БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Приходько Егор Андреевич

ОТЧЕТ ПО МЕТОДАМ ВЫЧИСЛЕНИЙ

студента 2 курса 13 группы

Лабораторная работа №3

Преподаватель

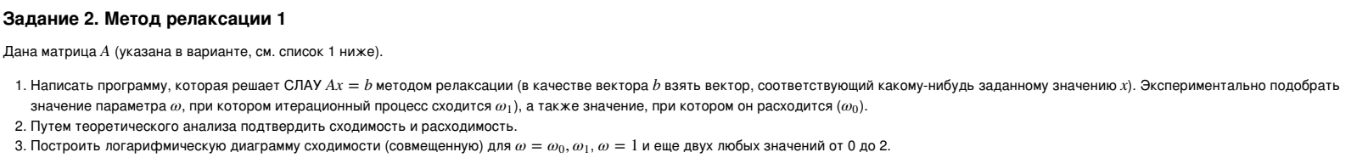
Бондарь И.В.

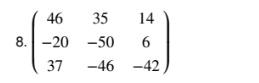
Минск 2020

**Вариант 8**: Задание 2 (8) + Задание 5 (метод 2, задача 1)

**Задание 2 (8)**

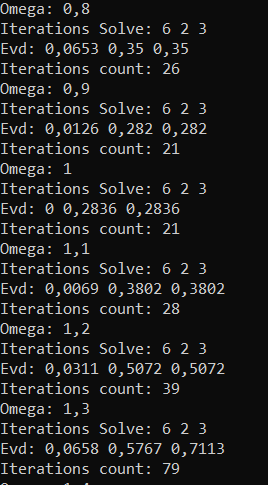
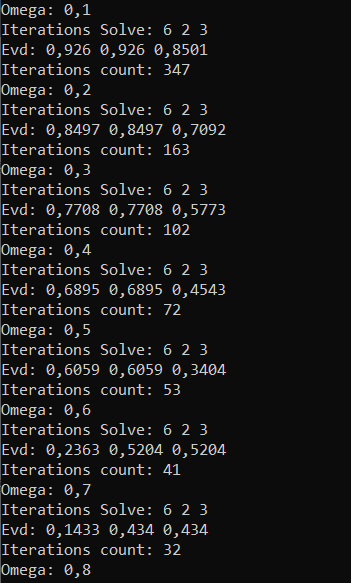
**Постановка задачи.**

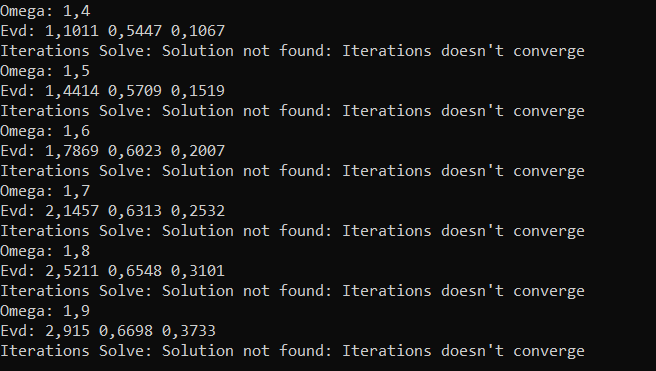




1. Протестируем программу для матрицы 8, в качестве вектора x возьмем вектор (6, 2, 3), вектор b при таком x равен (388, -202, 4).

В качестве омеги будем брать прогонку от 0.1 до 1.9.  
Вывод программы:





***Легенда:*** Omega - Омега, Evd – модули собственных значений матрицы,

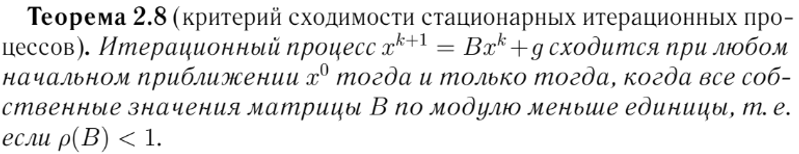
Iterations count – количество итераций, Iterations solve – решение методом релаксации.

2. Разберем вывод программы.

Как видим, метод сходится на промежутке омега э[0.1 , 1.3].

На сходимость перед метод происходит проверка следующие следующим образом:

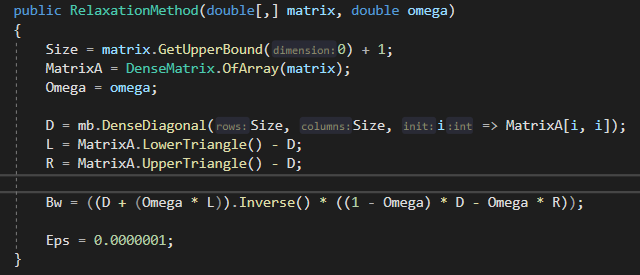
Теорема



Где B

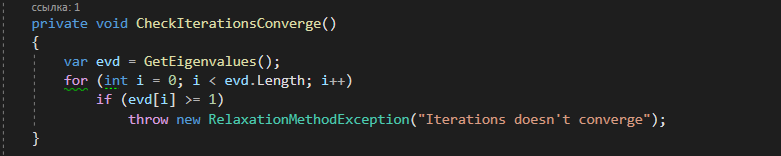


Инициализация метода релаксации



Проверка





Прим.: Для линейных операций над матрицами и поиска собственных значение использовал библиотеку **Microsoft MathNet.Numerics.LinearAlgebra.Double;**

В качестве значения омегаo - 1.4

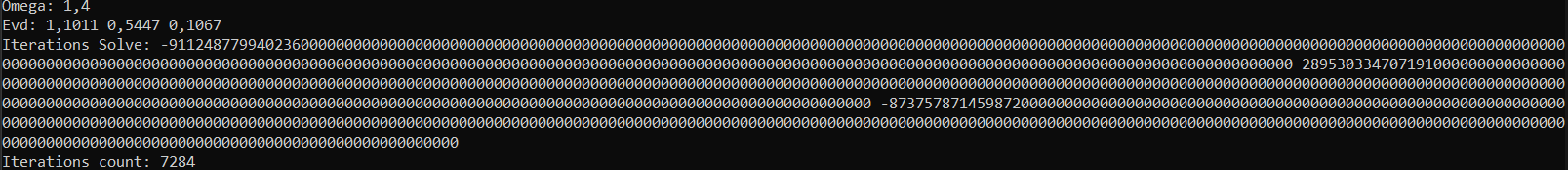
В качестве значения омега1 - 0.5

Для доказательства сходимости/расходимости из вывода программы возьмем модули собственных значений.  
Как видим для омегаo спектральный радиус равен 1.101

Как видим для омега1 спектральный радиус равен 0.606

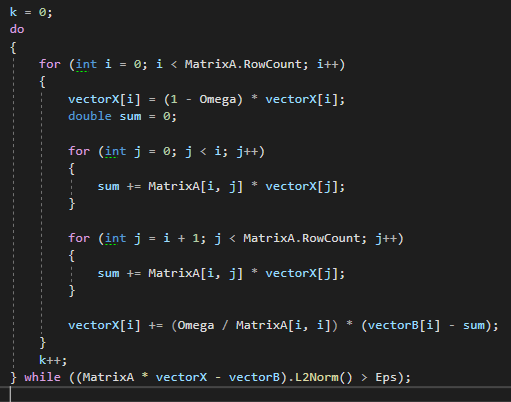
Для наглядности уберем проверку для омегаo.

Вывод программы:



Как видим, значения очень сильно расходятся(иногда, даже, до переполнение типа double)

**Сам код метода релоксации.**



Прим.: Для поиска евклидовой нормы вектора использую библиотеку

**Microsoft MathNet.Numerics.LinearAlgebra.Double;**

3. Построим график.

Возьмем значения:

**Омега0 - 0.5**

**Омега1 - 1.4**

**Омега2 -** **1.0**

**Омега3 - 0.1**

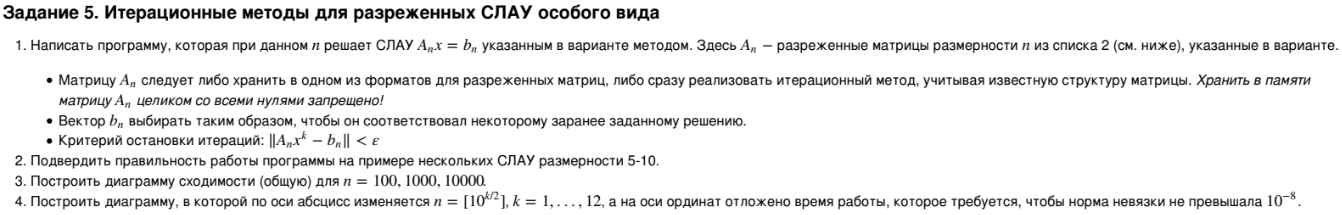
**Омега4 - 0.8**

График функции

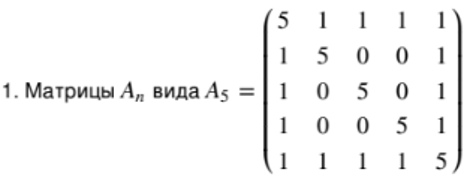
**log10(eps) = F(k)**.

**Задание 5 (метод 2, задача 1)**

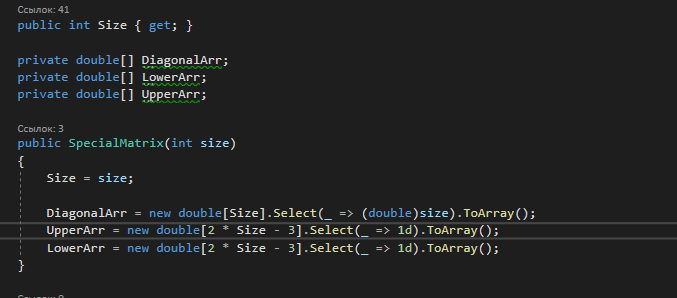
**Постановка задачи.**



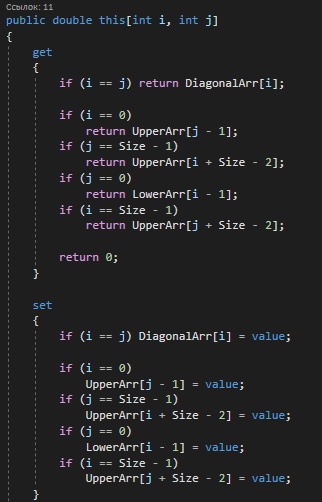




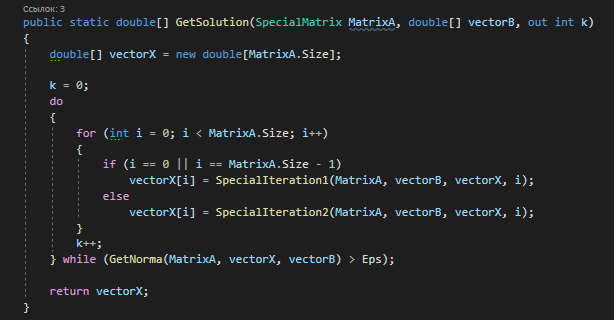
1. Реализация специальной матрицы и специального алгоритма Гаусса - Зейделя.

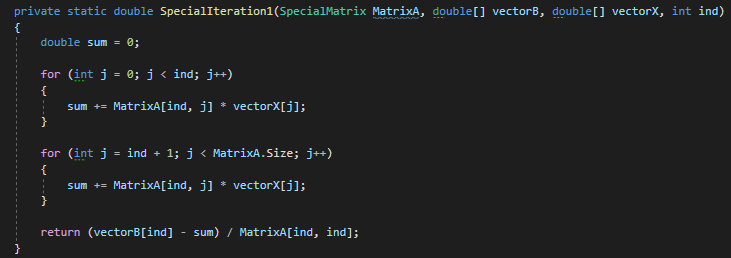
Создал специальный класс для матрицы своего варианта  
 

Для наглядности, удобства и лучшего понимание в коде, перегрузил индексаторы своего класса:

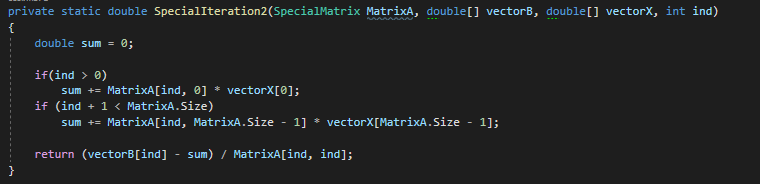


Сама реализация алгоритма Гаусса – Зейделя:

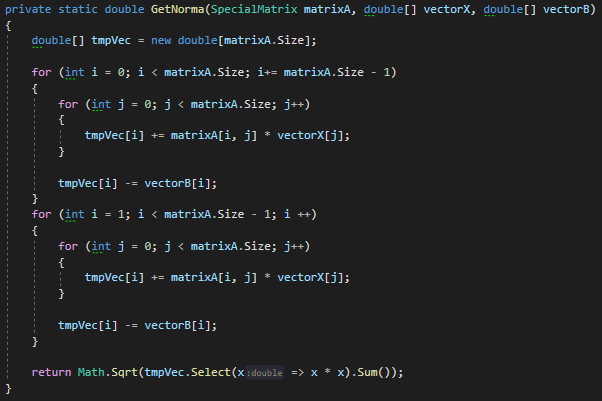


Где SpecialIteration1 – это итерация для первой и последней строки матрицы.  


А SpecialIteration2 – это итерация для остальных строк матрицы.

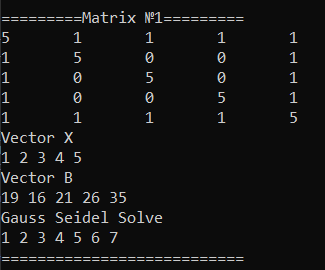


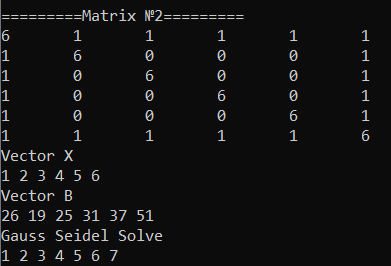
Поскольку в остальных строках большинство элементов – это нули, сделаем следующую оптимизацию: будем учитывать только последний и первый элементы в столбцах матрицы.

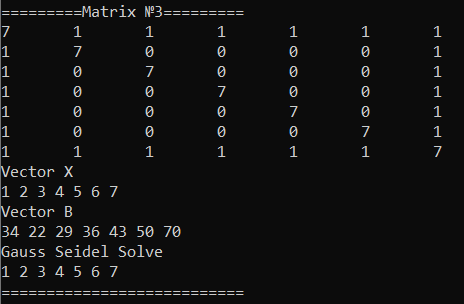
Так же оптимизированный код для поиска нормы матрицы. 

2. Тестирование программы на матрицах размероности 5, 6, 7.

Вывод программы:







Как видим все работает корректно.

3. Построим график сходимости.

**N = 100**

**N = 1000**

**N = 10000**

График функции

**eps = F(k)**.

Как видим по графкику, чем больше размерность – тем быстрее сходимость.

Думаю это связано с тем, что метод Гаусса – Зейделя, на текущей итерации k + 1, ипользует уже итерированные значение k + 1 и, следовательно, на большей размерности метод использует эти значение чаще.

4. Построим график зависимости времени от размерности.

**eps = F(k)**.

|  |  |
| --- | --- |
| **Размер** | **Время**  **мс.** |
| **3** | **0** |
| **10** | **0** |
| **32** | **0** |
| **100** | **0** |
| **316** | **3** |
| **1000** | **9** |
| **3162** | **102** |
| **10000** | **914** |
| **31623** | **8312** |
| **100000** | **84837** |
| **316228** | **509045** |
| **1000000** | **1527051** |

**Program.cs**

using System;

using System.Diagnostics;

using System.IO;

using System.Linq;

namespace Lab3

{

class Program

{

public static void PrintMatrix(double[,] matrix)

{

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

{

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

{

Console.Write(matrix[i, j].ToString("0.###\t"));

}

Console.WriteLine();

}

}

public static void PrintMatrix(SpecialMatrix matrix)

{

for (int i = 0; i < matrix.Size; i++)

{

for (int j = 0; j < matrix.Size; j++)

{

Console.Write(matrix[i, j].ToString("0.###\t"));

}

Console.WriteLine();

}

}

public static void PrintVector(double[] vector)

{

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

{

Console.Write(vector[i].ToString("0.#### "));

}

Console.WriteLine();

}

private static void TimeTest()

{

using (StreamWriter sw = new StreamWriter("Time Statistic.txt"))

{

for (int k = 1; k <= 12; k++)

{

int size = (int)Math.Pow(10, k / 2d);

double[] b = new double[size].Select(\_ => 1.0).ToArray();

SpecialMatrix matrixA = new SpecialMatrix(size);

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

SpecialGaussSeidel.GetSolution(matrixA, b, out \_);

stopwatch.Stop();

Console.WriteLine($"Time for a matrix with size {size}: {stopwatch.ElapsedMilliseconds}");

sw.WriteLine(stopwatch.ElapsedMilliseconds);

}

}

}

static void Main(string[] args)

{

//Task 1

#region Task1RunTestCode

Console.WriteLine("==============================Task 1==============================");

double[,] matrix = new double[,]

{

{46, 35, 14 },

{-20, -50, 6 },

{37, -46, -42 },

};

double[] vectorX = new double[] { 6, 2, 3 };

for (double i = 0.1; i < 2; i += 0.1)

{

var Method = new RelaxationMethod(matrix, i);

try

{

Console.WriteLine("Omega: " + i);

Console.Write("Evd: ");

PrintVector(Method.GetEigenvalues());

Console.Write("Iterations Solve: ");

PrintVector(Method.GetSolution(Method.GetVectorB(vectorX), out int k));

Console.WriteLine("Iterations count: " + k);

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

#endregion

//Task 2

#region Task2RunTestCode

Console.WriteLine("\n\n\n==============================Task 2==============================");

SpecialMatrix specialMatrix1 = new SpecialMatrix(5);

double[] vectorX1 = new double[] { 1, 2, 3, 4, 5 };

double[] vectorB1 = SpecialGaussSeidel.GetVectorB(specialMatrix1, vectorX1);

SpecialMatrix specialMatrix2 = new SpecialMatrix(6);

double[] vectorX2 = new double[] { 1, 2, 3, 4, 5, 6 };

double[] vectorB2 = SpecialGaussSeidel.GetVectorB(specialMatrix2, vectorX2);

SpecialMatrix specialMatrix3 = new SpecialMatrix(7);

double[] vectorX3 = new double[] { 1, 2, 3, 4, 5, 6, 7 };

double[] vectorB3 = SpecialGaussSeidel.GetVectorB(specialMatrix3, vectorX3);

Console.WriteLine("\n=========Matrix №1=========");

PrintMatrix(specialMatrix1);

Console.WriteLine("Vector X");

PrintVector(vectorX1);

Console.WriteLine("Vector B");

PrintVector(vectorB1);

Console.WriteLine("Gauss Seidel Solve");

PrintVector(SpecialGaussSeidel.GetSolution(specialMatrix1, vectorB1, out \_));

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

Console.WriteLine("\n=========Matrix №2=========");

PrintMatrix(specialMatrix2);

Console.WriteLine("Vector X");

PrintVector(vectorX2);

Console.WriteLine("Vector B");

PrintVector(vectorB2);

Console.WriteLine("Gauss Seidel Solve");

PrintVector(SpecialGaussSeidel.GetSolution(specialMatrix2, vectorB2, out \_));

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

Console.WriteLine("\n=========Matrix №3=========");

PrintMatrix(specialMatrix3);

Console.WriteLine("Vector X");

PrintVector(vectorX3);

Console.WriteLine("Vector B");

PrintVector(vectorB3);

Console.WriteLine("Gauss Seidel Solve");

PrintVector(SpecialGaussSeidel.GetSolution(specialMatrix3, vectorB3, out \_));

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

TimeTest();

#endregion

Console.ReadKey();

}

}

}

**RelaxationMethod.cs**

using System;

using System.Linq;

using MathNet.Numerics.LinearAlgebra;

using MathNet.Numerics.LinearAlgebra.Double;

namespace Lab3

{

class RelaxationMethodException : Exception

{

public RelaxationMethodException(string msg) : base("Solution not found: " + msg) { }

}

class RelaxationMethod

{

private static MatrixBuilder<double> mb = Matrix<double>.Build;

private Matrix<double> L;

private Matrix<double> D;

private Matrix<double> R;

public Matrix<double> MatrixA { get; }

public double Omega { get; }

public int Size { get; }

public Matrix<double> Bw { get; }

public double Eps { get; set; }

public RelaxationMethod(double[,] matrix, double omega)

{

Size = matrix.GetUpperBound(0) + 1;

MatrixA = DenseMatrix.OfArray(matrix);

Omega = omega;

D = mb.DenseDiagonal(Size, Size, i => MatrixA[i, i]);

L = MatrixA.LowerTriangle() - D;

R = MatrixA.UpperTriangle() - D;

Bw = ((D + (Omega \* L)).Inverse() \* ((1 - Omega) \* D - Omega \* R));

Eps = 0.0000001;

}

public double[] GetEigenvalues() => Bw.Evd().EigenValues.Select(item => item.Magnitude).ToArray();

public double[] GetSolution(double[] vec, out int k)

{

CheckIterationsConverge();

Vector<double> vectorB = DenseVector.OfArray(vec);

Vector<double> vectorX = DenseVector.OfArray(vec);

k = 0;

do

{

for (int i = 0; i < MatrixA.RowCount; i++)

{

vectorX[i] = (1 - Omega) \* vectorX[i];

double sum = 0;

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

{

sum += MatrixA[i, j] \* vectorX[j];

}

for (int j = i + 1; j < MatrixA.RowCount; j++)

{

sum += MatrixA[i, j] \* vectorX[j];

}

vectorX[i] += (Omega / MatrixA[i, i]) \* (vectorB[i] - sum);

}

k++;

} while ((MatrixA \* vectorX - vectorB).L2Norm() > Eps);

return vectorX.ToArray();

}

private void CheckIterationsConverge()

{

var evd = GetEigenvalues();

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

if (evd[i] >= 1)

throw new RelaxationMethodException("Iterations doesn't converge");

}

public double[] GetVectorB(double[] vectorX) => (MatrixA \* DenseVector.OfArray(vectorX)).ToArray();

}

}

**SpecialMatrix.cs**

using System.Linq;

namespace Lab3

{

public class SpecialMatrix

{

public int Size { get; }

private double[] DiagonalArr;

private double[] LowerArr;

private double[] UpperArr;

public SpecialMatrix(int size)

{

Size = size;

DiagonalArr = new double[Size].Select(\_ => (double)size).ToArray();

UpperArr = new double[2 \* Size - 3].Select(\_ => 1d).ToArray();

LowerArr = new double[2 \* Size - 3].Select(\_ => 1d).ToArray();

}

public SpecialMatrix(double[,] matrix)

{

Size = matrix.GetUpperBound(0) + 1;

DiagonalArr = new double[Size].Select((\_, i) => matrix[i, i]).ToArray();

UpperArr = new double[2 \* Size - 3];

LowerArr = new double[2 \* Size - 3];

int ind = 0;

for (int j = 1; j < Size; j++, ind++)

{

UpperArr[ind] = matrix[0, j];

LowerArr[ind] = matrix[j, 0];

}

for (int j = 1; j < Size - 1; j++, ind++)

{

UpperArr[ind] = matrix[j, Size - 1];

LowerArr[ind] = matrix[Size - 1, j];

}

}

public double this[int i, int j]

{

get

{

if (i == j) return DiagonalArr[i];

if (i == 0)

return UpperArr[j - 1];

if (j == Size - 1)

return UpperArr[i + Size - 2];

if (j == 0)

return LowerArr[i - 1];

if (i == Size - 1)

return UpperArr[j + Size - 2];

return 0;

}

set

{

if (i == j) DiagonalArr[i] = value;

if (i == 0)

UpperArr[j - 1] = value;

if (j == Size - 1)

UpperArr[i + Size - 2] = value;

if (j == 0)

LowerArr[i - 1] = value;

if (i == Size - 1)

UpperArr[j + Size - 2] = value;

}

}

}

}

**SpecialGaussSeidel.cs**

using System;

using System.Linq;

namespace Lab3

{

public static class SpecialGaussSeidel

{

public static double Eps = 0.0000001;

private static double GetNorma(SpecialMatrix matrixA, double[] vectorX, double[] vectorB)

{

double[] tmpVec = new double[matrixA.Size];

for (int i = 0; i < matrixA.Size; i+= matrixA.Size - 1)

{

for (int j = 0; j < matrixA.Size; j++)

{

tmpVec[i] += matrixA[i, j] \* vectorX[j];

}

tmpVec[i] -= vectorB[i];

}

for (int i = 1; i < matrixA.Size - 1; i ++)

{

for (int j = 0; j < matrixA.Size; j++)

{

tmpVec[i] += matrixA[i, j] \* vectorX[j];

}

tmpVec[i] -= vectorB[i];

}

return Math.Sqrt(tmpVec.Select(x => x \* x).Sum());

}

private static double SpecialIteration1(SpecialMatrix MatrixA, double[] vectorB, double[] vectorX, int ind)

{

double sum = 0;

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

{

sum += MatrixA[ind, j] \* vectorX[j];

}

for (int j = ind + 1; j < MatrixA.Size; j++)

{

sum += MatrixA[ind, j] \* vectorX[j];

}

return (vectorB[ind] - sum) / MatrixA[ind, ind];

}

private static double SpecialIteration2(SpecialMatrix MatrixA, double[] vectorB, double[] vectorX, int ind)

{

double sum = 0;

if(ind > 0)

sum += MatrixA[ind, 0] \* vectorX[0];

if (ind + 1 < MatrixA.Size)

sum += MatrixA[ind, MatrixA.Size - 1] \* vectorX[MatrixA.Size - 1];

return (vectorB[ind] - sum) / MatrixA[ind, ind];

}

public static double[] GetSolution(SpecialMatrix MatrixA, double[] vectorB, out int k)

{

double[] vectorX = new double[MatrixA.Size];

k = 0;

do

{

for (int i = 0; i < MatrixA.Size; i++)

{

if (i == 0 || i == MatrixA.Size - 1)

vectorX[i] = SpecialIteration1(MatrixA, vectorB, vectorX, i);

else

vectorX[i] = SpecialIteration2(MatrixA, vectorB, vectorX, i);

}

k++;

} while (GetNorma(MatrixA, vectorX, vectorB) > Eps);

return vectorX;

}

public static double[] GetVectorB(SpecialMatrix matrixA, double[] vectorX)

{

double[] result = new double[matrixA.Size];

for (int i = 0; i < matrixA.Size; i += matrixA.Size - 1)

{

for (int j = 0; j < matrixA.Size; j++)

{

result[i] += matrixA[i, j] \* vectorX[j];

}

}

for (int i = 1; i < matrixA.Size - 1; i++)

{

for (int j = 0; j < matrixA.Size; j++)

{

result[i] += matrixA[i, j] \* vectorX[j];

}

}

return result;

}

}

}