Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. І. Сікорського»

Кафедра автоматизації проектування енергетичних процесів і систем

Лабораторна робота №4
з дисципліни «Чисельні методи в моделюванні енергетичних процесів»
«Інтерполяційні поліноми»
Варіант №1

Виконав:
студент 2-го курсу, ТЕФ
групи ТР-15
Руденко В.І.

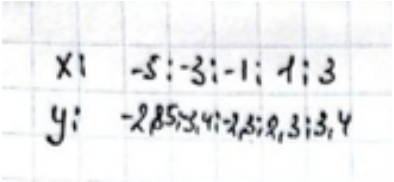
КИЇВ-2022

Мета роботи

Створити програму, яка для заданої функції по заданим точкам будує інтерполяційний поліном Pn(x) у формі Лагранжа або Ньютона, а також здійснює інтерполяцію кубічними сплайнами. Програма має розрахувавати значення похибки ε =| Pn (x) − y(x) | , для чого потрібно вивести на графік із кроком (графік можна будувати допоміжними засобами, наприклад, у Mathcad), меншим у 5-6 разів, ніж крок інтерполяції, відповідні значення поліному та точної функції. Якщо похибка дуже мала, застосувати масштабування. Знайти кубічний інтерполяційний сплайн для заданої функції у Mathcad. Вивести графік результатів.

Варіант завдання (Вихідна система рівнянь).

№	Функція $y(x)$	Вузли інтерполяції x_i
1-10	$\sin(\frac{\alpha}{2}\cdot x)+\sqrt[3]{x\cdot\alpha}$	-5+k, -3+k, -1+k, 1+k, 3+k; k = №*вар* – 1



Теоретична частина

Інтерполяція в обчислювальній математиці - спосіб знаходження проміжних значень величини по наявному дискретному наборі відомих значень.

Нехай маємо n значень x_i, кожному з яких відповідає своє значення y_i. Потрібно знайти таку функцію F, що

$$F(x_i) = y_i,\ i = 0,\dots,n.$$

При цьому x_i називаються вузлами інтерполяції; пари (x_i, y_i) - точками даних; функцію $F(x)$ - інтерполянтом.

Інтерполянти, як правило, будуються у вигляді лінійних комбінацій деяких елементарних функцій:

$$y = \sum_{k=0} c_k\,\Phi_k(\mathrm{x}),$$

де $\Phi_k(\mathrm{x})$ - фіксовані лінійно незалежні функції; $c_0, \dots, c_n$ - не визначені поки що коефіцієнти. З умови *(1)* отримуємо систему $n+1$ рівнянь відносно коефіцієнтів c_k:

$$\sum_{k=0} c_k\Phi_k(x_i) = y_i,\quad \boldsymbol{i = 0,\dots,n.}$$

Звідси отримуємо систему рівнянь:

У якості системи лінійно незалежних функцій $\Phi_k(\mathrm{x})$ частіше за все обирають: степеневі функції $\Phi_k(\mathrm{x}) = x^k$ (в цьому випадку $F = P_n(\mathrm{x})$ - поліном ступеня n); тригонометричні функції.

Поліном Лагранжа.

Будемо шукати інтерполяційний поліном у вигляді

$$P_n(\mathrm{x}) = \sum_{k=0}^{n} c_k\,x^k \quad . \qquad (2)$$

$$c_0 + c_1 x_0 + \ldots + c_n x_0^n = y_0$$
$$\ldots$$
$$c_0 + c_1 x_n + \ldots + c_n x_n^n = y_n$$

Ця система має єдиний розв'язок, а отже і інтерполяційний поліном вигляду *(2)* також єдиний. Форм запису його існує багато.
Лагранж запропонував наступну форму поліному, в основі якої лежить базис поліномів Лагранжа $l_k(x)$ ступеня n:

$$l(x) = 1, \textit{ якщо } i=k$$

Метод прогону

Більшість технічних задач зводиться до розв'язування СЛАР, у яких матриці містять багато нульових елементів, а ненульові елементи розміщені за спеціальною структурою (стрічкові квазітрикутні матриці).

Задачі побудови інтерполяційних сплайнів, різницевих методів розв'язування крайових задач для диференціальних рівнянь зводяться до розв'язування СЛАР з трьохдіагональною матрицею A. У матриці A всі елементи, що не лежать на головній діагоналі і двох сусідніх паралельних діагоналях, дорівнюють нулю. У загальному вигляді такі системи записують так:

Вибір найбільшого елемента при виключенні невідомих за методом Гауса в таких системах робити не можна, оскільки перестановка рядків руйнує структуру матриці. Найчастіше для розв'язку системи з трьохдіагональною матрицею використовують метод прогону, який є частковим випадком методу Гауса.

Прямий хід прогону (алгоритм прямого ходу методу Гауса).
Кожне невідоме x_i виражається через x_{i+1} з допомогою прогоночних коефіцієнтів A_i та B_i

Обернений хід прогонки (аналог оберненого ходу методу Гауса).
Він полягає в послідовному обчисленні невідомих x_i. Спочатку знаходять x_n.

Далі використовуючи формулу *(3)* знаходимо послідовно всі невідомі

Майже у всіх задачах, що приводять до розв'язку системи *(2)* з трьохдіагональною матрицею, забезпечується умова переважання діагональних коефіцієнтів

$$|b_i| \geq |a_i| + |c_i|$$

Це забезпечує існування єдиного розв’язку та достатню стійкістьметоду прогону відносно похибок заокруглення.

Для запису коефіцієнтів a_i, b_i, та прогоночних коефіцієнтів A_{i-1}, B_{i-1} використати один і той же масив.

Графіки:

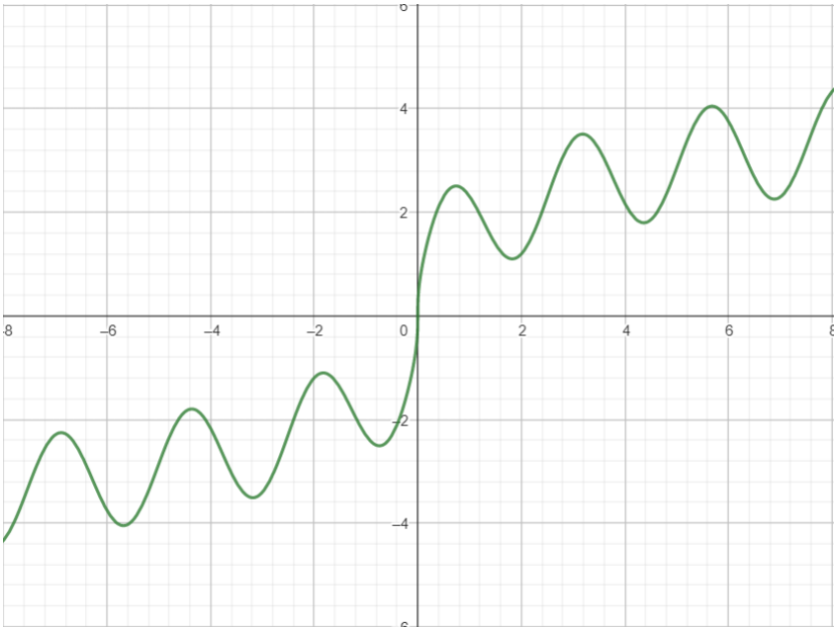


Рисунок 1 Графік За Варіантом

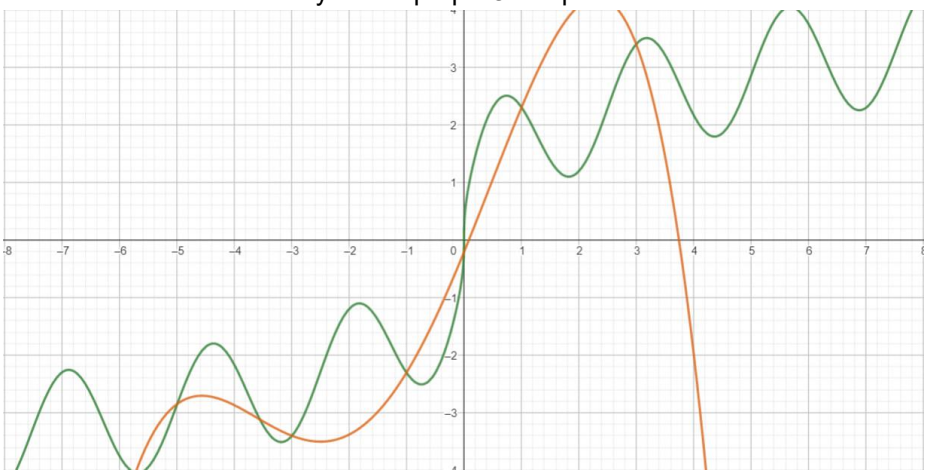
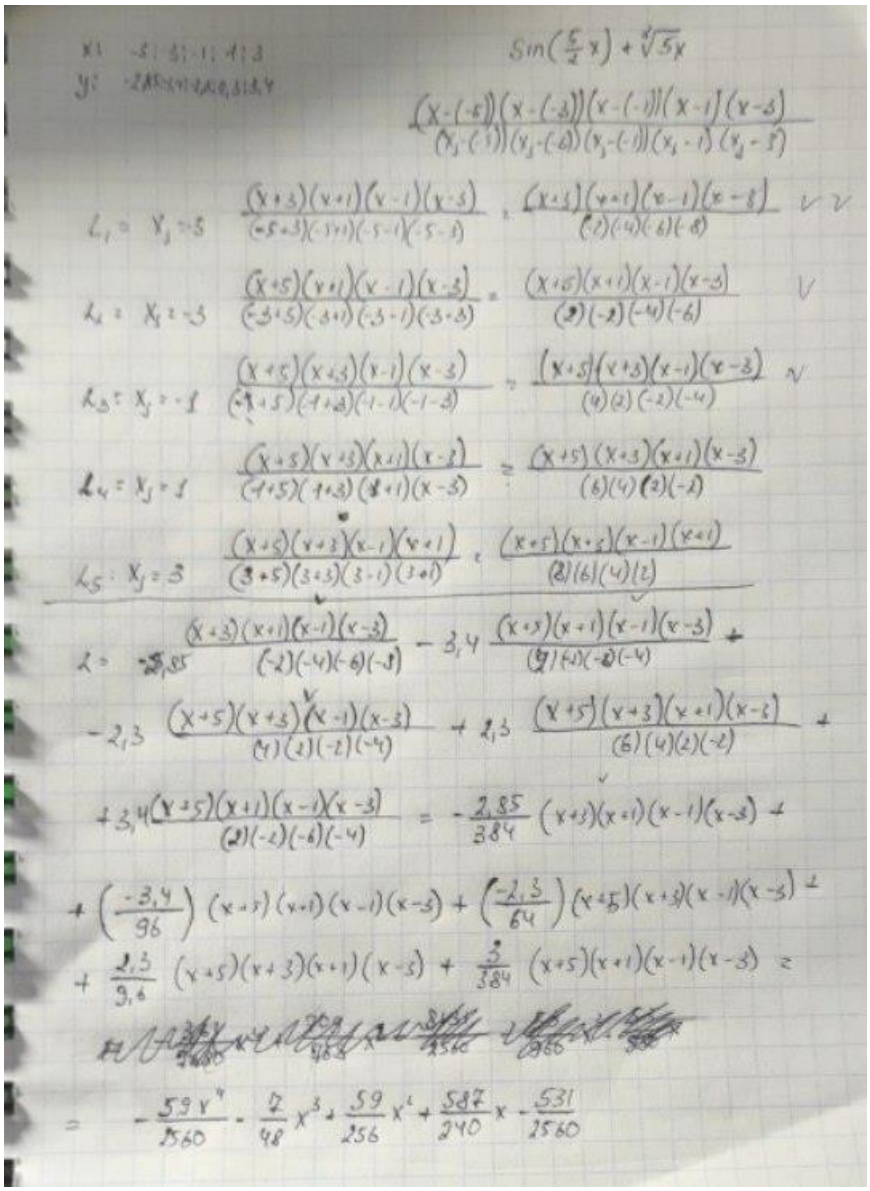


Рисунок 2 Графік функції та інтерполяційного поліному

Рисунок 3 Графік функції та сплайн інтерполяції в MathCad

Вигляд поліному Лагранжа:



Сплайни:

```
---------------|Splines|---------------
----------------------------------------
   Step: 1
X : -3.000000
        a : -2.857696    b :-0.193800     c :0.000000     d :-0.019864
Pn(-3.0) = -2.8577 + -0.1938 * (X - -5.0000) + 0.0000 * (X - -5.0000)^2 + -0.019
9 * (X - -5.0000)^3
----------------------------------------
----------------------------------------
   Step: 2
X : -1.000000
        a : -3.404212    b :0.114343     c :-0.119187    d :0.167978
Pn(-1.0) = -3.4042 + 0.1143 * (X - -3.0000) + -0.1192 * (X - -3.0000)^2 + 0.1680
 * (X - -3.0000)^3
----------------------------------------
----------------------------------------
   Step: 3
X : 1.000000
        a : -2.308448    b :0.557570     c :0.888682     d :-0.006621
Pn(1.0) = -2.3084 + 0.5576 * (X - -1.0000) + 0.8887 * (X - -1.0000)^2 + -0.0066
* (X - -1.0000)^3
----------------------------------------
----------------------------------------
   Step: 4
X : 3.000000
        a : 2.308448     b :1.176510     c :0.848953     d :-0.141492
Pn(3.0) = 2.3084 + 1.1765 * (X - 1.0000) + 0.8490 * (X - 1.0000)^2 + -0.1415 * (
X - 1.0000)^3
----------------------------------------
        Data:
```

Скріншоти роботи програми:

```
----------|Lagranj Method|----------
----------------------------------------
   Step: 1
F(-5.000000):-2.857696
LF(-5.000000) : -2.857696
Error: 0.000000
----------------------------------------
----------------------------------------
   Step: 2
F(-4.500000):-1.855300
LF(-4.500000) : -2.710126
Error: 0.854826
----------------------------------------
----------------------------------------
   Step: 3
F(-4.000000):-2.170397
LF(-4.000000) : -2.870330
Error: 0.699933
----------------------------------------
----------------------------------------
   Step: 4
F(-3.500000):-3.220971
LF(-3.500000) : -3.152133
Error: 0.068837
----------------------------------------
----------------------------------------
   Step: 5
F(-3.000000):-3.404212
LF(-3.000000) : -3.404212
Error: 0.000000
----------------------------------------
----------------------------------------
   Step: 6
F(-2.500000):-2.287615
LF(-2.500000) : -3.510088
Error: 1.222473
----------------------------------------
----------------------------------------
   Step: 7
F(-2.000000):-1.195510
LF(-2.000000) : -3.388132
Error: 2.192621
----------------------------------------
----------------------------------------
   Step: 8
F(-1.500000):-1.385872
LF(-1.500000) : -2.991563
Error: 1.605691
----------------------------------------
----------------------------------------
   Step: 9
F(-1.000000):-2.308448
LF(-1.000000) : -2.308448
Error: 0.000000
----------------------------------------
----------------------------------------
   Step: 10
F(-0.500000):-2.306193
LF(-0.500000) : -1.361702
Error: 0.944491
----------------------------------------
----------------------------------------
   Step: 11
F(0.000000):0.000000
LF(0.000000) : -0.209089
Error: 0.209089
----------------------------------------
----------------------------------------
   Step: 12
F(0.500000):2.306193
LF(0.500000) : 1.056781
Error: 1.249412
----------------------------------------
----------------------------------------
   Step: 13
F(1.000000):2.308448
LF(1.000000) : 2.308448
Error: 0.000000
----------------------------------------
----------------------------------------
   Step: 14
F(1.500000):1.385872
LF(1.500000) : 3.383604
Error: 1.997732
----------------------------------------
----------------------------------------
   Step: 11
F(0.000000):0.000000
LF(0.000000) : -0.209089
Error: 0.209089
----------------------------------------
----------------------------------------
   Step: 12
F(0.500000):2.306193
LF(0.500000) : 1.056781
Error: 1.249412
----------------------------------------
----------------------------------------
   Step: 13
F(1.000000):2.308448
LF(1.000000) : 2.308448
Error: 0.000000
----------------------------------------
----------------------------------------
   Step: 14
F(1.500000):1.385872
LF(1.500000) : 3.383604
Error: 1.997732
----------------------------------------
----------------------------------------
   Step: 15
F(2.000000):1.195510
LF(2.000000) : 4.085094
Error: 2.889584
----------------------------------------
----------------------------------------
   Step: 16
F(2.500000):2.287615
LF(2.500000) : 4.180914
Error: 1.893299
----------------------------------------
----------------------------------------
   Step: 17
F(3.000000):3.404212
LF(3.000000) : 3.404212
Error: 0.000000
----------------------------------------
```

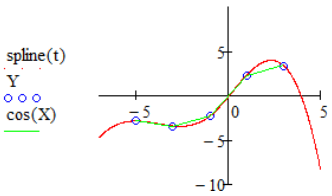
Розв’язок інтерполяції у MathCad

$$X := \begin{pmatrix} -5 \\ -3 \\ -1 \\ 1 \\ 3 \end{pmatrix} \qquad Y := \begin{pmatrix} -2.85 \\ -3.4 \\ -2.3 \\ 2.3 \\ 3.4 \end{pmatrix}$$

$s := \text{cspline}(X, Y)$

$$\cos(t) := \sin\left(\frac{5 \cdot X}{2}\right) + \sqrt[3]{5 \cdot X}$$

$\text{spline}(t) := \text{interp}(s, X, Y, t)$



.

Лістинг програми:

```
#include <cmath>
#include <iomanip>
#include <iostream>
#include <ostream>

using namespace std;


//////////////////////
const int A = 5,
          K = 0,
          ArraySize=5;
float Array[ArraySize] = {-5+K,-3+K,-1+K,1+K,3+K};
float FArray[ArraySize];
//////////////////////
float Function(float X)
{
    return sin((2.5)*X)+cbrt(X*5);
}

void Lagranj();
void Spline();

int main()
{
    cout.setf(ios::fixed);
    cout.precision(6);
    cout << "\tData:" << endl << "X:    ";
    for(int i=0;i<ArraySize;i++)
        cout << setw(11) << Array[i];
    cout << endl << "F(X): ";
    for(int i=0;i<ArraySize;i++)
    {
        FArray[i] = Function(Array[i]);
        cout << setw(11) << FArray[i];
    }
    cout << endl;


    //////////////////////
    Lagranj();
    Spline();
    //////////////////////
    return 0;
}
float LagranjFunction(float X)
{
    float Result = 0;
    for (int i = 0; i < ArraySize; i++)
    {
        float L = 1;
        for (int j = 0; j < ArraySize; j++)
        {
            if (i == j) continue;

            L *= (X - Array[j]);
            L /= (Array[i] - Array[j]);
```

```
        }
        L *= FArray[i];
        Result += L;
    }
    cout << "LF(" << X << ") : " << Result << endl;
    return Result;
}

void Lagranj()
{
    float temp=Array[0];
    int counter=0;
    cout << "-----------|Lagranj Method|------------" << endl;
    do
    {
        cout << "---------------------------------------" << endl << "   Step: " <<
++counter << endl;
        cout << "F("<<temp<<"):" << Function(temp) << endl;
        cout << "Error: " << (abs(Function(temp)-LagranjFunction(temp))) << endl;
        cout << "---------------------------------------" << endl;
        temp+=0.5;
    }while (temp>=Array[0] && temp<=Array[ArraySize-1]);
}

//////////////////////////////////////////////////////


float* Sweep(float** A, float* y, int n)
{
 float* alpha = new float[n];
 float* beta = new float[n];
 float* x = new float[n];
 for (int i = 0; i < n; i++)
 {
     if (i == 0)
     {
         alpha[i] = (-1) * (A[i][i + 1] / A[i][i]);
         beta[i] = y[i] / A[i][i];
     }
     else
     {
         float div = (A[i][i - 1] * alpha[i - 1]) + A[i][i];
         alpha[i] = (-1) * (A[i][i + 1] / div);
         beta[i] = (y[i] - (A[i][i - 1] * beta[i - 1])) / div;
     }
 }
 for (int i = n - 1; i >= 0; i--)
     x[i] = (i == n - 1) ? beta[i] : alpha[i] * x[i + 1] + beta[i];

 return x;
}
//////////////////////////////////////////////////////
void SplineFunction(float temp)
{
    float* a = new float[ArraySize - 1];
    float* c = new float[ArraySize - 1];
    float* d = new float[ArraySize - 1];
    float* b = new float[ArraySize - 1];
    float* h = new float[ArraySize - 1];
    float* vec = new float[ArraySize];
    float** matrix = new float* [ArraySize - 1];
```

```
    for (int i = 0; i < ArraySize - 1; i++)
    matrix[i] = new float[ArraySize - 1];
    for (int i = 0; i < ArraySize - 1; i++)
    {
         a[i] = FArray[i];
         h[i] = Array[i + 1] - Array[i];
         vec[i] = 0;
         for (int j = 0; j < ArraySize; j++)
            matrix[i][j] = 0;
    }
    for (int i = 0; i < ArraySize- 1; i++)
    {

         matrix[i][i - 1] = h[i - 1];
         matrix[i][i] = 2 * (h[i - 1] + h[i]);
         matrix[i][i + 1] = h[i];
         vec[i] = 3 * (((FArray[i + 1] - FArray[i]) / h[i]) + ((FArray[i] - FArray[i
- 1]) / h[i -1]));
    }
    c = Sweep(matrix, vec, ArraySize - 1);
    for (int i = 0; i < ArraySize - 1; i++)
    {
         if (i != ArraySize - 2)
         {
             d[i] = (c[i + 1] - c[i]) / (3 * h[i]);
             b[i] = ((FArray[i + 1] - FArray[i]) / h[i]) - ((h[i] * (c[i + 1] + 2 *
c[i])) / 3);
         }
         else
         {
             d[i] = (-1) * (c[i] / (3 * h[i]));
             b[i] = ((FArray[i] - FArray[i - 1]) / h[i]) - ((2 * h[i] * c[i]) / 3);
         }
    }
    int k = 0;
    for (int i = 1; i < ArraySize; i++)
        if (temp == Array[i])
            k = i - 1;

    cout << "X : " << temp << endl << "\ta : " << a[k] << "\tb :" << b[k] << "\tc :"
<< c[k] << "\td :" << d[k] << endl;
    printf("Pn(%.1f) = %.4f + %.4f * (X - %.4f) + %.4f * (X - %.4f)^2 + %.4f * (X -
%.4f)^3", temp,
    a[k], b[k], Array[k], c[k], Array[k], d[k], Array[k]);
}

void Spline()
{
    cout << "---------------|Splines|---------------" << endl;
    for(int i=1;i<ArraySize;i++)
    {
        cout << "----------------------------------------" << endl << "   Step: " << i
<< endl;
        SplineFunction(Array[i]);
        cout << endl << "----------------------------------------" << endl;
    }
}
```