1. **Постановка задачи**

Эллиптическая краевая задача для функции определяется дифференциальным уравнением



Заданным в некоторой области Ω с границей и краевыми условиями



1. **Вариационная постановка Галеркина**

Введём гильбертово пространство .

Запишем невязку в виде:



и потребуем, чтобы она была ортогональна, в смысле скалярного произведения пространства H0 , некоторому пространству пробных функций, т.е.:

 ,

где .

Перепишем уравнение в виде:



Для понижения требования к гладкости функции решения преобразуем полученное выражение, при помощи формулы Грина



т.к.то значение  на границе S1 равно 0, поэтому  следует исключить из уравнения.

Получаем вариационное уравнение вида:



1. **Построение дискретного аналога в МКЭ**

Рассмотрим построение для эллиптической задачи.

В пространствах  и  выделим соответственно конечномерные подпространства  и , которые их аппроксимируют. Определим подпространства как линейные пространства, натянутые на базисные функции . Заменим функцию  аппроксимирующей ее функцией , а функцию  - функцией .

Получим аппроксимацию уравнения Галеркина:



Поскольку любая функция  может быть представлена в виде линейной комбинации

,

вариационное уравнение эквивалентно следующей системе уравнений:

, 

где  - количество базисных функций, образующих пространства  и .

 может быть представлено в виде:



Получим СЛАУ для компонент  вектора весов :

, 

СЛАУ имеет размерность n, из которых  компонент вектора весов

 фиксированы и определены из первого краевого условия  для  , т.е.

СЛАУ может быть записана в матричном виде:

, где:





Компоненты матрицы А имеют следующие определения:

 -матрица жесткости

Для эффективности решения задачи, будем использовать метод поэлементной

сборки матрицы А. Сначала будут вычисляться вклады от каждого конечного элементав соответствующие компоненты глобальной матрицы А. Вклады этого элемента можно рассматривать как матрицы жесткости  и матрицы массы:

 -матрица жесткости

1. **Выбор базиса**

Так как вклады будем складывать по элементам функции будем выбирать с конечным носителем. Выберем их так, чтобы каждая принимала значение 1 только в своем узле и 0 во всех других.

Рассмотрим невырожденный тетраэдр имеющий вершины

Введем на нем линейные функции вида:



Такие, что каждая из функций  равна единице в вершине и 0 во всех остальных точках.



Откуда следует



Что позволяет однозначно определить все базисные функции.

Представим их как произведение одномерных квадратичных функций. Для определения квадратичной функции нужно 3 точки. Значит всего на элементе будет 9 точек.

1. **Сборка локальных матриц**

Локальные матрицы будем вычислять по формуле







Для вычисления правой части посмотрим на правую часть аппроксимации уравнения Галеркина и разложим функцию в правой части по нашему базису и тогда получится



Отсюда получим матрицу 

Тогда локальный вектор правой части можно вычислить как

где вектор составленный из значений правой части в узлах элемента.

1. Сборка глобальной матрицы

Матрица хранится в разряженном формате. Сначала формируются массивы ig, jg. Размерность ig равна количеству узлов. Для этого формируется список смежности узлов с учетом симметричности портрета. Для каждого узла заносятся номера узлов соединенных с ним ребром, при этом номера сортируются по возрастанию. Далее проходим по всем элементам списка считаем общее количество смежный узлов и в i-тую компоненту положим количество узлов + количество в предыдущей. Jg имеет размерность равную общему количеству узлов в ig. Теперь перепишем последовательно все вершины из списка (для каждой вершины все смежные).

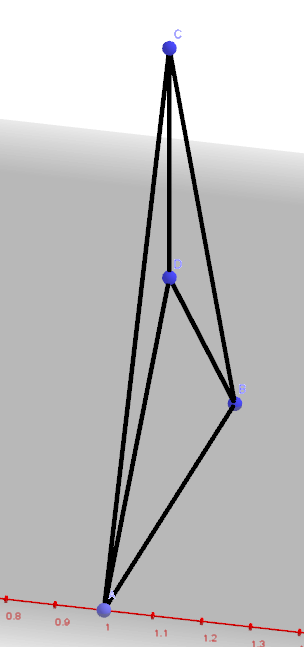
Далее собираем глобальную матрицу с помощью глобальной и локальной нумерации. Решаем систему методом ЛОС с не полной LU факторизацией.

1. Тесты

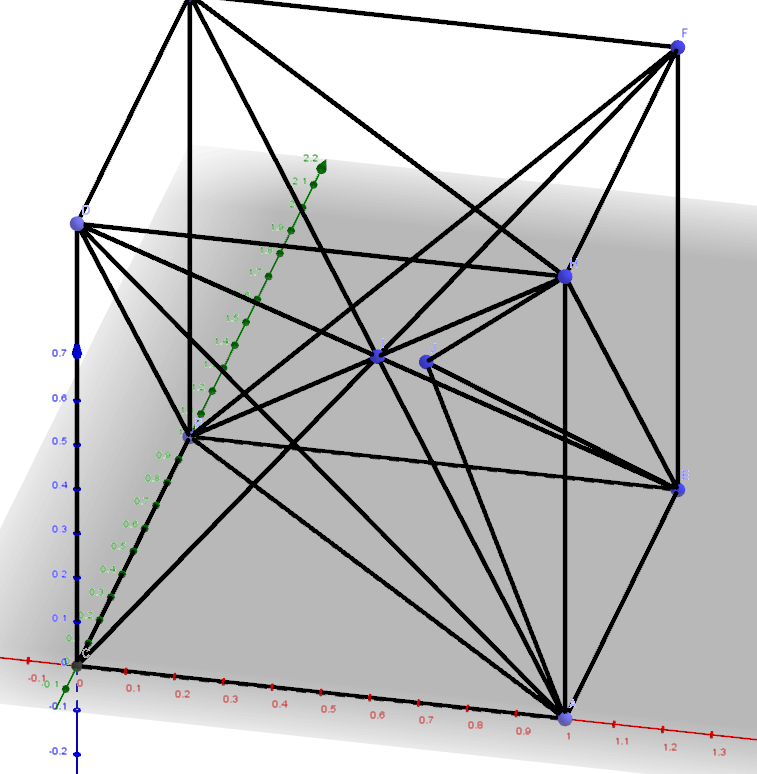
Для проверки корректности сетки сначала определим критерии корректности. Корректная сетка обладает следующими характеристиками:

* Отсутствие тетраэдров нулевого объема (плоских, точечных)

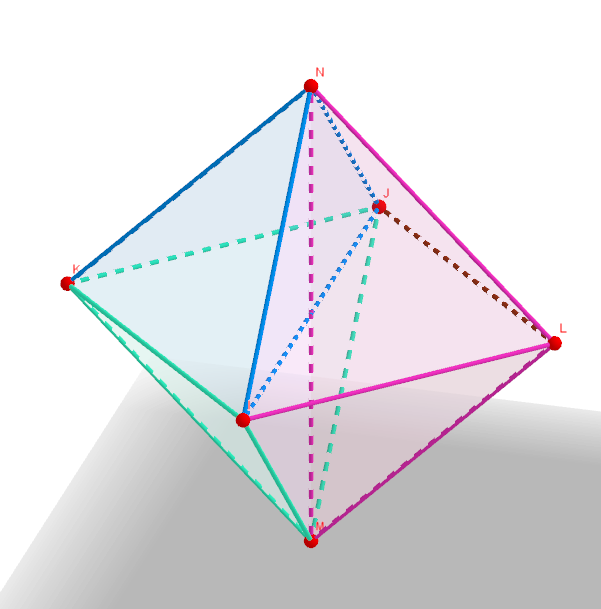
|  |  |  |
| --- | --- | --- |
| x | y | z |
| 1 | 0 | 0 |
| 1 | 1 | 0 |
| 1 | 0.5 | 1 |
| 1 | 0.5 | 0.5 |



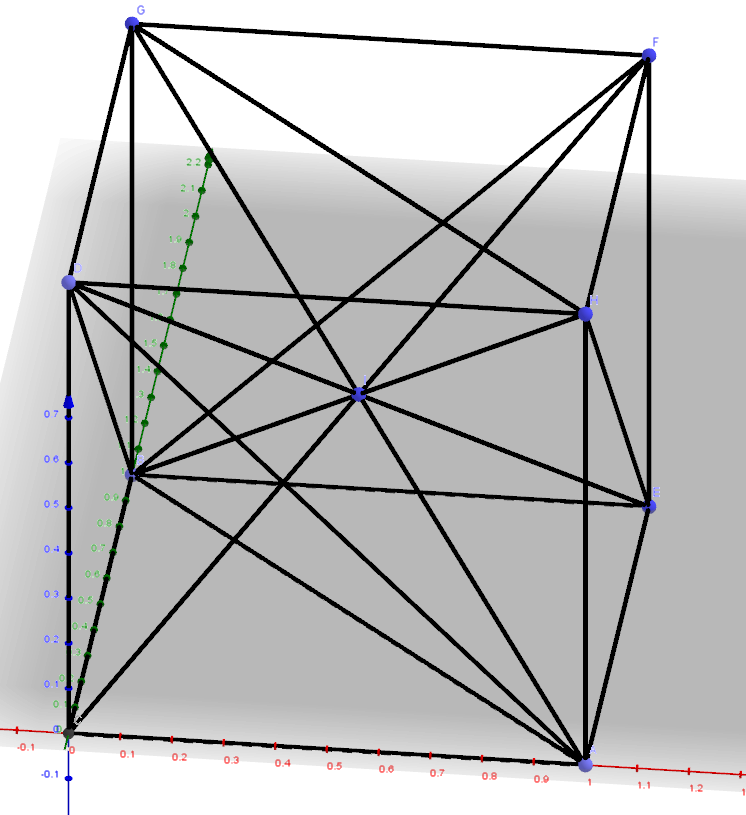
* Отсутствие тетраэдров с вершинами в точках, не лежащих в узлах сетки



* Отсутствие перекрещивания ребер



Пример корректной сетки



Координаты вершин Список тетраэдров

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8 | (0,0,0)  (1,0,0)  (0,1,0)  (1,1,0)  (0,0,1)  (1,0,1)  (0,1,1)  (1,1,1)  (0.5,0.5,0.5) |

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11 | (0,1,4,8,0)  (1,4,5,8,0)  (1,3,5,8,0)  (3,5,7,8,0)  (2,3,7,8,0)  (2,6,7,8,0)  (0,2,4,8,0)  (2,4,6,8,0)  (0,1,2,8,0)  (1,2,3,8,0)  (4,5,6,8,0)  (5,6,7,8,0) |

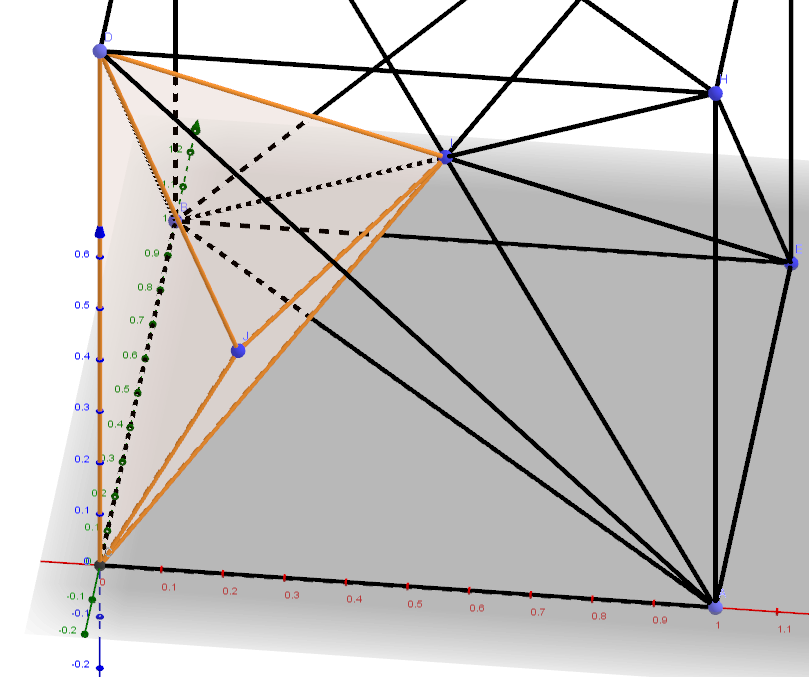
Для 

Значение весов полученное в ходе решения

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8 | 0  1  1  2  1  2  2  3  1,25 |

Некорректная сетка с тетраэдром нулевого объема

Для 

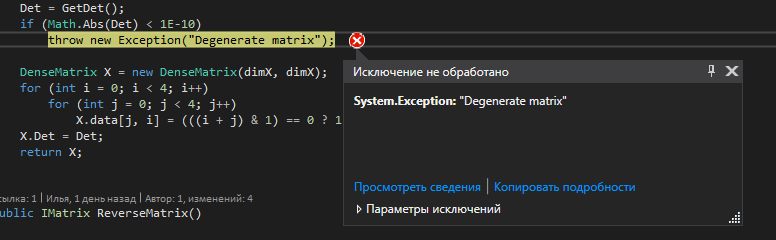


Координаты вершин Список тетраэдров

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9 | (0,0,0)  (1,0,0)  (0,1,0)  (1,1,0)  (0,0,1)  (1,0,1)  (0,1,1)  (1,1,1)  (0.5,0.5,0.5)  (0.2,0.2,0.3) |

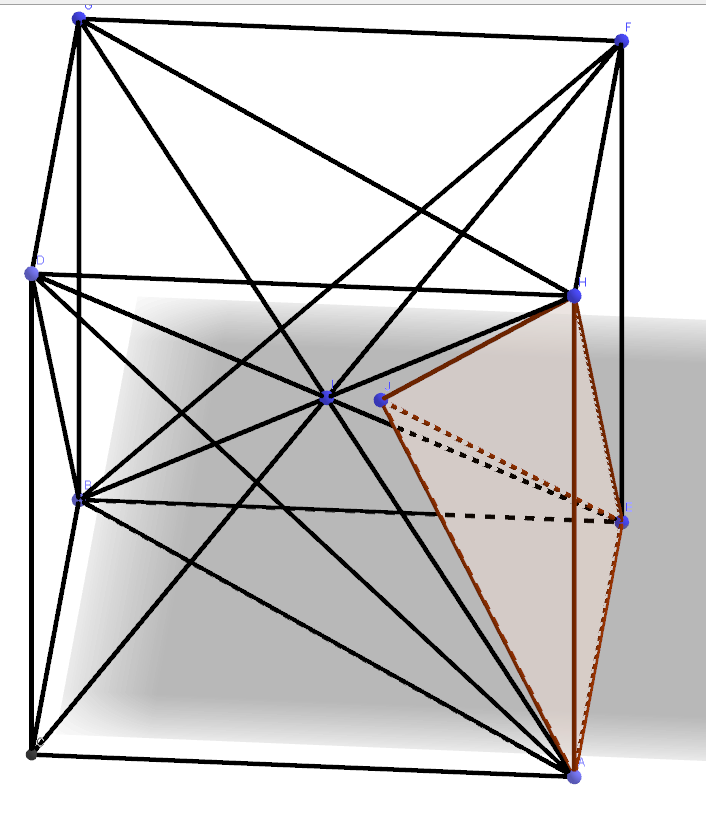
|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12 | (0,1,4,8,0)  (1,4,5,8,0)  (1,3,5,8,0)  (3,5,7,8,0)  (2,3,7,8,0)  (2,6,7,8,0)  (0,2,4,8,0)  (2,4,6,8,0)  (0,1,2,8,0)  (1,2,3,8,0)  (4,5,6,8,0)  (5,6,7,8,0)  (0,4,8,9,0) |

Программа не может посчитать и выбрасывает исключение при подсчете локальной матрицы из-за необходимости делить на определитель.



Некорректная сетка с вершинами тетраэдра не в узлах сетки

Для 



Координаты вершин Список тетраэдров

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9 | (0,0,0)  (1,0,0)  (0,1,0)  (1,1,0)  (0,0,1)  (1,0,1)  (0,1,1)  (1,1,1)  (0.5,0.5,0.5)  (0.6,0.5,0.5) |

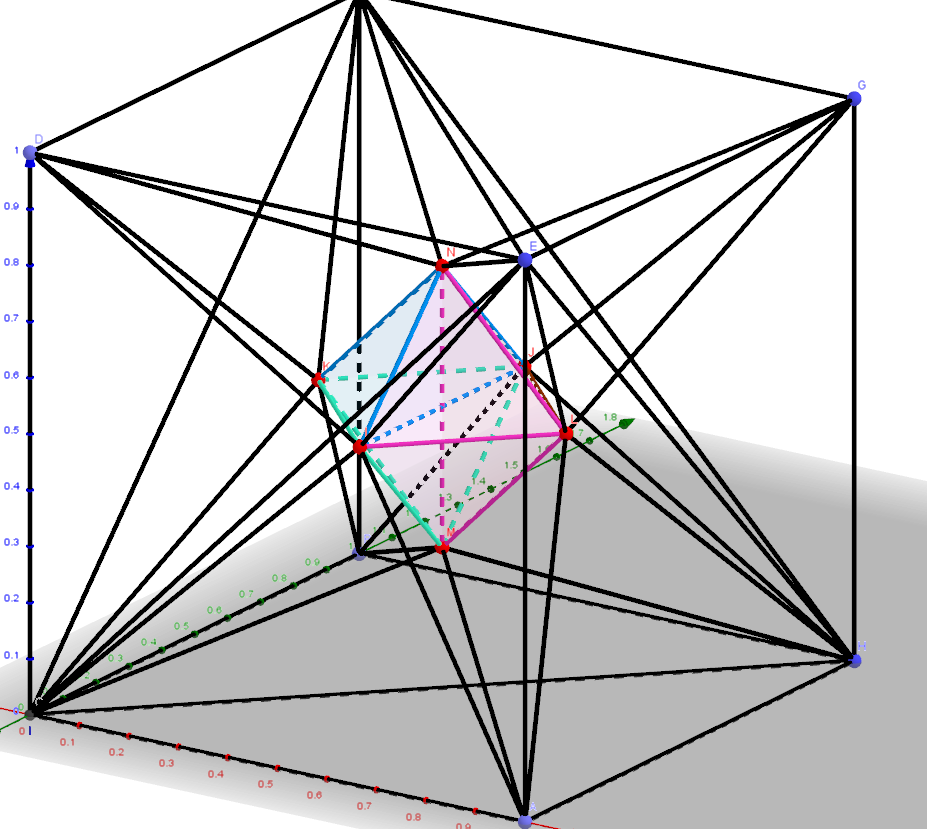
|  |  |
| --- | --- |
| 0  1  3  4  5  6  7  8  9  10  11  2 | (0,1,4,8,0)  (1,4,5,8,0)  (3,5,7,8,0)  (2,3,7,8,0)  (2,6,7,8,0)  (0,2,4,8,0)  (2,4,6,8,0)  (0,1,2,8,0)  (1,2,3,8,0)  (4,5,6,8,0)  (5,6,7,8,0)  (1,3,5,9,0) |

Значение весов полученное в ходе решения

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9 | -1.51358E-08  1.00000E+00  1.00000E+00  2.00000E+00  1.00000E+00  2.00000E+00  2.00000E+00  3.00000E+00  1.45455E+00  2.00000E+00 |

Некорректная сетка с перекрещенными ребрами

Для 



Координаты вершин Список тетраэдров

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13 | (0,0,0)  (1,0,0)  (0,1,0)  (1,1,0)  (0,0,1)  (1,0,1)  (0,1,1)  (1,1,1)  (0.5,0.5,0.25)  (0.25,0.5,0.5)  (0.5,0.25,0.5)  (0.75,0.5,0.5)  (0.5,0.75,0.5)  (0.5,0.5,0.75) |

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | (9,10,12,13,0)  (8,9,10,12,0)  (8,10,11,13,0)  (8,11,12,13,0)  (0,1,5,10,0)  (0,4,5,10,0)  (1,3,5,11,0)  (3,5,7,11,0)  (2,3,6,12,0)  (3,6,7,12,0)  (0,2,6,9,0)  (0,4,6,9,0)  (0,1,3,8,0)  (0,2,3,8,0)  (4,5,6,13,0)  (5,6,7,13,0)  (0,4,9,10,0)  (1,5,10,11,0)  (3,7,11,12,0)  (2,6,9,12,0)  (0,1,8,10,0)  (4,5,10,13,0)  (6,7,12,13,0)  (2,3,8,12,0) |

Значение весов полученное в ходе решения

|  |  |  |
| --- | --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13 | -1.18431E-08  1.00000E+00  1.00000E+00  2.00000E+00  1.00000E+00  2.00000E+00  2.00000E+00  3.00000E+00  1.15981E+00  1.14606E+00  1.19342E+00  1.85037E+00  1.78916E+00  1.86251E+00 | (0,0,0)  (1,0,0)  (0,1,0)  (1,1,0)  (0,0,1)  (1,0,1)  (0,1,1)  (1,1,1)  (0.5,0.5,0.25)  (0.25,0.5,0.5)  (0.5,0.25,0.5)  (0.75,0.5,0.5)  (0.5,0.75,0.5)  (0.5,0.5,0.75) |

1. Вывод

Этот способ проверки сетки позволяет точно отследить тетраэдры нулевого объема и заметить ухудшение решения при других некорректностях.

1. Текст программы

Program.cs

using NetCheckApp;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace NetCheckerFEM

{

public class Program

{

static void Main()

{

List<Vector3D> p = new List<Vector3D>(14)

{

new Vector3D(0,0,0),

new Vector3D(1,0,0),

new Vector3D(0,1,0),

new Vector3D(1,1,0),

new Vector3D(0,0,1),

new Vector3D(1,0,1),

new Vector3D(0,1,1),

new Vector3D(1,1,1),

new Vector3D(0.5,0.5,0.25),

new Vector3D(0.25,0.5,0.5),

new Vector3D(0.5,0.25,0.5),

new Vector3D(0.75,0.5,0.5),

new Vector3D(0.5,0.75,0.5),

new Vector3D(0.5,0.5,0.75)

};

List<Thetra> t = new List<Thetra>(24) {

new Thetra(9,10,12,13,0),

new Thetra(8,9,10,12,0),

new Thetra(8,10,11,13,0),

new Thetra(8,11,12,13,0),

new Thetra(0,1,5,10,0),

new Thetra(0,4,5,10,0),

new Thetra(1,3,5,11,0),

new Thetra(3,5,7,11,0),

new Thetra(2,3,6,12,0),

new Thetra(3,6,7,12,0),

new Thetra(0,2,6,9,0),

new Thetra(0,4,6,9,0),

new Thetra(0,1,3,8,0),

new Thetra(0,2,3,8,0),

new Thetra(4,5,6,13,0),

new Thetra(5,6,7,13,0),

new Thetra(0,4,9,10,0),

new Thetra(1,5,10,11,0),

new Thetra(3,7,11,12,0),

new Thetra(2,6,9,12,0),

new Thetra(0,1,8,10,0),

new Thetra(4,5,10,13,0),

new Thetra(6,7,12,13,0),

new Thetra(2,3,8,12,0)

};

Func<int, double> m = (int mat) =>

{

return 2;

};

Func<double, double, double, int, double> f = (double x, double y, double z, int mat)=>

{

return 0\*m(mat);

};

Func<double, double, double, double> bound = (double x, double y, double z) =>

{

return x+y+z;

};

List<int> borderPoints = new List<int>() {0,1,2,3,4,5,6,7};

MatrixGenerator c = new MatrixGenerator(p, t, f, m);

double[,] koefs = c.CollectGlobalMatrix();

c.AccountMainCondition(borderPoints, bound);

c.Save();

LOS L = new LOS();

L.Load();

L.FactorLU();

L.Solve();

L.Save("ans.txt");

}

}

}

DenseMatrix.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace NetCheckerFEM

{

public class DenseMatrix : IMatrix

{

public int dimX = 1, dimY = 1;

double[,] data;

public double Det = 0;

public double this[int x, int y]

{

get => data[x, y];

set => data[x, y] = value;

}

public DenseMatrix(int x, int y)

{

dimX = x;

dimY = y;

data = new double[x, y];

}

public static IMatrix DiagonalMatrix(int n, double d)

{

var res = new DenseMatrix(n, n);

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

res[i, i] = d;

return res;

}

public DenseMatrix(int x, int y, double[,] d)

{

dimX = x;

dimY = y;

data = new double[x, y];

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

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

data[i,j] = d[i, j];

}

double GetDet()

{

if (dimX != dimY)

throw new Exception("Matrix not squad");

return GetDet(Enumerable.Range(0, dimX).ToArray(), Enumerable.Range(0, dimX).ToArray(), dimX);

}

double GetDet(int[] X, int[] Y, int m)

{

int[,] switches;

int n;

if (m == 1)

return data[0, 0];

else if (m == 2)

{

switches = new int[2, 3] {

{0,1,1},

{1,0,-1} };

n = 2;

}

else if (m == 3)

{

switches = new int[6, 4] {

{0,1,2,1},

{0,2,1,-1},

{1,0,2,-1},

{1,2,0,1},

{2,0,1,1},

{2,1,0,-1}

};

n = 6;

}

else if (m == 4)

{

switches = new int[24, 5] {

{0,1,2,3,1},

{0,1,3,2,-1},

{0,2,1,3,-1},

{0,2,3,1,1},

{0,3,1,2,1},

{0,3,2,1,-1},

{1,0,2,3,-1},

{1,0,3,2,1},

{1,2,0,3,1},

{1,2,3,0,-1},

{1,3,0,2,-1},

{1,3,2,0,1},

{2,0,1,3,1},

{2,0,3,1,-1},

{2,1,0,3,-1},

{2,1,3,0,1},

{2,3,0,1,1},

{2,3,1,0,-1},

{3,0,1,2,-1},

{3,0,2,1,1},

{3,1,0,2,1},

{3,1,2,0,-1},

{3,2,0,1,-1},

{3,2,1,0,1}

};

n = 24;

}

else

throw new Exception("Not implemented dimension");

double det = 0, tmp;

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

{

tmp = 1;

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

tmp \*= data[X[j], Y[switches[i, j]]];

det += tmp \* switches[i, m];

}

return det;

}

double MinorDet(int k, int l)

{

if (dimX != dimY)

throw new Exception("Matrix not squad");

if(dimX < 1 || dimX > 5)

throw new Exception("Not implemented dimension");

int[] X = Enumerable.Range(0, dimX).Where(x => x != k).ToArray();

int[] Y = Enumerable.Range(0, dimX).Where(x => x != l).ToArray();

return GetDet(X, Y, dimX-1);

}

public IMatrix ReverseDefault()

{

if (dimX != dimY)

throw new Exception("Matrix not squad");

Det = GetDet();

if (Math.Abs(Det) < 1E-10)

throw new Exception("Degenerate matrix");

DenseMatrix X = new DenseMatrix(dimX, dimX);

for (int i = 0; i < 4; i++)

for (int j = 0; j < 4; j++)

X.data[j, i] = (((i + j) & 1) == 0 ? 1 : -1) \* MinorDet(i, j) / Math.Abs(Det);

X.Det = Det;

return X;

}

public IMatrix ReverseMatrix()

{

if (dimX != dimY)

throw new Exception("Matrix not squad");

int n = dimX;

var a = new double[n, n];

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

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

a[i, j] = data[i, j];

DenseMatrix B = (DenseMatrix)DiagonalMatrix(n, 1);

try

{

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

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

{

if (Math.Abs(a[k, k]) < 1e-14)

throw new Exception("Degenerate matrix");

double t = a[i, k] / a[k, k];

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

B[i, j] -= t \* B[k, j];

for (int j = k + 1; j < n; j++)

{

B[i, j] -= t \* B[k, j];

a[i, j] -= t \* a[k, j];

}

}

}

catch (Exception ex)

{

if (ex.Message == "Degenerate matrix")

return ReverseDefault();

}

var X = new DenseMatrix(n, n);

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

X[n - 1, i] = B[n - 1, i] / a[n - 1, n - 1];

for (int k = n - 2; k >= 0; k--)

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

{

double sum = 0;

for (int j = k + 1; j < n; j++)

sum += a[k, j] \* X[j, i];

X[k, i] = (B[k, i] - sum) / a[k, k];

}

X.Det = 1;

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

X.Det \*= a[i, i];

return X;

}

public void Print()

{

Console.WriteLine(ToString().Replace("|", Environment.NewLine));

}

public static DenseMatrix operator \*(DenseMatrix a, DenseMatrix b)

{

if (a.dimX != b.dimY || a.dimY != b.dimX)

throw new Exception("Invalid dimensions");

var res = new DenseMatrix(a.dimY, a.dimY);

int n = a.dimY;

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

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

for (int k = 0; k < a.dimX; k++)

res[i, j] += a[i, k] \* b[k, j];

return res;

}

public override string ToString()

{

var res = "|";

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

{

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

res += Math.Round(data[i, j],2) + " ";

res += "|";

}

return res;

}

}

}

IMatrix.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace NetCheckerFEM

{

public interface IMatrix

{

IMatrix ReverseMatrix();

}

}

MatrixGenerator.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using NetCheckApp;

namespace NetCheckerFEM

{

class MatrixGenerator

{

#region Input data

List<Vector3D> points;

List<Thetra> thetras;

#endregion

#region Local data

DenseMatrix LocalMatrix = new DenseMatrix(4, 4);

double[] LocalRightPart = new double[4];

double[,] LocalM = new double[4, 4]

{

{ 2,1,1,1},

{ 1,2,1,1},

{ 1,1,2,1},

{ 1,1,1,2}

};

#endregion

#region Global data

public SparseMatrix GlobalMatrix;

public double[] RightPart;

public Func<double, double, double, int, double> FunctionRightPart;

public Func<int, double> Material;

#endregion

#region Solver params

int maxiter = 10000;

double eps = 1E-14;

#endregion

#region Basis coefficients

double[,] koefs;

#endregion

public MatrixGenerator(List<Vector3D> \_vector3s, List<Thetra> \_thetras, Func<double, double, double, int, double> rightpart, Func<int, double> mat)

{

points = \_vector3s;

thetras = \_thetras;

GlobalMatrix = new SparseMatrix(\_vector3s.Count, \_thetras);

FunctionRightPart = rightpart;

Material = mat;

RightPart = new double[\_vector3s.Count];

koefs = new double[\_vector3s.Count,4];

}

void MakeLocalMatrix(Thetra t)

{

double[,] data = new double[4, 4] {

{1,1,1,1 },

{points[t[0]].X,points[t[1]].X,points[t[2]].X,points[t[3]].X },

{points[t[0]].Y,points[t[1]].Y,points[t[2]].Y,points[t[3]].Y },

{points[t[0]].Z,points[t[1]].Z,points[t[2]].Z,points[t[3]].Z }

};

DenseMatrix a = (DenseMatrix)(new DenseMatrix(4, 4, data)).ReverseDefault();

double lambda = Material(t.mat);

for (int i = 0; i < 4; i++)

{

LocalRightPart[i] = 0;

for (int j = 0; j < 4; j++)

{

LocalMatrix[i, j] = lambda \* Math.Abs(a.Det) / 6.0 \* (a[i, 1] \* a[j, 1] + a[i, 2] \* a[j, 2] + a[i, 3] \* a[j, 3]);

LocalRightPart[i] += LocalM[i, j] \* FunctionRightPart(points[t[j]].X, points[t[j]].Y, points[t[j]].Z, t.mat);

koefs[t[i], j] = a[i, j];

}

LocalRightPart[i] \*= Math.Abs(a.Det) / 60;

}

}

public double[,] CollectGlobalMatrix()

{

foreach (Thetra item in thetras)

{

MakeLocalMatrix(item);

for (int j = 0; j < 4; j++)

{

for (int k = 0; k < 4; k++)

GlobalMatrix[item[j], item[k]] += LocalMatrix[j, k];

RightPart[item[j]] += LocalRightPart[j];

}

}

return koefs;

}

public void AccountMainCondition(List<int> b\_points, Func<double, double, double, double> bound)

{

foreach (int p in b\_points)

{

GlobalMatrix[p, p] = 1;

RightPart[p] = bound(points[p].X, points[p].Y, points[p].Z);

GlobalMatrix.ZeroRow(p);

}

}

public void Save()

{

string path = AppDomain.CurrentDomain.BaseDirectory;

using (StreamWriter sw = new StreamWriter(Path.Combine(path, "kuslau.txt")))

sw.Write($"{points.Count} {maxiter} {eps}");

using (StreamWriter sw = new StreamWriter(Path.Combine(path, "x.txt")))

sw.Write(Enumerable.Range(0, points.Count).Select(x => "1\n").Aggregate((x, y) => x += y));

using (StreamWriter sw = new StreamWriter(Path.Combine(path, "pr.txt")))

sw.Write(RightPart.Select(x => x.ToString() + "\n").Aggregate((x, y) => x += y));

using (StreamWriter sw = new StreamWriter(Path.Combine(path, "di.txt")))

sw.Write(GlobalMatrix.Diagonal.Select(x => x.ToString() + "\n").Aggregate((x, y) => x += y));

using (StreamWriter sw = new StreamWriter(Path.Combine(path, "ig.txt")))

sw.Write(GlobalMatrix.Ig.Select(x => x.ToString() + "\n").Aggregate((x, y) => x += y));

using (StreamWriter sw = new StreamWriter(Path.Combine(path, "jg.txt")))

sw.Write(GlobalMatrix.Jg.Select(x => x.ToString() + "\n").Aggregate((x, y) => x += y));

using (StreamWriter sw = new StreamWriter(Path.Combine(path, "ggl.txt")))

sw.Write(GlobalMatrix.Ggl.Select(x => x.ToString() + "\n").Aggregate((x, y) => x += y));

using (StreamWriter sw = new StreamWriter(Path.Combine(path, "ggu.txt")))

sw.Write(GlobalMatrix.Ggu.Select(x => x.ToString() + "\n").Aggregate((x, y) => x += y));

}

}

}

NetGenerator.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using NetCheckApp;

namespace NetCheckerFEM

{

class NetGenerator

{

public List<Vector3D> points = new List<Vector3D>();

public List<Thetra> thetras = new List<Thetra>();

public NetGenerator(string path)

{

Load(path);

}

void Load(string path)

{

using (System.IO.StreamReader file = new System.IO.StreamReader(path))

{

bool isNodes = false, isT = false;

string str;

while ((str = file.ReadLine()) != null)

{

if (str == "$Nodes")

isNodes = true;

else if (str == "$Elements")

isT = true;

if (isNodes)

{

while (str != "$EndNodes" && str.Split(' ').Length != 3) str = file.ReadLine();

if (str == "$EndNodes")

{

isNodes = false;

break;

}

var coords = file.ReadLine().Split(' ').Select(x => Double.Parse(x)).ToArray();

points.Add(new Vector3D(coords[0], coords[1], coords[2]));

}

else if (isT)

{

while (str != "$EndElements" && str.Split(' ').Length != 5) str = file.ReadLine();

if (str == "$EndElements")

{

isT = false;

break;

}

var coords = file.ReadLine().Split(' ').Select(x => Int32.Parse(x)).ToArray();

thetras.Add(new Thetra(coords[0], coords[1], coords[2], coords[3], coords[4]));

}

}

}

}

}

}

Solve.cs

using NetCheckApp;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace NetCheckerFEM

{

class Solve

{

List<Vector3D> points;

List<Thetra> thetras;

double[,] coefficients;

double[] q;

Func<double, double, double, double> exact;

Vector3D max, min;

double step = 0.5;

public Solve(List<Vector3D> \_points, List<Thetra> \_thetras, double[,] \_coefficients, double[] \_q, Func<double, double, double, double> \_exact)

{

q = \_q;

points = \_points;

max = points[0];

min = points[0];

foreach (Vector3D item in points)

{

max = Vector3D.Max(item, max);

min = Vector3D.Min(item, min);

}

thetras = \_thetras;

coefficients = \_coefficients;

exact = \_exact;

}

bool IsContains(Vector3D p, Thetra t)

{

Vector3D a, b, c;

bool ans = true;

int[,] indexes = new int[4, 4]

{

{ 1,2,0,3},

{ 3,2,1,0},

{ 1,3,0,2},

{ 2,3,0,1}

};

for(int i=0;i<4 && ans;i++)

{

a = Vector3D.Cross(points[t[indexes[i,0]]] - points[t[indexes[i, 2]]], points[t[indexes[i, 1]]] - points[t[indexes[i, 2]]]);

b = points[t[indexes[i, 3]]] - (points[t[indexes[i, 0]]] + points[t[indexes[i, 1]]] + points[t[indexes[i, 2]]])/3.0;

c = p - (points[t[indexes[i, 0]]] + points[t[indexes[i, 1]]] + points[t[indexes[i, 2]]])/3.0;

ans &= !((a \* b >= 0) ^ (a \* c >= 0));

}

return ans || t.data.Select(x => points[x]).Contains(p);

}

public double U(Vector3D p)

{

double ans = 0;

foreach (Thetra t in thetras)

if (IsContains(p, t))

{

for (int i = 0; i < 4; i++)

ans += (coefficients[t[i], 0] + coefficients[t[i], 1] \* p.X + coefficients[t[i], 2] \* p.Y + coefficients[t[i], 3] \* p.Z) \* q[i];

break;

}

return ans;

}

public void PrintAbsDiff()

{

int i = 0;

foreach (Vector3D item in points)

Console.WriteLine("#{0}\t{1:#.####E+00} \t{2:#.####E+00} \t{3:#.####E+00}",

i, q[i], exact(item.X, item.Y, item.Z), Math.Abs(q[i++] - exact(item.X, item.Y, item.Z))

);

}

public void Resolve()

{

StreamWriter sw = new StreamWriter(Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "resolve.txt"));

double stepX = (max.X-min.X)\*step, stepY = (max.Y - min.Y) \* step, stepZ = (max.Z - min.Z) \* step;

int n = (int)(1 / step)+1;

string top = "\t" + Enumerable.Range(0, n).Select(x => (x \* stepY).ToString("000.0000\t")).Aggregate((x, y) => x += y);

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

{

Console.WriteLine($"{i \* stepZ:0.00e+00}" + top);

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

{

Console.Write($"{j \* stepY:0.00e+00}\t");

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

Console.Write($"{U(min + new Vector3D(k \* stepX, j \* stepY, i \* stepZ)):0.00e+00}\t");

Console.WriteLine();

}

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

}

sw.Close();

}

}

}

SparseMatrix.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using NetCheckApp;

namespace NetCheckerFEM

{

class SparseMatrix

{

public int dim = 1;

double[][] data;//2 x n

public double Det = 0;

double[] di;

int[] ig, jg;

int curElem = 0;

public double[] Diagonal { get => di; }

public int[] Ig { get => ig; }

public int[] Jg { get => jg; }

public double[] Ggu { get => data[1]; }

public double[] Ggl { get => data[0]; }

public double this[int x, int y]

{

get {

if (x > dim || y > dim)

throw new Exception("Not found");

if (x == y)

return di[x];

int isUp = 0;

if(x < y)

{

isUp = x;

x = y;

y = isUp;

isUp = 1;

}

int i = ig[x];

for (int n = ig[x + 1] + 1; i < n && jg[i] != y; i++) ;

if (i == ig[x + 1] + 1)

throw new Exception("Not found");

return data[isUp][i];

}

set

{

if (x > dim || y > dim)

throw new Exception("Not found");

if (x == y)

di[x] = value;

else

{

int isUp = 0;

if (x < y)

{

isUp = x;

x = y;

y = isUp;

isUp = 1;

}

int i = ig[x];

for (int n = ig[x + 1] + 1; i < n && jg[i] != y; i++) ;

if (i == ig[x + 1] + 1)

throw new Exception("Not found");

data[isUp][i] = value;

}

}

}

public SparseMatrix(int n)

{

dim = n;

data = new double[2][];

data[0] = new double[n];

data[1] = new double[n];

di = new double[n];

}

public SparseMatrix(int num, List<Thetra> thetras)

{

dim = num;

HashSet<int>[] edges = new HashSet<int>[num];

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

edges[i] = new HashSet<int>();

foreach (Thetra item in thetras)

for (int i = 0; i < 4; i++)

for (int j = i + 1; j < 4; j++)

edges[item[j]].Add(item[i]);

int nJg = 0;

ig = new int[num + 1];

ig[0] = 0;

ig[1] = 0;

for (int i = 2; i <= num; i++)

{

ig[i] = ig[i - 1] + edges[i-1].Count;

nJg += edges[i-1].Count;

}

jg = new int[nJg];

for (int i = 0, k = 0; i < num; i++)

foreach (int var in edges[i].OrderBy(x => x))

jg[k++] = var;

data = new double[2][];

data[0] = new double[nJg];

data[1] = new double[nJg];

di = new double[num];

}

public void ZeroRow(int p)

{

int ibeg, iend;

bool flag = true;

ibeg = ig[p];

iend = ig[p+1];

for (int i= ibeg, n = iend; i < n; i++)

data[0][i] = 0;

for (int i = p + 1, n = dim, tmp = -1, ind; i < n; i++)

{

iend = ig[i+1];

ibeg = ig[i];

if ((iend > 0 && jg[iend - 1] < p) || jg[ibeg] > p)

continue;

while (jg[ibeg] != p && flag)//binary find

{

ind = ((ibeg + iend) % 2) != 0 ? (ibeg + iend) / 2 + 1 : (ibeg + iend) / 2;

flag = ind != tmp;

if (jg[ind] <= p)

ibeg = ind;

else

iend = ind;

tmp = ind;

}

if (flag)//если нашли индекс в ibeg

data[1][ibeg] = 0;

else

flag = true;

}

}

public void Print()

{

int ind = 0;

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

{

Console.Write($"#{i:D2}\t");

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

if (jg[ind] == j)

{ Console.Write($"{Math.Round(data[0][ind],2)} "); ind++; }

else Console.Write("0.00 ");

Console.Write($"{Math.Round(di[i], 2)} ");

for (int j = i + 1; j < dim; j++)

{

int k = ig[j], n = ig[j + 1];

for (; k < n; k++)

if (jg[k] == i)

{ Console.Write($"{Math.Round(data[1][k], 2)} "); break; }

if(k==n)

Console.Write("0.00 ");

}

Console.WriteLine();

}

}

public void Add(bool isUpperTriangle, double d)

{

if (curElem > dim)

{ Console.WriteLine("WARNING! Current value overflow"); curElem = 0; }

data[isUpperTriangle ? 1 : 0][curElem++] = d;

}

}

}

TSM.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace NetCheckerFEM

{

//ThreeStepMethod

class LOS

{

double normar;

int n, maxiter, iter;

double eps;

double[] gglLU, gguLU, diLU;

double[] gglA, gguA, diA;

double[] diD;

int[] ig, jg;

double[] r, z, x, p, pr, Ax, Ay, x0;

//c=b-a

public void sub(double[] a, double[] b, double[] c)

{

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

c[i] = b[i] - a[i];

}

//return (a,b)

public double scalar(double[] a, double[] b)

{

double ans = 0;

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

ans += a[i] \* b[i];

return ans;

}

//d=a+b\*c

public void addmult(double[] a, double b, double[] c, double[] d)

{

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

d[i] = a[i] + b \* c[i];

}

public double[] GetAnswer()

{

return x;

}

public void Load()

{

string[] line;

string path = AppDomain.CurrentDomain.BaseDirectory;

StreamReader file = new StreamReader(Path.Combine(path, "kuslau.txt"));

line = file.ReadLine().Split(' ');

n = Int32.Parse(line[0]);

maxiter = Int32.Parse(line[1]);

eps = Double.Parse(line[2]);

diLU = new double[n];

diA = new double[n];

diD = new double[n];

ig = new int[n + 1];

pr = new double[n];

r = new double[n];

z = new double[n];

x = new double[n];

p = new double[n];

Ax = new double[n];

Ay = new double[n];

x0 = new double[n];

using (StreamReader sr = new StreamReader(Path.Combine(path, "x.txt")))

{

//var qw = sr.ReadToEnd().Split('\n');

x0 = sr.ReadToEnd().Split('\n').Take(n).Select(x => Double.Parse(x)).ToArray();

x = x0.ToArray();

}

using (StreamReader sr = new StreamReader(Path.Combine(path, "pr.txt")))

pr = sr.ReadToEnd().Split('\n').Take(n).Select(x => Double.Parse(x)).ToArray();

using (StreamReader sr = new StreamReader(Path.Combine(path, "di.txt")))

diA = sr.ReadToEnd().Split('\n').Take(n).Select(x => Double.Parse(x)).ToArray();

using (StreamReader sr = new StreamReader(Path.Combine(path, "ig.txt")))

ig = sr.ReadToEnd().Split('\n').Take(n+1).Select(x => Int32.Parse(x)).ToArray();

int tmp = ig[n];

jg = new int[tmp];

gglA = new double[tmp];

gguA = new double[tmp];

gglLU = new double[tmp];

gguLU = new double[tmp];

using (StreamReader sr = new StreamReader(Path.Combine(path, "jg.txt")))

jg = sr.ReadToEnd().Split('\n').Take(tmp).Select(x => Int32.Parse(x)).ToArray();

using (StreamReader sr = new StreamReader(Path.Combine(path, "ggl.txt")))

gglA = sr.ReadToEnd().Split('\n').Take(tmp).Select(x => Double.Parse(x)).ToArray();

using (StreamReader sr = new StreamReader(Path.Combine(path, "ggu.txt")))

gguA = sr.ReadToEnd().Split('\n').Take(tmp).Select(x => Double.Parse(x)).ToArray();

}

public void Save(string path)

{

using (StreamWriter sr = new StreamWriter(path))

sr.Write(x.Select(a => a.ToString()).Aggregate((x, y) => x + Environment.NewLine + y));

}

//Ax = y умножение в разряженном формате

//y - можно любой

public void MultMatrix(double[] x, double[] y, bool f)

{

if (f)

{

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

y[i] = diA[i] \* x[i];

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

for (int k = ig[i], m = ig[i + 1]; k < m; k++)

{

y[i] += gglA[k] \* x[jg[k]];

y[jg[k]] += gguA[k] \* x[i];

}

}

else

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

y[i] = diD[i] \* x[i];

}

public void Solve()

{

MultMatrix(x, Ax, true);

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

{

r[i] = pr[i] - Ax[i];

z[i] = r[i];

}

MultMatrix(z, Ax, true);

var tttt = new double[x.Length];

tttt = Ax;

Ax = p;

p = tttt;

normar = scalar(r, r);

for (iter = 0; iter < maxiter && Math.Abs(normar) > eps && Math.Abs(normar) > 1E-30; iter++)

{

double normap = scalar(p, p);

double a = scalar(p, r) / normap;

addmult(x, a, z, x);

addmult(r, -a, p, r);

MultMatrix(r, Ax, true);

double b = -scalar(p, Ax) / normap;

addmult(r, b, z, z);

normar -= a \* a \* normap;

addmult(Ax, b, p, p);

}

}

public void FactorD()

{

try

{

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

{

if (diA[i] < 1E-30)

throw new Exception();

diD[i] = 1 / Math.Sqrt(diA[i]);

}

}

catch (Exception)

{

Console.WriteLine("Divided by zero!");

}

}

public void SolveD()

{

MultMatrix(x, Ax, true);

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

{

r[i] = diD[i] \* (pr[i] - Ax[i]);

z[i] = diD[i] \* r[i];

}

MultMatrix(z, Ax, true);

MultMatrix(Ax, p, false);

normar = scalar(r, r);

for (iter = 0; iter < maxiter && Math.Abs(normar) > eps && Math.Abs(normar) > 1E-30; iter++)

{

double normap = scalar(p, p);

double a = scalar(p, r) / normap;

addmult(x, a, z, x);

addmult(r, -a, p, r);

MultMatrix(r, Ax, false);

MultMatrix(Ax, Ay, true);

MultMatrix(Ay, Ay, false);

double b = -scalar(p, Ay) / normap;

addmult(Ax, b, z, z);

normar -= a \* a \* normap;

addmult(Ay, b, p, p);

}

}

public void FactorLU()

{

int j0 = 0;

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

{

int m = ig[i + 1] - ig[i];

double sum = 0;

if (m != 0)

{

int jm = j0 + m;

for (int j = j0; j < jm; j++)

{

int stolbez = jg[j];

double sumL = 0, sumU = 0;

if (j != j0)//npred!=0

for (int l = ig[i], nl = j, nu = ig[stolbez + 1]; l < nl; l++)

for (int u = ig[stolbez]; u < nu; u++)

{

int gl = jg[l], gu = jg[u];

if (gl == gu)

{

sumL += gguLU[u] \* gglLU[l];

sumU += gguLU[l] \* gglLU[u];

}

}

gglLU[j] = (gglA[j] - sumL) / diLU[jg[j]];

gguLU[j] = (gguA[j] - sumU) / diLU[jg[j]];

sum += gglLU[j] \* gguLU[j];

}

j0 = jm;

}

diLU[i] = Math.Sqrt(diA[i] - sum);

}

}

//Uy = x решение слау

//н - любой

public void SLAEU(double[] x, double[] y)

{

for (int i = 0; i < n; i++) y[i] = x[i];

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

{

y[i] /= diLU[i];

int kend = ig[i + 1];

int k = ig[i];

for (; k != kend; k++)

y[jg[k]] -= y[i] \* gguLU[k];

}

}

//Ly = x решение слау

//y - любой

public void SLAEL(double[] x, double[] y)

{

double sum;

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

{

sum = 0;

int k0 = ig[i];

int kend = ig[i + 1];

for (int k = k0; k < kend; k++)

sum += y[jg[k]] \* gglLU[k];

y[i] = (x[i] - sum) / diLU[i];

}

}

public void InitLU()

{

MultMatrix(x, r, true);

sub(r, pr, r);

SLAEL(r, r);

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

z[i] = r[i];

SLAEU(z, z);

MultMatrix(z, p, true);

SLAEL(p, p);

normar = scalar(r, r);

}

public void SolveLU()

{

InitLU();

for (iter = 0; iter < maxiter && normar > eps && normar > 1E-30; iter++)

{

double normap = scalar(p, p);

double a = scalar(p, r) / normap;

addmult(x, a, z, x);

addmult(r, -a, p, r);

//(L-1)A(U-1)r = Ay

//U-1r = Ax

SLAEU(r, Ax);

MultMatrix(Ax, Ay, true);

SLAEL(Ay, Ay);

double b = -scalar(p, Ay) / normap;

addmult(Ax, b, z, z);

normar = Math.Abs(normar - a \* a \* normap);

addmult(Ay, b, p, p);

if (Math.Abs(a \* a \* normap) < 1E-20)

InitLU();

}

}

public void ClearX()

{

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

x[i] = x0[i];

}

public bool TestLU()

{

n = 4;

maxiter = 1000;

eps = 1E-9;

diLU = new double[n];

diA = new double[n];

diD = new double[n];

ig = new int[n + 1];

pr = new double[n];

r = new double[n];

z = new double[n];

x = new double[n];

p = new double[n];

Ax = new double[n];

Ay = new double[n];

x0 = new double[n];

ig = "0 0 1 2 5".Split(' ').Select(x => Int32.Parse(x)).ToArray();

int tmp = ig[n];

jg = new int[tmp];

gglA = new double[tmp];

gguA = new double[tmp];

gglLU = new double[tmp];

gguLU = new double[tmp];

jg = "0 1 0 1 2".Split(' ').Select(x => Int32.Parse(x)).ToArray();

pr = "-32 215 29 92".Split(' ').Select(x => Double.Parse(x)).ToArray();

diA = "64 43 21 20".Split(' ').Select(x => Double.Parse(x)).ToArray();

gglA = "-24 7 16 4 -4".Split(' ').Select(x => Double.Parse(x)).ToArray();

gguA = "16 35 -32 12 -12".Split(' ').Select(x => Double.Parse(x)).ToArray();

FactorLU();

SolveLU();

double[] answer = new double[n];

answer = "1 2 3 4".Split(' ').Select(x => Double.Parse(x)).ToArray();

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

if (Math.Abs(answer[i] - x[i]) > eps)

return false;

return true;

}

};

}

Vectors.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Runtime.Serialization;

namespace NetCheckApp

{

struct Thetra

{

public int[] data;

public int mat;

public Thetra(int \_a, int \_b, int \_c, int \_d, int \_m)

{

data = new int[4];

data[0] = \_a;

data[1] = \_b;

data[2] = \_c;

data[3] = \_d;

mat = \_m;

}

public int this[int x]

{

get => data[x];

}

public override string ToString()

{

return $"{data[0]} {data[1]} {data[2]} {data[3]} | {mat}";

}

}

public struct Vector3D : IEquatable<Vector3D>

{

public static readonly Vector3D Zero = new Vector3D(0, 0, 0);

public static readonly Vector3D XAxis = new Vector3D(1, 0, 0);

public static readonly Vector3D YAxis = new Vector3D(0, 1, 0);

public static readonly Vector3D ZAxis = new Vector3D(0, 0, 1);

public static readonly Vector3D[] Axes = { XAxis, YAxis, ZAxis };

public double X { get; }

public double Y { get; }

public double Z { get; }

public Vector3D(double x, double y, double z)

{

X = x;

Y = y;

Z = z;

}

public double Distance(Vector3D b) => Distance(this, b);

public double SqrDistance(Vector3D b) => SqrDistance(this, b);

public static double Distance(Vector3D a, Vector3D b) => (a - b).Norm;

public static double SqrDistance(Vector3D a, Vector3D b)

{

var diff = a - b;

return diff \* diff;

}

public double this[int i]

{

get

{

switch (i)

{

case 0: return X;

case 1: return Y;

case 2: return Z;

default: throw new IndexOutOfRangeException();

}

}

}

public double[] AsArray() => new[] { X, Y, Z };

public double Norm => Math.Sqrt(X \* X + Y \* Y + Z \* Z);

public double MaxNorm => Math.Max(Math.Abs(X), Math.Max(Math.Abs(Y), Math.Abs(Z)));

public Vector3D Projection(Vector3D p) => (this \* p) \* p;

public Vector3D Normalize() => this / Norm;

public Vector3D Round(int digits) => new Vector3D(Math.Round(X, digits), Math.Round(Y, digits), Math.Round(Z, digits));

public override string ToString() => $"Vec({X}, {Y}, {Z})";

public override bool Equals(object obj) => obj is Vector3D && Equals((Vector3D)obj);

public override int GetHashCode() => X.GetHashCode() ^ Y.GetHashCode() ^ Z.GetHashCode();

public bool Equals(Vector3D a) => a.X == X && a.Y == Y && a.Z == Z;

public static bool TryParse(string line, out Vector3D res)

{

double x, y, z;

var words = line.Split(new[] { ' ', '\t', ',', '(', ')', '<', '>' }, StringSplitOptions.RemoveEmptyEntries);

if (words[0] == "Vec")

{

if (words.Length != 4 || !double.TryParse(words[1], out x) || !double.TryParse(words[2], out y)

|| !double.TryParse(words[3], out z))

{

res = Vector3D.Zero;

return false;

}

res = new Vector3D(x, y, z);

return true;

}

if (words.Length != 3 || !double.TryParse(words[0], out x) || !double.TryParse(words[1], out y)

|| !double.TryParse(words[2], out z))

{

res = Vector3D.Zero;

return false;

}

res = new Vector3D(x, y, z);

return true;

}

public static Vector3D Vec(double x, double y, double z) => new Vector3D(x, y, z);

public static Vector3D Parse(string line)

{

Vector3D res;

if (!TryParse(line, out res))

throw new FormatException("Can't parse Vector3D!");

return res;

}

#region Static operators

[MethodImpl(MethodImplOptions.AggressiveInlining)]

public static Vector3D operator -(Vector3D a) => new Vector3D(-a.X, -a.Y, -a.Z);

[MethodImpl(MethodImplOptions.AggressiveInlining)]

public static Vector3D operator +(Vector3D a) => a;

[MethodImpl(MethodImplOptions.AggressiveInlining)]

public static double operator \*(Vector3D a, Vector3D b) => a.X \* b.X + a.Y \* b.Y + a.Z \* b.Z;

[MethodImpl(MethodImplOptions.AggressiveInlining)]

public static Vector3D operator \*(double a, Vector3D b) => new Vector3D(a \* b.X, a \* b.Y, a \* b.Z);

[MethodImpl(MethodImplOptions.AggressiveInlining)]

public static Vector3D operator \*(Vector3D b, double a) => new Vector3D(a \* b.X, a \* b.Y, a \* b.Z);

[MethodImpl(MethodImplOptions.AggressiveInlining)]

public static Vector3D operator /(Vector3D a, double v) => new Vector3D(a.X / v, a.Y / v, a.Z / v);

[MethodImpl(MethodImplOptions.AggressiveInlining)]

public static Vector3D operator +(Vector3D a, Vector3D b) => new Vector3D(a.X + b.X, a.Y + b.Y, a.Z + b.Z);

[MethodImpl(MethodImplOptions.AggressiveInlining)]

public static Vector3D operator -(Vector3D a, Vector3D b) => new Vector3D(a.X - b.X, a.Y - b.Y, a.Z - b.Z);

[MethodImpl(MethodImplOptions.AggressiveInlining)]

public static bool operator ==(Vector3D a, Vector3D b) => a.X == b.X && a.Y == b.Y && a.Z == b.Z;

[MethodImpl(MethodImplOptions.AggressiveInlining)]

public static bool operator !=(Vector3D a, Vector3D b) => a.X != b.X || a.Y != b.Y || a.Z != b.Z;

[MethodImpl(MethodImplOptions.AggressiveInlining)]

public static Vector3D Cross(Vector3D v1, Vector3D v2) =>

new Vector3D(v1.Y \* v2.Z - v2.Y \* v1.Z, v1.Z \* v2.X - v1.X \* v2.Z, v1.X \* v2.Y - v1.Y \* v2.X);

[MethodImpl(MethodImplOptions.AggressiveInlining)]

public static double Mixed(Vector3D v1, Vector3D v2, Vector3D v3) =>

(v1.Y \* v2.Z - v2.Y \* v1.Z) \* v3.X + (v1.Z \* v2.X - v1.X \* v2.Z) \* v3.Y + (v1.X \* v2.Y - v1.Y \* v2.X) \* v3.Z;

public static Vector3D Sum(Vector3D a, Vector3D b) => a + b;

public static Vector3D Min(Vector3D a, Vector3D b) =>

new Vector3D(Math.Min(a.X, b.X), Math.Min(a.Y, b.Y), Math.Min(a.Z, b.Z));

public static Vector3D Max(Vector3D a, Vector3D b) =>

new Vector3D(Math.Max(a.X, b.X), Math.Max(a.Y, b.Y), Math.Max(a.Z, b.Z));

#endregion

}

}