Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

Лабораторна робота №4

з дисципліни «Спеціальні розділи математики»

на тему

«Обчислення власних значень та власних векторів матриць»

Виконав:

студент гр. ІС-02

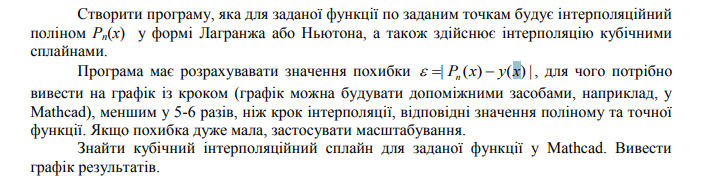
Плостак Ілля

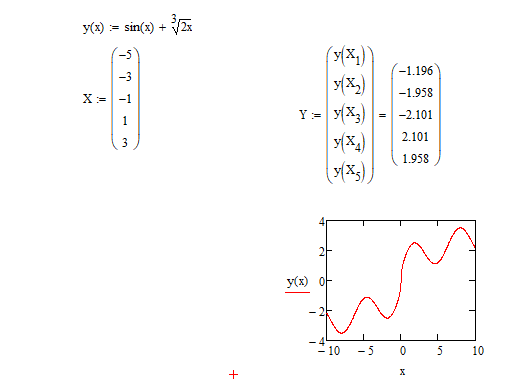
Викладач:

доц. Рибачук Л.В.

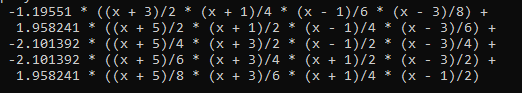
Київ – 2021

**1. Завдання**

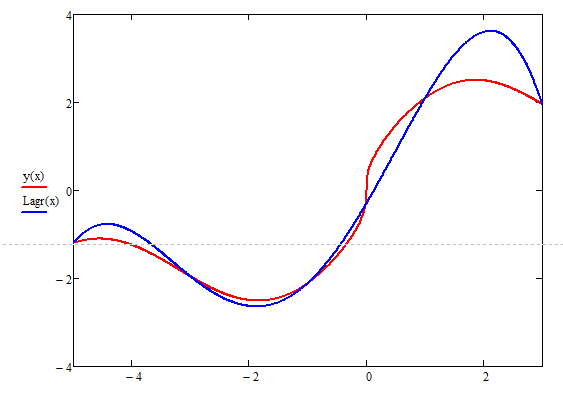




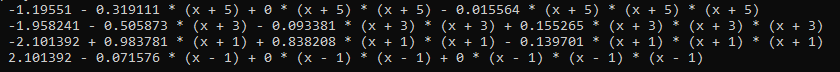
**2. Вигляд поліному Лагранжа**



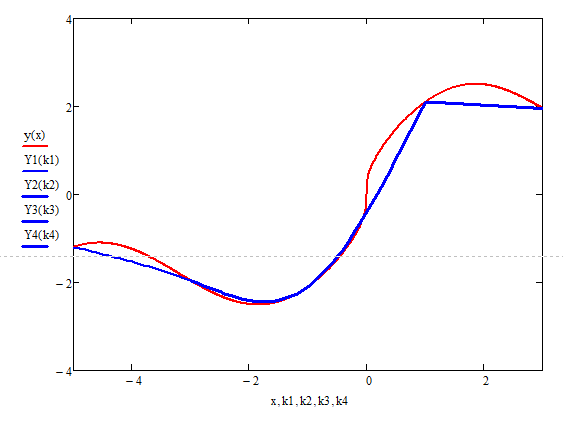
**3. Порівняльний графік функції та інтерполяційного поліному:**



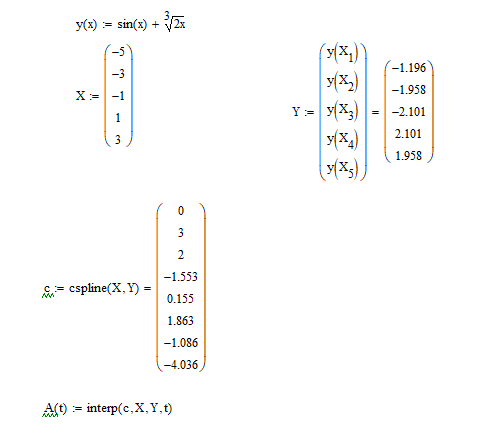
**4. Сплайни:**

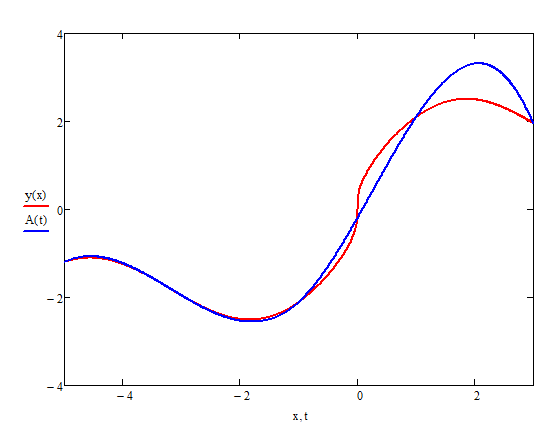


**5. Порівняльний графік функції та сплайн-інтерполяції:**



**6. Розв’язок інтерполяції сплайнами у Mathcad:**





**7. Лістинг програми:**

Program.cs:

using System;

using MyClass;

namespace Lab5

{

class Program

{

static void Main(string[] args)

{

double[] arrX = { -5, -3, -1, 1, 3 };

Func<double, double> FuncToPolynom = x => Math.Sin(x) + Pow(2 \* x, 1.0 / 3.0);

LagrangePolynom poly1 = new LagrangePolynom(arrX, FuncToPolynom);

poly1.CalculateLagrange();

Console.WriteLine("Result polynom is:");

Console.WriteLine(poly1);

Console.WriteLine();

QuadraticPolynom poly2 = new QuadraticPolynom(arrX, FuncToPolynom);

Console.WriteLine("Result polynoms are:");

for (int i = 0; i < poly2.Length; i++)

Console.WriteLine("\t{0}", poly2[i]);

Console.WriteLine();

for (double i = arrX[0]; i <= arrX[arrX.Length - 1]; i += (arrX[1] - arrX[0]) / 5)

Console.WriteLine("Error of Lagrange polynom in {0}: {1}", Math.Round(i, 1), Math.Abs(Math.Round((RoflanLagrangeCalculate(Math.Round(i, 1)) - FuncToPolynom(Math.Round(i, 1))), 6)));

Console.WriteLine();

for (double i = arrX[0]; i <= arrX[arrX.Length - 1]; i += (arrX[1] - arrX[0]) / 5)

{

if (i < arrX[1])

Console.WriteLine("Error of Spline polynom in {0}: {1}", Math.Round(i, 1), Math.Abs(Math.Round((RoflanSplineCalculate(Math.Round(i, 1), 1) - FuncToPolynom(Math.Round(i, 1))), 6)));

else if (i < arrX[2])

Console.WriteLine("Error of Spline polynom in {0}: {1}", Math.Round(i, 1), Math.Abs(Math.Round((RoflanSplineCalculate(Math.Round(i, 1), 2) - FuncToPolynom(Math.Round(i, 1))), 6)));

else if (i < arrX[3])

Console.WriteLine("Error of Spline polynom in {0}: {1}", Math.Round(i, 1), Math.Abs(Math.Round((RoflanSplineCalculate(Math.Round(i, 1), 3) - FuncToPolynom(Math.Round(i, 1))), 6)));

else

Console.WriteLine("Error of Spline polynom in {0}: {1}", Math.Round(i, 1), Math.Abs(Math.Round((RoflanSplineCalculate(Math.Round(i, 1), 4) - FuncToPolynom(Math.Round(i, 1))), 6)));

}

Console.ReadKey();

}

public static double Pow(double x, double power)

{

bool IsThereMinus = false;

if (x < 0)

IsThereMinus = true;

double numOut = Math.Pow(Math.Abs(x), power);

if (IsThereMinus)

return -numOut;

else

return numOut;

}

public static double RoflanLagrangeCalculate(double x)

{

return -1.19551 \* ((x + 3) / 2 \* (x + 1) / 4 \* (x - 1) / 6 \* (x - 3) / 8) +

1.958241 \* ((x + 5) / 2 \* (x + 1) / 2 \* (x - 1) / 4 \* (x - 3) / 6) -

2.101392 \* ((x + 5) / 4 \* (x + 3) / 2 \* (x - 1) / 2 \* (x - 3) / 4) -

2.101392 \* ((x + 5) / 6 \* (x + 3) / 4 \* (x + 1) / 2 \* (x - 3) / 2) +

1.958241 \* ((x + 5) / 8 \* (x + 3) / 6 \* (x + 1) / 4 \* (x - 1) / 2);

}

public static double RoflanSplineCalculate(double x, int index)

{

switch (index)

{

case 1:

return -1.19551 - 0.319111 \* (x + 5) + 0 \* (x + 5) \* (x + 5) - 0.015564 \* (x + 5) \* (x + 5) \* (x + 5);

case 2:

return -1.958241 - 0.505873 \* (x + 3) - 0.093381 \* (x + 3) \* (x + 3) + 0.155265 \* (x + 3) \* (x + 3) \* (x + 3);

case 3:

return -2.101392 + 0.983781 \* (x + 1) + 0.838208 \* (x + 1) \* (x + 1) - 0.139701 \* (x + 1) \* (x + 1) \* (x + 1);

case 4:

return 2.101392 - 0.071576 \* (x - 1) + 0 \* (x - 1) \* (x - 1) + 0 \* (x - 1) \* (x - 1) \* (x - 1);

}

return -1;

}

}

}

QuadraticPolynom.cs:

using System;

using System.Collections.Generic;

using System.Text;

namespace MyClass

{

public class QuadraticPolynom

{

private double[] \_arrX;

private double[] \_arrY;

private double[] \_arrA;

private double[] \_arrB;

private double[] \_arrC;

private double[] \_arrD;

private double[] \_arrH;

private Func<double, double> \_FuncY;

private string[] \_result;

public QuadraticPolynom(double[] arr, Func<double, double> func)

{

\_arrX = arr;

\_FuncY = func;

int n = \_arrX.Length - 1;

\_arrY = new double[\_arrX.Length];

\_arrA = new double[\_arrX.Length - 1];

\_arrB = new double[\_arrX.Length - 1];

\_arrC = new double[\_arrX.Length - 1];

\_arrD = new double[\_arrX.Length - 1];

\_arrH = new double[\_arrX.Length - 1];

\_result = new string[n];

\_result[0] = "Not calculated yet.";

for (int i = 0; i < \_arrY.Length; i++)

{

\_arrY[i] = \_FuncY(\_arrX[i]);

if (i != \_arrY.Length - 1)

{

\_arrA[i] = \_arrY[i];

}

if (i != 0)

{

\_arrH[i - 1] = \_arrX[i] - \_arrX[i - 1];

}

}

\_arrC[0] = 0;

\_arrC[\_arrC.Length - 1] = 0;

double[][] matrix = new double[2][];

matrix[0] = new double[2];

matrix[1] = new double[2];

matrix[0][0] = 2 \* (\_arrH[1] + \_arrH[2]); matrix[0][1] = \_arrH[2];

matrix[1][0] = \_arrH[2]; matrix[1][1] = 2 \* (\_arrH[2] + \_arrH[3]);

double[] rightVec = { 3 \* (((\_arrY[2] - \_arrY[1]) / \_arrH[1]) - ((\_arrY[1] - \_arrY[0]) / \_arrH[0])),

3 \* (((\_arrY[3] - \_arrY[2]) / \_arrH[2]) - ((\_arrY[2] - \_arrY[1]) / \_arrH[1])) };

SystemOfEq Sys = new SystemOfEq(matrix, rightVec);

\_arrC[1] = Sys.\_X[0];

\_arrC[2] = Sys.\_X[1];

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

{

if(i == n - 1)

\_arrD[i] = (-\_arrC[i]) / (3 \* \_arrH[i]);

else

\_arrD[i] = (\_arrC[i + 1] - \_arrC[i]) / (3 \* \_arrH[i]);

}

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

{

if (i == n - 1)

\_arrB[i] = ((\_arrY[i + 1] - \_arrY[i]) / \_arrH[i]) - (2 \* \_arrH[i] \* \_arrC[i] / 3);

else

\_arrB[i] = ((\_arrY[i + 1] - \_arrY[i]) / \_arrH[i]) - (\_arrH[i] / 3 \* (\_arrC[i + 1] + 2 \* \_arrC[i]));

}

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

{

\_result[i] = Math.Round(\_arrA[i], 6).ToString();

if (\_arrB[i] < 0)

\_result[i] += " - ";

else

\_result[i] += " + ";

\_result[i] += Math.Abs(Math.Round(\_arrB[i], 6)).ToString() + " \* (x";

if (\_arrX[i] < 0)

\_result[i] += " + ";

else

\_result[i] += " - ";

\_result[i] += Math.Abs(Math.Round(\_arrX[i], 6)).ToString() + ")";

if (\_arrC[i] < 0)

\_result[i] += " - ";

else

\_result[i] += " + ";

\_result[i] += Math.Abs(Math.Round(\_arrC[i], 6)).ToString() + " \* (x";

if (\_arrX[i] < 0)

\_result[i] += " + ";

else

\_result[i] += " - ";

\_result[i] += Math.Abs(Math.Round(\_arrX[i], 6)).ToString() + ")";

\_result[i] += " \* (x";

if (\_arrX[i] < 0)

\_result[i] += " + ";

else

\_result[i] += " - ";

\_result[i] += Math.Abs(Math.Round(\_arrX[i], 6)).ToString() + ")";

if (\_arrD[i] < 0)

\_result[i] += " - ";

else

\_result[i] += " + ";

\_result[i] += Math.Abs(Math.Round(\_arrD[i], 6)).ToString() + " \* (x";

if (\_arrX[i] < 0)

\_result[i] += " + ";

else

\_result[i] += " - ";

\_result[i] += Math.Abs(Math.Round(\_arrX[i], 6)).ToString() + ")";

\_result[i] += " \* (x";

if (\_arrX[i] < 0)

\_result[i] += " + ";

else

\_result[i] += " - ";

\_result[i] += Math.Abs(Math.Round(\_arrX[i], 6)).ToString() + ")";

\_result[i] += " \* (x";

if (\_arrX[i] < 0)

\_result[i] += " + ";

else

\_result[i] += " - ";

\_result[i] += Math.Abs(Math.Round(\_arrX[i], 6)).ToString() + ")";

}

}

public string this[int index]

{

get => this.\_result[index];

}

public int Length { get => this.\_result.Length; }

}

}

LagrangePolynom.cs:

using System;

namespace MyClass

{

public class LagrangePolynom

{

private double[] \_arrX;

private Func<double, double> \_FuncY;

private string \_result;

public LagrangePolynom(double[] arr, Func<double, double> func)

{

\_arrX = arr;

\_FuncY = func;

\_result = "Not calculated yet.";

}

public void CalculateLagrange()

{

string strOut = "";

for (int k = 0; k < \_arrX.Length; k++)

{

double curY = \_FuncY(\_arrX[k]);

strOut += "\t";

if (CountMinuses(\_arrX[k], \_arrX) % 2 == 1)

curY = -curY;

if(curY >= 0)

strOut += " ";

strOut += Math.Round(curY, 6).ToString();

strOut += " \* (";

bool flag = false;

for (int i = 0; i < \_arrX.Length; i++)

{

if (i != k)

{

if(flag)

strOut += " \* ";

strOut += "(x";

if (\_arrX[i] < 0)

strOut += " + " + (-\_arrX[i]).ToString();

else

strOut += " - " + \_arrX[i].ToString();

strOut += ")/" + (Math.Abs(\_arrX[k] - \_arrX[i])).ToString();

flag = true;

}

}

strOut += ")";

if (k != \_arrX.Length - 1)

strOut += " +\n";

}

\_result = strOut;

}

public override string ToString()

{

return this.\_result;

}

private static int CountMinuses(double num, double[] arr)

{

int counter = 0;

for (int i = 0; i < arr.Length; i++)

{

if(num - arr[i] < 0)

counter++;

}

return counter;

}

}

}

SystemOfEq.cs:

using System;

using System.Collections.Generic;

using System.Text;

namespace MyClass

{

public class SystemOfEq

{

private double[][] \_Matrix;

private double[] \_rightVector;

private int[] \_numbersX;

private int \_size;

public double[] \_X { get; private set; }

public SystemOfEq(double[][] matrix, double[] rightVector)

{

\_Matrix = matrix;

\_rightVector = rightVector;

\_numbersX = new int[rightVector.Length];

for (int i = 0; i < \_numbersX.Length; i++)

{

\_numbersX[i] = i;

}

\_X = new double[rightVector.Length];

\_size = rightVector.Length;

gaussAlgorithmTriangle();

gaussAlgorithmGetSolution();

}

private int[] getIndexOfMaxElement(int n)

{

int[] keys = new int[2];

double Max = Math.Abs(\_Matrix[n][n]);

keys[0] = n;

keys[1] = n;

for (int i = n; i < \_size; i++)

{

for (int j = n; j < \_size; j++)

{

if (Math.Abs(\_Matrix[i][j]) > Max)

{

Max = Math.Abs(\_Matrix[i][j]);

keys[0] = i;

keys[1] = j;

}

}

}

return keys;

}

private double[] getMultipliersArray(int[] key, int n)

{

double[] arr = new double[\_size];

for (int i = 0; i < \_size; i++)

{

if (i != key[0] && i >= n)

{

arr[i] = \_Matrix[i][key[1]] / \_Matrix[key[0]][key[1]];

}

else

{

arr[i] = 0;

}

}

return arr;

}

private void swapRows(int numRow1, int numRow2)

{

for (int i = 0; i < \_size; i++)

{

double tmp = \_Matrix[numRow1][i];

\_Matrix[numRow1][i] = \_Matrix[numRow2][i];

\_Matrix[numRow2][i] = tmp;

}

double temp = \_rightVector[numRow1];

\_rightVector[numRow1] = \_rightVector[numRow2];

\_rightVector[numRow2] = temp;

}

private void swapCols(int numCol1, int numCol2)

{

for (int i = 0; i < \_size; i++)

{

double temp = \_Matrix[i][numCol1];

\_Matrix[i][numCol1] = \_Matrix[i][numCol2];

\_Matrix[i][numCol2] = temp;

int tmp = \_numbersX[numCol1];

\_numbersX[numCol1] = \_numbersX[numCol2];

\_numbersX[numCol2] = tmp;

}

}

private void gaussStep(int[] keys, double[] multipl, int n)

{

for (int i = n; i < \_size; i++)

{

for (int j = n; j < \_size; j++)

{

\_Matrix[i][j] -= multipl[i] \* \_Matrix[keys[0]][j];

}

\_rightVector[i] -= multipl[i] \* \_rightVector[keys[0]];

}

swapRows(n, keys[0]);

swapCols(n, keys[1]);

}

private void gaussAlgorithmTriangle()

{

for (int n = 0; n < \_size; n++)

{

int[] maxKeys = getIndexOfMaxElement(n);

double maxA = \_Matrix[maxKeys[0]][maxKeys[1]];

//Console.Write("\nCurrent max element: ");

//print(maxA);

if (maxA == 0)

{

Console.WriteLine("This system has no solutions!");

break;

}

double[] m = getMultipliersArray(maxKeys, n);

//Console.Write("\nCurrent vector of multipliers: ");

//print(m, \_size);

gaussStep(maxKeys, m, n);

//Console.WriteLine("\nCurrent Matrix:\n");

//print(\_Matrix, \_size);

//Console.Write("Current right vector: ");

//print(\_rightVector, \_size);

//Console.WriteLine("-----------------------------------------------------------------");

}

}

private void gaussAlgorithmGetSolution()

{

for (int n = \_size - 1; n >= 0; n--)

{

\_X[n] = (\_rightVector[n] - Solution(n)) / \_Matrix[n][n];

}

for (int i = 0; i < \_size; i++)

{

double temp = \_X[i];

\_X[i] = \_X[\_numbersX[i]];

\_X[\_numbersX[i]] = temp;

}

}

private double Solution(int n)

{

double output = 0;

for (int i = \_size - 1; i > n; i--)

{

output += \_Matrix[n][i] \* \_X[i];

}

return output;

}

public static void print(double[][] Matrix, int size)

{

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

{

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

{

Console.Write("{0:11} ", Math.Round(Matrix[i][j], 6));

}

Console.WriteLine();

}

}

public static void print(double[] Vector, int size)

{

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

{

Console.Write("{0} ", Math.Round(Vector[i], 6));

}

Console.WriteLine();

}

public static void print(double number)

{

Console.WriteLine(number);

}

}

}