

***ФАКУЛТЕТ ПО ИЗЧИСЛИТЕЛНА ТЕХНИКА И АВТОМАТИЗАЦИЯ***

*Катедра* ***„Компютърни науки и технологии (КНТ)“***

*Специалност* ***„Компютърни системи и технологии“***

***КУРСОВ ПРОЕКТ ПО СИСТЕМЕН АНАЛИЗ***

*Изготвил****: Хакан Х. Осман*** *Преподавател:* ***ас. Мая Тодорова***

*Ф№****: 20621333***

*Група и курс:* ***2 а, III курс***

***Дата: 12.12.2022 г.***

***гр. Варна***

***СЪДЪРЖАНИЕ***

* Задание на курсовия проект 3
* Теоретична част: постановка на метода на най-малките квадрати 3
* Практическа част: описание на алгоритъма, чрез формули и блокова схема 12
* Изводи и заключения 15
* Изполвана литература 15
* Приложение - сорс кода на програмата 15

# Задание на проекта

Да се състави програма по метода на най-малките квадрати за определяне коефициентите на полиноми от 1 и 2 ред:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Х** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| **Y** | 0 | 1.5 | 1.9 | 2.8 | 3.5 | 4.9 | 5.9 | 7.5 | 9.2 | 11.1 |

Да се определи коя е най-подходящата апроксимация.

# Теоретична част – постановка на метода на най-малките квадрати. Теорията да се илюстрира с подходящи премери

## Методът на най-малките квадрати (НМК) в [числения анализ](https://bg.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%B5%D0%BD_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7) е един от най-разпространените методи за решаване на системи уравния с повече неизвестни от броя на уравненията. Полученото решение е [апроксимация](https://bg.wikipedia.org/wiki/%D0%90%D0%BF%D1%80%D0%BE%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D1%86%D0%B8%D1%8F), при която се минимизира сумата от квадратите на грешката, получена за всяко едно уравнение.

Полиномната апроксимация е метод за апроксимиране на стойността на функция в точка чрез полином, минаващ през дадени стойности (точки). Главният източник на грешки е грешката от прекъсване (стойностите между две точки, в които няма измерване).

Прецизната дефиниция на апроксимации по метода на най-малките квадрати е следната: Нека f(x) е функция, а {xi}, i=1, 2, …, n, е редица от зададени точки, в които са измерени стойностите на f(x), като тези измерени стойности съдържат грешки. Нека с fi се означи точната стойност на f(xi) в точката xi, а с i f - измерената стойност в същата точка. Дефинира се разликата (грешка, остатък) E = f − f . Предполага се, че грешките в различните точки са независими. Нека {Φ (x)}, j = 0,1,... j е редица от функции, дефинирани за всяко xi. Целта е да се апроксимира fi с линейни комбинации на {Φ j (x)}

Където коефициентите трябва да се определят така, че да се минимизира



Функцията *w(x)* се нарича тегловна функция, за която се предполага, че е неотрицателна за всяко *xi*. Величината *Ri* се нарича остатък в *xi*.

След определяне на  така, че да се удовлетворява горното условие, се получава апроксимацията

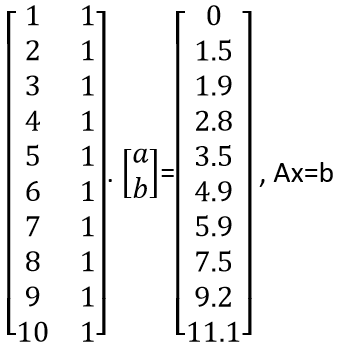


която се нарича апроксимация по метода на най-малките квадрати.

Пример по задание

Стъпки:

1. Избира се математически модел, който да даде зависимостта на пареметрите Х към параметрите У. Като първо приближение се избира линеен модел У=а.х+b. На базата на този модел се построява следната система линейни алгебричнои уравнения, в които броят на уравненията е по голям от броя на неизвестните коефициенти.

0=a.1+b

1.5=a.2+b

1.9=a.3+b

2.8=a.4+b

3.5=a.5+b или

4.9=a.6+b

5.9=a.7+b

7.5=a.8+b

9.2=a.9+b

11.1=a.10+b

## Фрагмент от кода:

int col = 0;

for(int row = 0; row < argumentsX.Length; row++)

{

matrixAPolinomOne[row, col] = argumentsX[row];

}

for(int row = 0; row < argumentsX.Length; row++)

{

matrixAPolinomOne[row, col+1] = 1;

}

2.Транспонира се матрицата А{\displaystyle n\times m}.

АT =

Фрагмент от кода:

int[,] matrixTransport = new int[matrixA.GetLength(1), matrixA.GetLength(0)];

for(int col = 0; col < matrixA.GetLength(1); col++)

{

for(int row = 0; row < matrixA.GetLength(0); row++)

{

matrixTransport[col, row] = matrixA[row, col];

}

}

return matrixTransport;

1. Умножават се матриците АT.A;

АTA= АT.A=

Фрагмент от кода:

int[] currentColum = new int[10];

List<int> resultOfMultiplication = new List<int>(0);

for(int col = 0; col < matrixAPolinom.GetLength(1); col++)

{

for (int row = 0; row < matrixAPolinom.GetLength(0); row++)

{

currentColum[row] = matrixAPolinom[row, col];

}

int currentResult = 0;

for (int currentRow = 0; currentRow < transportMatrixAPolinom.GetLength(0); currentRow++)

{

for (int currentCol = 0; currentCol < transportMatrixAPolinom.GetLength(1); currentCol++)

{

currentResult += currentColum[currentCol] \* transportMatrixAPolinom[currentRow, currentCol];

}

resultOfMultiplication.Add(currentResult);

currentResult = 0;

}

1. Изчислява се детерминанта на матрицата АТ.А;

det(ATA)=825

Фрагмент от кода:

int size = multiplcateMatrixPolinomOne.GetLength(1);

int firstDiagonal = 1;

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

{

firstDiagonal \*= multiplcateMatrixPolinomOne[cur, cur];

}

int secondDiagonal = 1;

int currentRow = 0;

int currentCol = size - 1;

for (int cur = size - 1; cur >= 0; cur--)

{

secondDiagonal \*= multiplcateMatrixPolinomOne[currentRow, currentCol];

++currentRow;

--currentCol;

}

int determinant = firstDiagonal - secondDiagonal;

Проверява се дали детерминантата е различно от нула. Ако е различна от нула се изчислява обратна матрица, но ако е равна на нула, няма решение.

1. Намира се обратна матрица на АТА;

А1=(AT.A)-1=

Фрагмент от кода:

double[,] reverseMatirx = new double[multiplcateMatrixPolinomOne.GetLength(0),

multiplcateMatrixPolinomOne.GetLength(1)];

int swamp = multiplcateMatrixPolinomOne[0,0];

multiplcateMatrixPolinomOne[0, 0] = multiplcateMatrixPolinomOne[1, 1];

multiplcateMatrixPolinomOne[1, 1] = swamp;

multiplcateMatrixPolinomOne[0, 1] \*= -1;

multiplcateMatrixPolinomOne[1, 0] \*= -1;

double valueOfDetA = (1.00 / determinantOfPolinomOne);

for(int row = 0; row < multiplcateMatrixPolinomOne.GetLength(0); row++)

{

for (int col = 0; col < multiplcateMatrixPolinomOne.GetLength(1); col++)

{

reverseMatirx[row, col] =(multiplcateMatrixPolinomOne[row, col] \* valueOfDetA);

}

}

1. Умножава се матрицата АТ.У;

АТУ=АТ.У=

Фрагмент от кода:

double[,] resultMatrix = new double[transportMatrixA.GetLength(0), 1];

for (int row = 0; row < transportMatrixA.GetLength(0); row++)

{

double currentResult = 0;

for (int col = 0; col < argumentsY.Length; col++)

{

currentResult += (transportMatrixA[row, col] \* argumentsY[col]);

}

resultMatrix[row, 0] = currentResult;

}

1. Изчислява се Х, което дава коефициентите а и b.

Х=А1.АТУ

а=1.17 и b= -1.59

Фрагмент от кода:

double[,] coefficients = new double[multplicateMatrixY.GetLength(0),

multplicateMatrixY.GetLength(1)];

for (int row = 0; row < reverseMatrix.GetLength(0); row++)

{

double currentResult = 0;

for (int col = 0; col < reverseMatrix.GetLength(1); col++)

{

currentResult += reverseMatrix[row, col] \* multplicateMatrixY[col, 0];

}

coefficients[row, 0] = currentResult;

}

1. Изчисляват се оценките за реда.

Фрагмент от кода:

double[] grades = new double[argumentsX.Length];

for(int cur = 0; cur < argumentsX.Length; cur++)

{

grades[cur] = (coefficientsOfPolynomOne[0, 0] \* argumentsX[cur]) +

coefficientsOfPolynomOne[1, 0];

}

1. Изчислява се квадтратичната грешка.

δm = 1.81

Фрагмент от кода:

double[] calculateGrades = new double[argumentsY.Length];

for (int cur = 0; cur < argumentsY.Length; cur++)

{

calculateGrades[cur] = Math.Pow((argumentsY[cur] - grades[cur]),2);

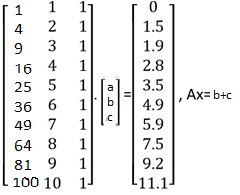
}

double sum = calculateGrades.Sum();

double quadraticError = Math.Sqrt(sum);

Стъпки:

1. Като второ приближение се избира матаматически модел от по-висок ред в случя от втори ред У= а.х2 + b.x + c. На базата на този модел се построява следната система линейни алгебричнои уравнения, в които броят на уравненията е по голям от броя на неизвестните коефициенти.

0=a.1+b+с

1.5=a.4+b+с

1.9=a.9+b+с

2.8=a.16+b+с

3.5=a.25+b+с или

4.9=a.36+b+с

5.9=a.49+b+с

7.5=a.64+b+с

9.2=a.81+b+с

11.1=a.100+b+с

По аналогичен начин се спразва алгоритъма.

1. Транспонира се матрицата А.

AT =

Фрагмент от кода:

for (int row = 0; row < argumentsX.Length; row++)

{

matrixAPolinomTwo[row, col] = (int)(Math.Pow(argumentsX[row],2));

}

for (int row = 0; row < argumentsX.Length; row++)

{

matrixAPolinomTwo[row, col+1] =argumentsX[row];

}

for (int row = 0; row < argumentsX.Length; row++)

{

matrixAPolinomTwo[row, col+2] = 1;

}

1. Умножават се матриците АT.A

АT.A =

Фрагмент от кода:

for (int row = 0; row < matrixAPolinom.GetLength(0); row++)

{

currentColum[row] = matrixAPolinom[row, col];

}

int currentResult = 0;

for (int currentRow = 0; currentRow < transportMatrixAPolinom.GetLength(0); currentRow++)

{

for (int currentCol = 0; currentCol < transportMatrixAPolinom.GetLength(1); currentCol++)

{

currentResult += currentColum[currentCol] \* transportMatrixAPolinom[currentRow, currentCol];

}

resultOfMultiplication.Add(currentResult);

currentResult = 0;

}

1. Изчислява се детерминанта на матрицата АТ.А по правило на триъгълниците

det(ATA)=435600

Фрагмент от кода:

int determinant = (matrix[0, 0] \* matrix[1, 1] \* matrix[2, 2]) +

(matrix[0, 1] \* matrix[1, 2] \* matrix[2, 0])

+ (matrix[0, 2] \* matrix[1, 0] \* matrix[2, 1]) -

(matrix[0, 2] \* matrix[1, 1] \* matrix[2, 0]) -

(matrix[0, 0] \* matrix[1, 2] \* matrix[2, 1]) -

(matrix[0, 1] \* matrix[1, 0] \* matrix[2, 2]);

Проверява се дали детерминантата е различно от нула. Ако е различна от нула се изчислява обратна матрица, но ако е равна на нула, няма решение.

1. Намира се обратна матрица на АТА по адюнгирани количества;

АТА-1=

Фрагмент от кода:

for(int row = 0; row < multiplicatePolinomTwo.GetLength(0); row++)

{

for(int col = 0; col < multiplicatePolinomTwo.GetLength(1); col++)

{

List<int> valuesOfCurrentRow = CalculateCurrentResult(multiplicatePolinomTwo, row,col);

FullMatrixFromList(valuesOfCurrentRow, currentSquareMatrix);

int sumOfColAndRow = (col + 1)+(row + 1);

int determinantOfSquare = GetDeterminantAtPolinomOne(currentSquareMatrix);

int currentValue = (int)Math.Pow(-1, sumOfColAndRow) \* (determinantOfSquare);

results.Add(currentValue);

}

}

1. Умножава се матрицата АТ.У.

АТУ=АТ.У=

1. Изчислява се Х, което дава коефициентите а,b и c.

Х=А1.АТУ

а=0.07 , b=0.38 и c=-0.01

1. Изчисляват се оценките за реда.

9. Изчислява се квадтратичната грешка.

δm = 0.76

Друг Пример

Задача. В една държава са правени преброявания на нейните жители през 10 години. Резултатите от преброяването са дадени в следната таблица:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Данни от преброяване на населението** | | | | | | | | |
| **Година** | 1945 | 1955 | 1965 | 1975 | 1985 | 1995 | 2005 | 2015 |
| **Млн. жители** | 4,2 | 5,3 | 5,9 | 7,5 | 9,1 | 9,9 | 10,7 | 11,4 |

Да се даде оценка на населението през *1990*, *2007* и прогноза за броя през *2022* година. Както се вижда от таблицата по-горе, през посочените години не е имало (1990 и 2007) и няма как да е имало (2022) преброяване.

За решаването на тази задача е необходимо да се избере подходящ математически модел, който да даде достатъчно точно зависимостта на населението от годината на преброяванетоЗа решаването на тази задачата от началото на темата е необходимо да се избере подходящ математически модел, който да даде зависимостта на населението от годината на преброяването. Нека като първо приближение бъде избран линеен модел. Т.е.: , където *y* е населението в млн. жители, *x* – годината на преброяването, *a* и *b* – постоянни коефициенти, които трябва да бъдат определени. Следователно известните величини *y* и *x* трябва да послужат за определяне на неизвестните *a* и *b*. броят на точките (годините, в които е имало преброяване) са 8, а неизвестните – 2. Това ще даде възможност за апроксимации с полиноми и от по-висок ред.

На базата на този модел може да се построи следната система линейни алгебрични уравнения, които броят на уравненията ще бъде по-голям от броя на неизвестните:

 или в матричен вид  или 

Възниква необходимостта от решаване на система линейни уравнения, в която броят на уравненията е по-голям от броя на неизвестните. В съответствие с метода на най-малките квадрати него е необходимо матрицата *А* да се транспонира и да се умножат отляво дясната и лявата част на системата. В резултат на това действие се получава квадратна система линейни алгебрични уравнения от вида: . Окончателното решение се дава с израза: .

Вследствие на решението на системата уравнения се получават оценките на параметрите *a* и *b* съответно 0,1083 и -204,9.

След това се пресмята квадратичната грешка като . В таблицата по-долу са дадени резултатите от оценката на параметрите и квадратичната грешка.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
|  | **Реални данни** | 4,200 | 5,300 | 5,900 | 7,500 | 9,100 | 9,900 | 10,700 | 11,400 |  |
|  | **Оценки** | 4,208 | 5,292 | 6,375 | 7,458 | 8,542 | 9,625 | 10,708 | 11,792 |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  | **Квадратична грешка** | **0,88** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

Окончателните отговори на задачата се изчисляват на базата на получения модел: и са дадени в таблицата по-долу:

|  |  |  |  |
| --- | --- | --- | --- |
| **Година** | 1990 | 2007 | 2022 |
| **Млн. жители** | 9,08 | 11,47 | 13,09 |

Прави впечатление лошата оценка на броя на населението. Причината за това е избраният математически модел. Нека увеличим реда на модела. Т.е. изберем полином от трета степен за решаване на поставената задача.

, като за данните от задачата той ще изглежда:

 или 

След решаване на горната система, като се спазва същата последователност на решение на линеен модел по-горе, се получават следните резултати:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1945 | 1955 | 1965 | 1975 | 1985 | 1995 | 2005 | 2015 |  |
|  | **Реални данни** | 4,200 | 5,300 | 5,900 | 7,500 | 9,100 | 9,900 | 10,700 | 11,400 |  |
|  | **Оценки** | 4,239 | 5,081 | 6,217 | 7,505 | 8,802 | 9,965 | 10,854 | 11,324 |  |
|  | **Квадратична грешка** | 0,5222 |  |  | **Оценки** | **a** | **b** | **c** | **d** |  |
|  |  |  |  |  |  | -2E-05 | 0,14 | -273,5 | 2E+05 |  |
|  | **Година** | 1990 | 2007 | 2022 |  |  |  |  |  |  |
|  | **Млн. жители** | 9,41 | 11,24 | 11,14 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

Анализът на оценките и квадратичната грешка е в полза на модела от трети ред. В зависимост от характера и особеностите на решаваните задачи за оценка параметри могат да се използват и други модели (логаритмичен, експоненциален и др.)

# Практическа част: описание на алгоритъма чрез формули и блокова схема.

#### Блок схема на алгоритъма

Въвеждане на параметрите Х и Y

На базата на модел от първи ред се построява система линейни алгебрични уравнения

int[,] matrixA

matrixA

Транспониране на matrixA

matrixAT

Умножение на matrixA на matrixAT

multiplcateMatrix=matrixA\*matrixAT

multiplcateMatrix

Намиране на детерминанта

det (multiplcateMatrix)

det (multiplcateMatrix)

Детерминантата различна ли е от 0 ?

Няма решение !

Не

Да

Намиране на обратна матрица multiplcateMatrix-1

multiplcateMatrix-1

Умножава се матрицата АТУ=matrixAT\*У

АТУ

multiplcateMatrix-1\*АТУ

Извеждане на коефициентите

а и b

Изчислява се средната квадратична грешка по формулата

δm

δm

# Изводи и заключения

При избрания Като първо приближение линеен модел У=а.х+b след прилагане на метода на най-малките квадрати,коефициентите на на линейната функция се получават съответно a=1.17 и b= -1.59.След изчислени на квадратичната грешка по формулата δm като резултат се получава δm=1.81.При избрания модел от втори ред У=а.х2+b+c след прилагате на метода на най-малките квадрати коефициентите на функция от втори ред се получават съответно a=0.07 , b=0.38 и c=-0.01.Cлед изчислени на квадратичната грешка като резултат се получава δm=0.76.Следователно избрания модел от втори ред е по-доброто приближение и съответно по-подходящата апроксимация за съответната задача.

# Използавана литература

- <http://cs.tu-varna.bg/discipline/29>

- https://bg.wikipedia.org/wiki

# Приложение-сорс кода на програмата

using System;

using System.Collections.Generic;

using System.Linq;

namespace SystemAnalis

{

class Program

{

static void Main(string[] args)

{

Console.Write("X= ");

int[] argumentsX = Console.ReadLine()

.Split(' ', StringSplitOptions.RemoveEmptyEntries)

.Select(int.Parse)

.ToArray();

Console.Write("Y= ");

double[] argumentsY = Console.ReadLine()

.Split(' ', StringSplitOptions.RemoveEmptyEntries)

.Select(double.Parse)

.ToArray();

int[,] matrixAPolinomOne = new int[argumentsX.Length,2];

int[,] matrixPolinomTwo = new int[argumentsX.Length, 3];

FullMatrixA(matrixAPolinomOne,matrixPolinomTwo,argumentsX);

Console.WriteLine("First Look At matrix of polinom 1: ");

PrintMatrix(matrixAPolinomOne);

Console.Write("First Look At matrix of polinom 2: ");

Console.WriteLine();

PrintMatrix(matrixPolinomTwo);

int[,] transportMatrixAPolinomOne =TransopotingAlgorithm(matrixAPolinomOne);

int[,] transportMatrixAPolinomTwo = TransopotingAlgorithm(matrixPolinomTwo);

Console.WriteLine("===============");

Console.WriteLine("Second step after transport matrix of polinom 1: ");

PrintMatrix(transportMatrixAPolinomOne);

Console.WriteLine("Second step after transport matrix of polinom 2: ");

PrintMatrix(transportMatrixAPolinomTwo);

Console.WriteLine("===============");

List<int>firstRowResutOfMultoplication=MultiplicateMatrix(matrixAPolinomOne,transportMatrixAPolinomOne);

List<int>secondRowResutOfMultoplication= MultiplicateMatrix(matrixPolinomTwo, transportMatrixAPolinomTwo);

int[,] multiplcateMatrixPolinomOne = new int[matrixAPolinomOne.GetLength(1), matrixAPolinomOne.GetLength(1)];

int[,] multiplicatePolinomTwo = new int[matrixPolinomTwo.GetLength(1), matrixPolinomTwo.GetLength(1)];

FullMatrixFromList(firstRowResutOfMultoplication, multiplcateMatrixPolinomOne);

FullMatrixFromList(secondRowResutOfMultoplication, multiplicatePolinomTwo);

Console.WriteLine("After multiplication matrixA on matrixAT of polinom 1:");

PrintMatrix(multiplcateMatrixPolinomOne);

Console.WriteLine("After multiplication matrixA on matrixAT of polinom 2:");

PrintMatrix(multiplicatePolinomTwo);

Console.WriteLine("===============");

Console.WriteLine("Determinanant at polinom One:");

int determinantOfPolinomOne = GetDeterminantAtPolinomOne(multiplcateMatrixPolinomOne);

Console.WriteLine(determinantOfPolinomOne);

Console.WriteLine("Determinanant at polinom Two:");

int determinantOfPolinomTwo = RuleOfTriangle(multiplicatePolinomTwo);

Console.WriteLine(determinantOfPolinomTwo);

Console.WriteLine("===============");

double quadraticErrorOfPolinomOne = double.MaxValue;

double quadraticErrorOfPolinomTwo = double.MaxValue;

bool isZeroOfPolinomOne = CheckIsDeterminantDiffrentOfZero(determinantOfPolinomOne);

if (!isZeroOfPolinomOne)

{

Console.WriteLine("Reverse Matrix of PolinomOne:");

double[,] reverseMatrixOfPolinomOne = AlgorithmFindReverseMatrixOfPolinmOne

(multiplcateMatrixPolinomOne, determinantOfPolinomOne);

PrintDoubleMatrix(reverseMatrixOfPolinomOne);

Console.WriteLine("===============");

double[,] multplicateMatrixYOfPolinomTwo = multiplicationTransoprtMatrixOnMatrixB

(transportMatrixAPolinomOne, argumentsY);

Console.WriteLine("Multiplication of transportMatrixA on argumentsY of PolinomOne:");

PrintDoubleMatrix(multplicateMatrixYOfPolinomTwo);

Console.WriteLine("===============");

double[,] multplicateMatrixYOfPolinomOne = multiplicationTransoprtMatrixOnMatrixB

(transportMatrixAPolinomOne, argumentsY);

Console.WriteLine("Coefficient of PolynomOne:");

double[,] coefficientsOfPolynomOne = CalculateCoefficients

(reverseMatrixOfPolinomOne, multplicateMatrixYOfPolinomOne);

PrintDoubleMatrix(coefficientsOfPolynomOne);

double[] grades = GetGradesOfPolynomOne(coefficientsOfPolynomOne, argumentsX);

quadraticErrorOfPolinomOne = CalculateQuadraticError(grades, argumentsY);

Console.WriteLine($"The quadraticError of PolinomOne is: {quadraticErrorOfPolinomOne:f2}");

}

else

{

Console.WriteLine("No solition of polynomOne!!!");

}

bool isZeroOfPolinomTwo = CheckIsDeterminantDiffrentOfZero(determinantOfPolinomTwo);

if (!isZeroOfPolinomTwo)

{

Console.WriteLine("Reverse Matrix of PolinomTwo:");

List<int> valuesOfReverseMatrixOfPolinomTwo = AlgorithmFindReverseMatrixOfPolinmTwo

(multiplicatePolinomTwo);

int[,] reverseMatrixOfPolinomTwo = new int[3, 3];

FullMatrixFromList(valuesOfReverseMatrixOfPolinomTwo, reverseMatrixOfPolinomTwo);

double[,] finaleReverseMatrixOfPolinomTwo =

CalcualateFinaleReverseMatrix(reverseMatrixOfPolinomTwo, determinantOfPolinomTwo);

PrintDoubleMatrix(finaleReverseMatrixOfPolinomTwo);

Console.WriteLine("===============");

double[,] multplicateMatrixYOfPolinomTwo = multiplicationTransoprtMatrixOnMatrixB

(transportMatrixAPolinomTwo, argumentsY);

Console.WriteLine("Multiplication of transportMatrixA on argumentsY of PolinomTwo:");

PrintDoubleMatrix(multplicateMatrixYOfPolinomTwo);

Console.WriteLine("===============");

Console.WriteLine("Coefficient of PolynomTwo:");

double[,] coefficientsOfPolynomTwo = CalculateCoefficients

(finaleReverseMatrixOfPolinomTwo, multplicateMatrixYOfPolinomTwo);

PrintDoubleMatrix(coefficientsOfPolynomTwo);

double[] grades = GetGradesOfPolynomTwo(coefficientsOfPolynomTwo, argumentsX);

quadraticErrorOfPolinomTwo = CalculateQuadraticError(grades, argumentsY);

Console.WriteLine("===============");

Console.WriteLine($"The quadraticError of PolinomTwo is: {quadraticErrorOfPolinomTwo:f2}");

if (quadraticErrorOfPolinomOne < quadraticErrorOfPolinomTwo)

{

Console.WriteLine("A first-order polynomial is the better approximation!");

}

else

{

Console.WriteLine("A second-order polynomial is the better approximation!");

}

}

else

{

Console.WriteLine("No solition of polinomTwo!!!");

}

}

static void FullMatrixA(int[,] matrixAPolinomOne, int[,] matrixAPolinomTwo,

int[] argumentsX)

{

int col = 0;

for(int row = 0; row < argumentsX.Length; row++)

{

matrixAPolinomOne[row, col] = argumentsX[row];

}

for(int row = 0; row < argumentsX.Length; row++)

{

matrixAPolinomOne[row, col+1] = 1;

}

col = 0;

for (int row = 0; row < argumentsX.Length; row++)

{

matrixAPolinomTwo[row, col] = (int)(Math.Pow(argumentsX[row],2));

}

for (int row = 0; row < argumentsX.Length; row++)

{

matrixAPolinomTwo[row, col+1] =argumentsX[row];

}

for (int row = 0; row < argumentsX.Length; row++)

{

matrixAPolinomTwo[row, col+2] = 1;

}

}

static void PrintMatrix(int[,] matrixA)

{

for (int row = 0; row <matrixA.GetLength(0); row++)

{

for(int col = 0; col < matrixA.GetLength(1); col++)

{

Console.Write($"{matrixA[row,col]} ");

}

Console.WriteLine();

}

}

static void PrintDoubleMatrix(double[,] matrix)

{

for (int row = 0; row < matrix.GetLength(0); row++)

{

for (int col = 0; col < matrix.GetLength(1); col++)

{

Console.Write($"{matrix[row, col]:f2} ");

}

Console.WriteLine();

}

}

static int[,] TransopotingAlgorithm(int[,] matrixA)

{

int[,] matrixTransport = new int[matrixA.GetLength(1), matrixA.GetLength(0)];

for(int col = 0; col < matrixA.GetLength(1); col++)

{

for(int row = 0; row < matrixA.GetLength(0); row++)

{

matrixTransport[col, row] = matrixA[row, col];

}

}

return matrixTransport;

}

static List<int> MultiplicateMatrix(int[,] matrixAPolinom ,int[,] transportMatrixAPolinom)

{

int[] currentColum = new int[10];

List<int> resultOfMultiplication = new List<int>(0);

for(int col = 0; col < matrixAPolinom.GetLength(1); col++)

{

for (int row = 0; row < matrixAPolinom.GetLength(0); row++)

{

currentColum[row] = matrixAPolinom[row, col];

}

int currentResult = 0;

for (int currentRow = 0; currentRow < transportMatrixAPolinom.GetLength(0); currentRow++)

{

for (int currentCol = 0; currentCol < transportMatrixAPolinom.GetLength(1); currentCol++)

{

currentResult += currentColum[currentCol] \* transportMatrixAPolinom[currentRow, currentCol];

}

resultOfMultiplication.Add(currentResult);

currentResult = 0;

}

}

return resultOfMultiplication;

}

static void FullMatrixFromList(List<int> results, int[,] resultMatrix)

{

int currentIndex = 0;

for(int row = 0; row < resultMatrix.GetLength(0); row++)

{

for (int col = 0; col <resultMatrix.GetLength(1); col++)

{

resultMatrix[row, col] = results[currentIndex];

++currentIndex;

}

}

}

static int GetDeterminantAtPolinomOne(int[,] multiplcateMatrixPolinomOne)

{

int size = multiplcateMatrixPolinomOne.GetLength(1);

int firstDiagonal = 1;

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

{

firstDiagonal \*= multiplcateMatrixPolinomOne[cur, cur];

}

int secondDiagonal = 1;

int currentRow = 0;

int currentCol = size - 1;

for (int cur = size - 1; cur >= 0; cur--)

{

secondDiagonal \*= multiplcateMatrixPolinomOne[currentRow, currentCol];

++currentRow;

--currentCol;

}

int determinant = firstDiagonal - secondDiagonal;

return determinant;

}

static bool CheckIsDeterminantDiffrentOfZero(int determinant)

{

bool isZero = false;

if (determinant == 0)

{

isZero = true;

}

return isZero;

}

static int RuleOfTriangle(int[,] matrix)

{

int determinant = (matrix[0, 0] \* matrix[1, 1] \* matrix[2, 2]) +

(matrix[0, 1] \* matrix[1, 2] \* matrix[2, 0])

+ (matrix[0, 2] \* matrix[1, 0] \* matrix[2, 1]) -

(matrix[0, 2] \* matrix[1, 1] \* matrix[2, 0]) -

(matrix[0, 0] \* matrix[1, 2] \* matrix[2, 1]) -

(matrix[0, 1] \* matrix[1, 0] \* matrix[2, 2]);

return determinant;

}

static double[,]AlgorithmFindReverseMatrixOfPolinmOne(int[,] multiplcateMatrixPolinomOne, int determinantOfPolinomOne)

{

double[,] reverseMatirx = new double[multiplcateMatrixPolinomOne.GetLength(0),

multiplcateMatrixPolinomOne.GetLength(1)];

int swamp = multiplcateMatrixPolinomOne[0,0];

multiplcateMatrixPolinomOne[0, 0] = multiplcateMatrixPolinomOne[1, 1];

multiplcateMatrixPolinomOne[1, 1] = swamp;

multiplcateMatrixPolinomOne[0, 1] \*= -1;

multiplcateMatrixPolinomOne[1, 0] \*= -1;

double valueOfDetA = (1.00 / determinantOfPolinomOne);

for(int row = 0; row < multiplcateMatrixPolinomOne.GetLength(0); row++)

{

for (int col = 0; col < multiplcateMatrixPolinomOne.GetLength(1); col++)

{

reverseMatirx[row, col] =(multiplcateMatrixPolinomOne[row, col] \* valueOfDetA);

}

}

return reverseMatirx;

}

static List<int>CalculateCurrentResult(int[,] multiplicatePolinomTwo,int currentRow,int currentCol)

{

List<int> valuesOfSquareMatrix = new List<int>();

for(int row= 0; row < multiplicatePolinomTwo.GetLength(1); row++)

{

for (int col = 0; col < multiplicatePolinomTwo.GetLength(0); col++)

{

if(currentRow==row ||col==currentCol)

{

continue;

}

valuesOfSquareMatrix.Add(multiplicatePolinomTwo[row, col]);

}

}

return valuesOfSquareMatrix;

}

static List<int> AlgorithmFindReverseMatrixOfPolinmTwo(int[,] multiplicatePolinomTwo)

{

int[,] currentSquareMatrix = new int[2, 2];

List<int> results = new List<int>();

for(int row = 0; row < multiplicatePolinomTwo.GetLength(0); row++)

{

for(int col = 0; col < multiplicatePolinomTwo.GetLength(1); col++)

{

List<int> valuesOfCurrentRow = CalculateCurrentResult(multiplicatePolinomTwo, row,col);

FullMatrixFromList(valuesOfCurrentRow, currentSquareMatrix);

int sumOfColAndRow = (col + 1)+(row + 1);

int determinantOfSquare = GetDeterminantAtPolinomOne(currentSquareMatrix);

int currentValue = (int)Math.Pow(-1, sumOfColAndRow) \* (determinantOfSquare);

results.Add(currentValue);

}

}

return results;

}

static double[,] CalcualateFinaleReverseMatrix(int[,] reverseMatrixOfPolinomTwo,

int determinantOfPolinomTwo)

{

double[,] oppisteMatrix = new double[3, 3];

for (int row = 0; row <oppisteMatrix.GetLength(0); row++)

{

for (int col = 0; col < oppisteMatrix.GetLength(1); col++)

{

oppisteMatrix[row, col] =(1.00 / determinantOfPolinomTwo) \*

reverseMatrixOfPolinomTwo[row, col];

}

}

return oppisteMatrix;

}

static double[,] multiplicationTransoprtMatrixOnMatrixB(int[,] transportMatrixA, double[] argumentsY)

{

double[,] resultMatrix = new double[transportMatrixA.GetLength(0), 1];

for (int row = 0; row < transportMatrixA.GetLength(0); row++)

{

double currentResult = 0;

for (int col = 0; col < argumentsY.Length; col++)

{

currentResult += (transportMatrixA[row, col] \* argumentsY[col]);

}

resultMatrix[row, 0] = currentResult;

}

return resultMatrix;

}

static double[,] CalculateCoefficients(double[,]reverseMatrix, double[,] multplicateMatrixY)

{

double[,] coefficients = new double[multplicateMatrixY.GetLength(0),

multplicateMatrixY.GetLength(1)];

for (int row = 0; row < reverseMatrix.GetLength(0); row++)

{

double currentResult = 0;

for (int col = 0; col < reverseMatrix.GetLength(1); col++)

{

currentResult += reverseMatrix[row, col] \* multplicateMatrixY[col, 0];

}

coefficients[row, 0] = currentResult;

}

return coefficients;

}

static double[]GetGradesOfPolynomOne(double[,] coefficientsOfPolynomOne, int[] argumentsX)

{

double[] grades = new double[argumentsX.Length];

for(int cur = 0; cur < argumentsX.Length; cur++)

{

grades[cur] = (coefficientsOfPolynomOne[0, 0] \* argumentsX[cur]) +

coefficientsOfPolynomOne[1, 0];

}

return grades;

}

static double[] GetGradesOfPolynomTwo(double[,] coefficientsOfPolynomTwo, int[] argumentsX)

{

double[] grades = new double[argumentsX.Length];

for (int cur = 0; cur < argumentsX.Length; cur++)

{

grades[cur] = (Math.Pow(argumentsX[cur], 2) \* coefficientsOfPolynomTwo[0, 0])

+ (argumentsX[cur] \* coefficientsOfPolynomTwo[1, 0])

+ coefficientsOfPolynomTwo[2, 0];

}

return grades;

}

static double CalculateQuadraticError(double[] grades, double[] argumentsY)

{

double[] calculateGrades = new double[argumentsY.Length];

for (int cur = 0; cur < argumentsY.Length; cur++)

{

calculateGrades[cur] = Math.Pow((argumentsY[cur] - grades[cur]),2);

}

double sum = calculateGrades.Sum();

double quadraticError = Math.Sqrt(sum);

return quadraticError;

}

}

}