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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ ТА СПЕЦІАЛІЗОВАНИХ КОМП’ЮТЕРНИХ СИСТЕМ

**Лабораторна робота №6  
з дисципліни «Алгоритми та методи обчислення» на тему: «НАБЛИЖЕННЯ ФУНКЦІЙ ЗА ДОПОМОГОЮ**

**ІНТЕРПОЛЯЦІЙНИХ СПЛАЙНІВ»**

**Варіант 21**

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

Київ – 2016

**Завдання для лабораторної роботи**

1. Написати програму побудови натурального кубічного сплайна для заданої функції на заданому проміжку (варіанти завдань наведені в Лабораторній роботі 5).

Вибір способу розв’язання системи (6.7) здійснюється наступним чином. У двійковому поданні XYZ номера залікової книжки, взятої за модулем 6,

000 – проста ітерація, 001 – схема єдиного поділу, 010 – виключення Гауса-Жордана, 011 – схема з вибором головного елемента, 100 – метод прогону, 101 – ітерація Зейделя.

1. За допомогою програми виконати інтерполяцію функції на проміжку відповідно до варіанту. Кількість часткових проміжків *n* має бути в межах 50 – 100.

3. За допомогою Advanced Grapher побудувати графік функції, що апроксимується, та графік інтерполяційного сплайна.

**Варіант 121:**

121 mod 6=001; X=0, Y=0, Z=1.



**Текст програми**:

**Methods.h**

#pragma once

#include <iostream>

#include <cmath>

#include <fstream>

double func(double x);

double\* defineA(double a, double b, int n);

double\* defineC(double\* A, double a, double b, int n);

double\* defineD(double\* C, double a, double b, int n);

double\* defineB(double\* A, double\* C, double\* D, double a, double b, int n);

double\*\* defineSMatrix(double\* A, double a, double b, int n);

double\* gauss\_elemination(double\*\* A, int n);

double getInt(double x, double a, double b, int n, double\*A, double\*B, double\*C, double\*D);

**methods.cpp**

#include "methods.h"

double\* defineA(double a, double b, int n)

{

double\* A = new double[n];

double h = (b - a) / n;

for (int i = 0; i < n; ++i)

{

A[i] = func(a + (i + 1)\*h);

}

return A;

}

double\* defineC(double\* A, double a, double b, int n)

{

double\* C = new double[n];

double\* res = gauss\_elemination(defineSMatrix(A, a, b, n), n);

for (int i = 0; i < n - 1; ++i)

C[i] = res[i];

C[n - 1] = 0;

return C;

}

double\* defineD(double\* C, double a, double b, int n)

{

double h = (b - a) / n;

double\* D = new double[n];

for (int i = n - 1; i > 0; --i)

{

D[i] = (C[i] - C[i - 1]) / h;

}

D[0] = C[0];

return D;

}

double\* defineB(double\* A, double\* C, double\* D, double a, double b, int n)

{

double\* B = new double[n];

double h = (b - a) / n;

B[0] = h\*C[0] / 2 - h\*h\*D[0] / 6 + (A[0] - func(a)) / h;

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

B[i] = h\*C[i] / 2 - h\*h\*D[i] / 6 + (A[i] - A[i - 1]) / h;

return B;

}

double\*\* defineSMatrix(double\* A, double a, double b, int n)

{

double\*\* M = new double\*[n - 1];

double h = (b - a) / n;

for (int i = 0; i < n - 1; i++)

{

M[i] = new double[n];

for (int j = 0; j < n; j++)

{

if (i == j) M[i][j] = 4 \* h;

else if ((j == i - 1) || (j == i + 1)) M[i][j] = h;

else M[i][j] = 0;

}

double k = A[i + 1];

if (i == 0)

M[i][n - 1] = (6 / n)\*(A[i + 1] - 2 \* (A[i] - func(a)));

else

M[i][n - 1] = (6 / n)\*(A[i + 1] - 2 \* A[i] - A[i - 1]);

}

return M;

}

double\* gauss\_elemination(double\*\* AM, int n)

{

int i, j, k;

double p, s;

double\* x = new double[n - 1];

for (k = 0; k<n - 2; k++)

{

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

{

p = AM[i][k] / AM[k][k];

for (j = k; j<n; j++)

AM[i][j] = AM[i][j] - p\*AM[k][j];

}

}

x[n - 2] = AM[n - 2][n - 1] / AM[n - 2][n - 2];

for (i = n - 3; i >= 0; i--)

{

s = 0;

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

{

s += (AM[i][j] \* x[j]);

x[i] = (AM[i][n - 1] - s) / AM[i][i];

}

}

return x;

}

double getInt(double x, double a, double b, int n, double\*A, double\*B, double\*C, double\*D)

{

double xi = a, h = (b - a) / n;

int i;

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

{

xi += h;

if (x < xi) break;

}

double dx = (x - xi);

return A[i] + B[i] \* dx + C[i] \* dx\*dx / 2 + D[i] \* dx\*dx\*dx / 6;

}

**main.cpp**

#include "methods.h"

using namespace std;

double func(double x)

{

return 10 \* sin(x)\*cos(x)\*cos(11 \* x);

}

void main()

{

ofstream tbl("tbl.csv");

double a = -1.0, b = 1.0, x;

int n = 100;

double\* A = new double[n];

double\* B = new double[n];

double\* C = new double[n];

double\* D = new double[n];

double h = (b - a) / n;

A = defineA(a, b, n);

C = defineC(A, a, b, n);

D = defineD(C, a, b, n);

B = defineB(A, C, D, a, b, n);

for (int i = 0; i < n; ++i)

{

x = a + i\*h;

cout << "x= " << x << "; " << "f(x)= " << func(x) << "; " << "Evalsp= " << getInt(x, a, b, n, A, B, C, D) << ";" << endl;

tbl << x << ";" << getInt(x, a, b, n, A, B, C, D) << ";" << endl;

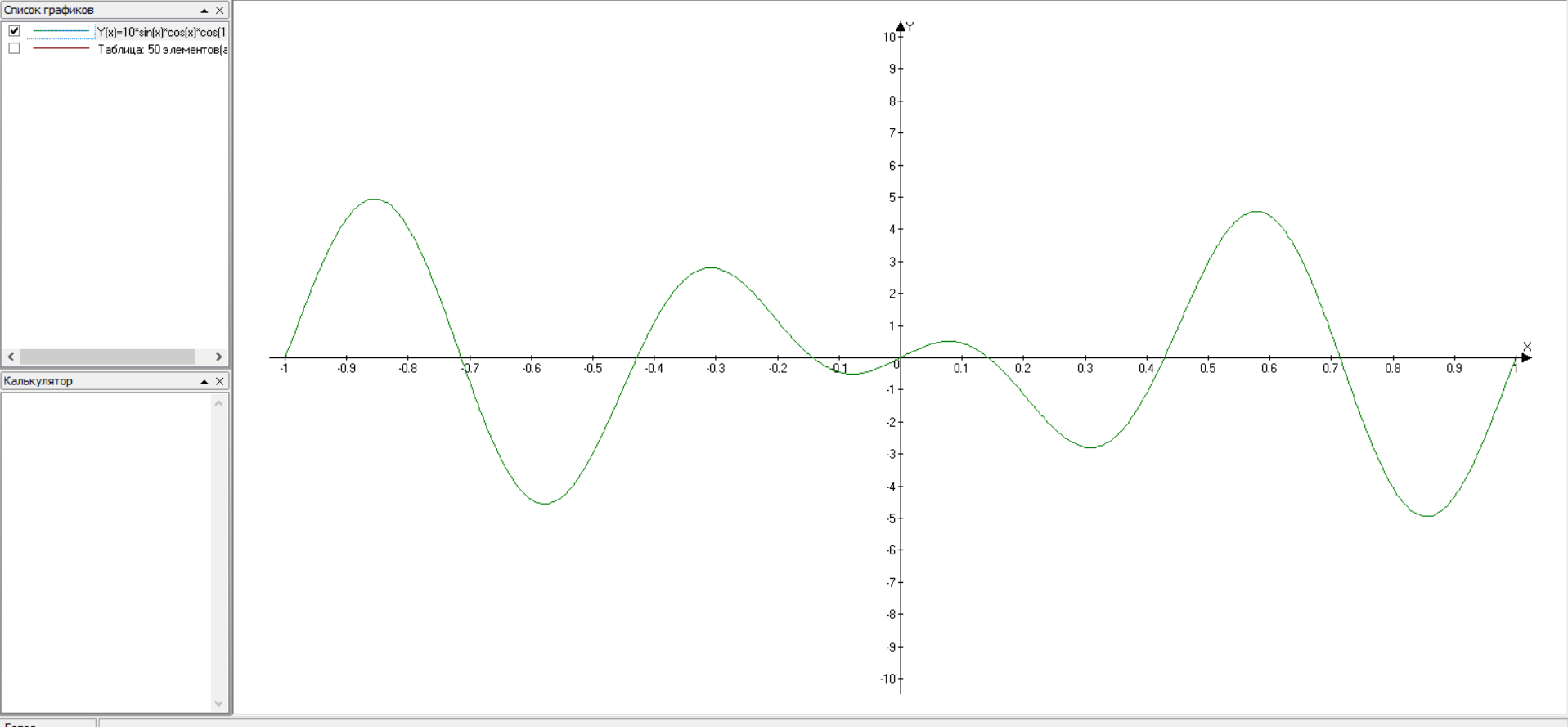
}

return;

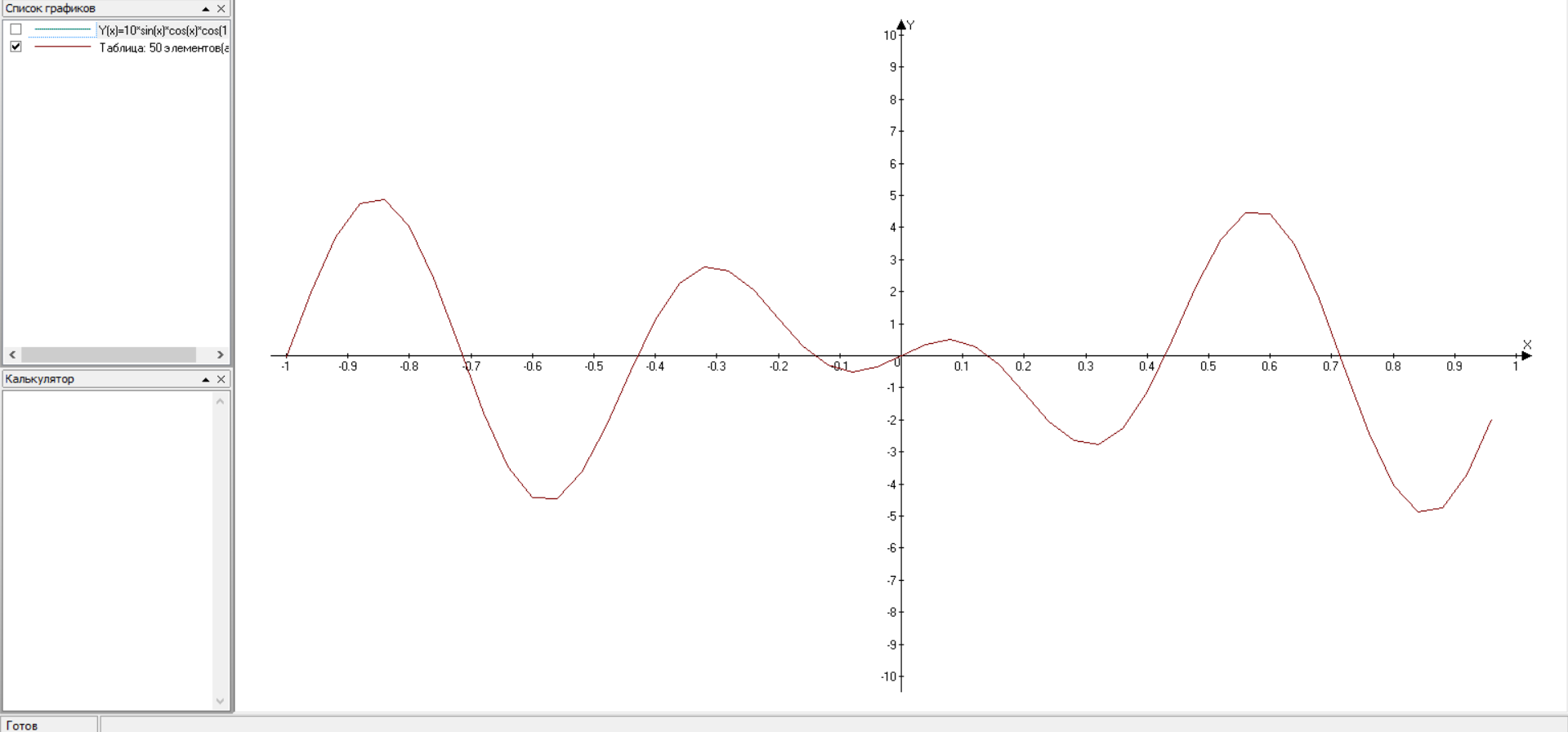
}

**Результати:**

Графік функції:

****

Графік узагальненого многочлена:

****

**Висновки:**

В даній лабораторній роботі ми ознайомилися з ще одним методом інтерполяції функцій – наближення функцій за допомогою інтерполяційних сплайнів. Сплайн-функцією або сплайном називають кусково-поліноміальну функцію, що визначена на відрізку [*a, b*] й має на цьому відрізку декілька неперервних похідних.

Даний метод є більш точнішим від методу середньоквадратичного наближення, оскільки позбавлений недоліку феномену Рунге. За рахунок того, що при обчисленні не потрібно вираховувати інтеграли, даний метод працює значно швидше від середньоквадратичного наближення.