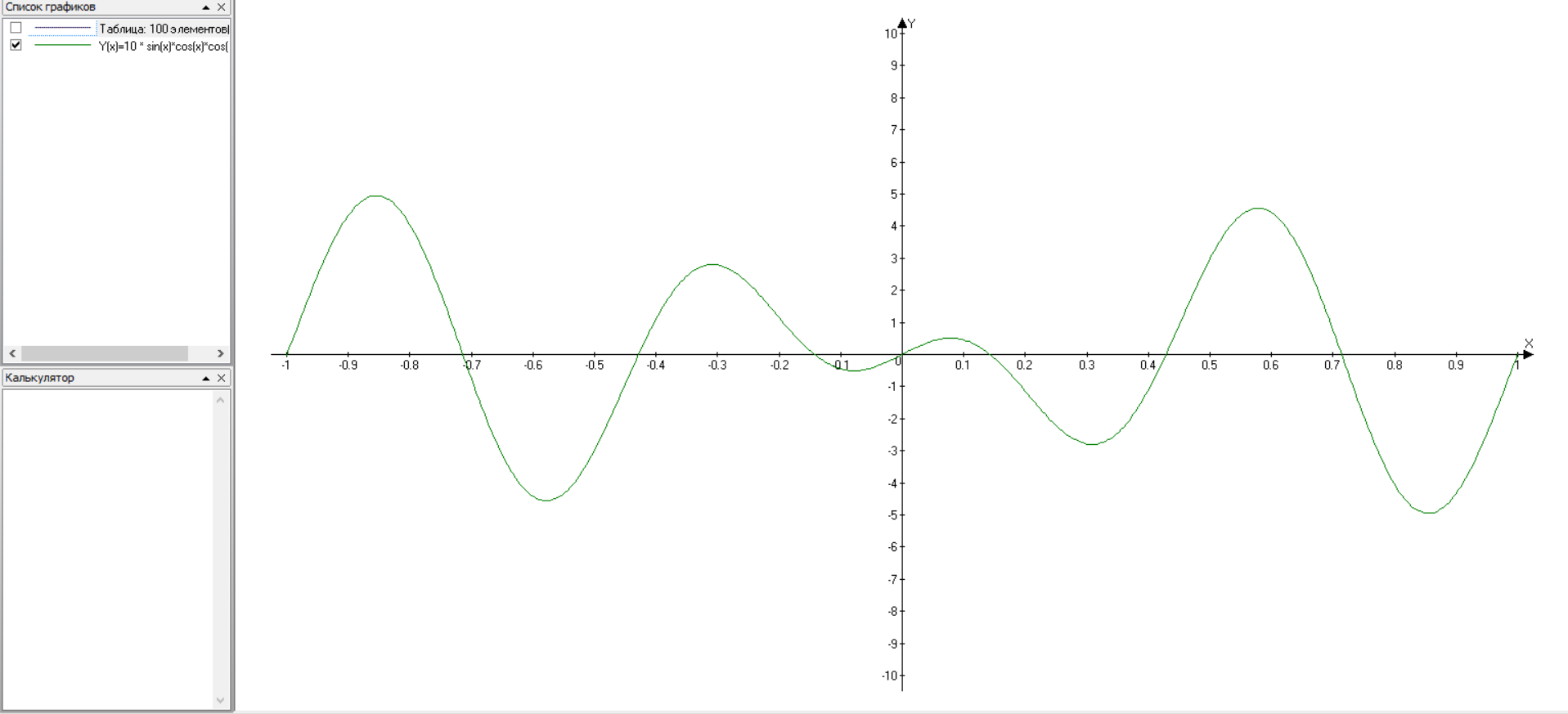
}

}  
 return 0;

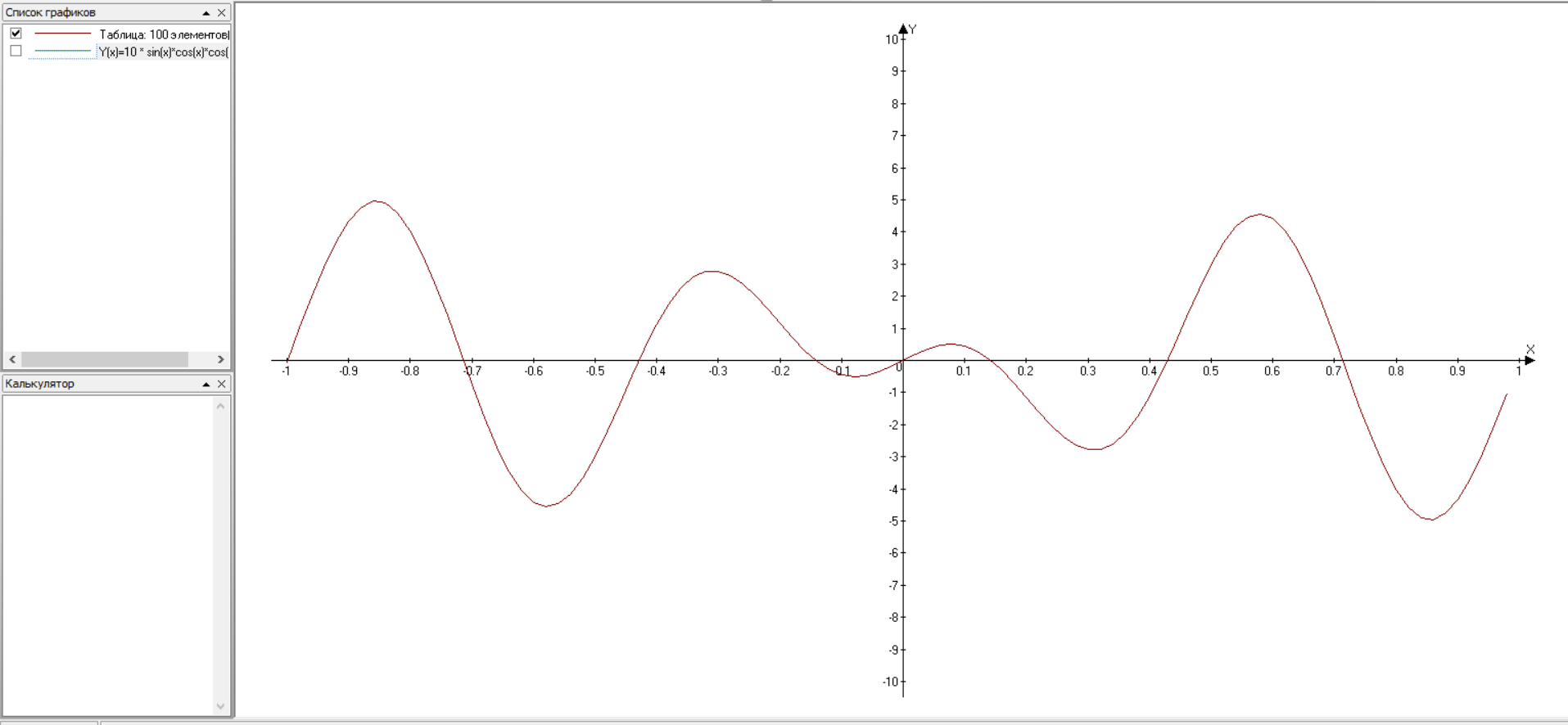
}

**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.