using System;

using System.Collections.Generic;

using System.IO;

namespace Laba2

{

class Laba

{

static string outpath = "C:\\Users\\Vladimir\\Desktop\\Учеба 3 курс\\Численные методы\\2 лаба\\output.txt";

static string inpath = "C:\\Users\\Vladimir\\Desktop\\Учеба 3 курс\\Численные методы\\2 лаба\\input.txt";

//произведение матрицы и вектора

static void GetMultiVM(List<List<double>> L, List<double> b, List<double> y)

{

for (int i = 0; i < y.Count; i++)

{

y[i] = 0;

for (int j = 0; j < b.Count; j++)

{

y[i] += L[i][j] \* b[j];

}

}

}

//получение единичной матрицы

static List<List<double>> GetIdentityMatrix(int n)

{

List<List<double>> A = new List<List<double>>();

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

{

A.Add(new List<double>());

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

{

if (i == j)

{

A[i].Add(1);

}

else

{

A[i].Add(0);

}

}

}

return A;

}

//метод меняющий строки матрицы местами

static void SwapRows(List<List<double>> A, int i, int j)

{

double tmp;

for (int k = 0; k < A.Count; k++)

{

tmp = A[i][k];

A[i][k] = A[j][k];

A[j][k] = tmp;

}

}

//вывод матрицы в файл

static void WriteMatrix(List<List<double>> A)

{

using (StreamWriter file = File.AppendText(outpath))

{

for (int i = 0; i < A.Count; i++)

{

string str = "";

for (int j = 0; j < A[i].Count; j++)

{

str += String.Format(" {0,10:f7} ", A[i][j]);

}

file.WriteLine(str);

}

}

}

//Вывод матрицы в е форме

static void WriteMatrixform16(List<List<double>> A)

{

using (StreamWriter file = File.AppendText(outpath))

{

for (int i = 0; i < A.Count; i++)

{

string str = "";

for (int j = 0; j < A[i].Count; j++)

{

if (A[i][j] != 0)

str += String.Format(" {0,10:e16} ", A[i][j]);

else

str += String.Format(" {0,10:f16} ", A[i][j]);

}

file.WriteLine(str);

}

}

}

//вывод вектора в файл

static void WriteVector(List<double> A)

{

using (StreamWriter file = File.AppendText(outpath))

{

string str = "";

for (int i = 0; i < A.Count; i++)

{

str += String.Format(" {0,10:f7} ", A[i]);

}

file.WriteLine(str);

}

}

//вывод вектора с 16 знаками после запятой

static void WriteVectorform16(List<double> A)

{

using (StreamWriter file = File.AppendText(outpath))

{

string str = "";

for (int i = 0; i < A.Count; i++)

{

str += String.Format(" {0,10:f16} ", A[i]);

}

file.WriteLine(str);

}

}

//вывод слова в файл

static void AppendFile(string str)

{

using (StreamWriter file = File.AppendText(outpath))

{

file.WriteLine(str);

}

}

//инициализация матрицы

static List<List<double>> InitNewMatrix(int n)

{

List<List<double>> A = new List<List<double>>();

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

{

A.Add(new List<double>());

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

{

A[i].Add(0);

}

}

return A;

}

//создание копии матрицы

static List<List<double>> GetMatrixCopy(List<List<double>> A)

{

List<List<double>> B = new List<List<double>>();

for (int i = 0; i < A.Count; i++)

{

B.Add(new List<double>());

for (int j = 0; j < A[i].Count; j++)

{

B[i].Add(A[i][j]);

}

}

return B;

}

//произведение матриц

static List<List<double>> MatrixMultiply(List<List<double>> A, List<List<double>> B)

{

List<List<double>> R = new List<List<double>>();

for (int i = 0; i < A.Count; i++)

{

R.Add(new List<double>());

for (int j = 0; j < A[i].Count; j++)

{

R[i].Add(0);

for (int k = 0; k < B[i].Count; k++)

{

R[i][j] += A[i][k] \* B[k][j];

}

}

}

return R;

}

//разность векторов

static List<double> VectorDiferc(List<double> A, List<double> B)

{

List<double> R = new List<double>();

for (int i = 0; i < A.Count; i++)

{

R.Add(A[i] - B[i]);

}

return R;

}

//нахождение вектора x

static List<double> GetX(List<double> b, List<List<double>> L, List<List<double>> U)

{

//Ly = b

List<double> x = new List<double>();

for (int i = 0; i < b.Count; i++)

{

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

{

b[i] -= L[i][j] \* x[j];

}

x.Add(b[i] / L[i][i]);

}

//Ux = x

for (int i = U.Count - 1; i >= 0; i--)

{

for (int j = U[i].Count - 1; j > i; j--)

{

x[i] -= U[i][j] \* x[j];

}

}

return x;

}

//обратная матрица

static List<List<double>> GetMatrixReverse(List<List<double>> L, List<List<double>> U, List<List<double>> P)

{

int n = U.Count;

List<List<double>> R = InitNewMatrix(n);

List<double> x, e = new List<double>();

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

{

e.Add(0);

}

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

{

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

{

e[i] = 0;

}

e[k] = 1;

x = GetX(e, L, U);

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

{

R[i][k] = x[i];

}

}

return MatrixMultiply(R, P);//возвращаем в исходный вид

}

//транспонирование матрицы

static void MatrixTranspose(List<List<double>> A)

{

int n = A.Count;

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

{

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

{

double temp = A[i][j];

A[i][j] = A[j][i];

A[j][i] = temp;

}

}

}

//разность матриц

static List<List<double>> MatrixDiferc(List<List<double>> A, List<List<double>> B)

{

List<List<double>> R = new List<List<double>>();

for (int i = 0; i < A.Count; i++)

{

R.Add(new List<double>());

for (int j = 0; j < A[i].Count; j++)

{

R[i].Add(A[i][j] - B[i][j]);

}

}

return R;

}

//Получение диагональной матрицы собственных значений

static List<List<double>> Jacobi\_Rotation(List<List<double>> C)

{

double theta, c, s;

double eps = 0.0000001;

while (true)

{

for (int i = 0; i < C.Count - 1; i++)

{

for (int j = i + 1; j < C.Count; j++)

{

int t1 = 0, t2 = 1;

for (int c1 = 0; c1 < C.Count - 1; c1++)

for (int c2 = c1 + 1; c2 < C.Count; c2++)

if (Math.Abs(C[c1][c2]) > Math.Abs(C[t1][t2])) { t1 = c1; t2 = c2; }

if (Math.Abs(C[t1][t2]) < eps) return C;

if (C[t1][t1] == C[t2][t2]) theta = Math.PI / 4;

else theta = Math.Atan(2 \* C[t1][t2] / (C[t1][t1] - C[t2][t2])) / 2;

c = Math.Cos(theta);

s = Math.Sin(theta);

for (int m = 0; m < C.Count; m++)

{

double qi = C[m][t1]; double qj = C[m][t2];

C[m][t1] = c \* qi + s \* qj;

C[m][t2] = -s \* qi + c \* qj;

}

for (int m = 0; m < C.Count; m++)

{

double qi = C[t1][m]; double qj = C[t2][m];

C[t1][m] = c \* qi + s \* qj;

C[t2][m] = -s \* qi + c \* qj;

}

}

}

}

}

//Кубическая норма (норма 1)

static double FirstNorm(List<List<double>> A)

{

double max = 0;

for (int i = 0; i < A.Count; i++)

{

double sum = 0;

for (int j = 0; j < A.Count; j++)

{

sum += Math.Abs(A[i][j]);

}

max = sum > max ? sum : max;

}

return max;

}

//Октаэдрическая норма (норма 2)

static double SecondNorm(List<List<double>> A)

{

double max = 0; int n = A.Count;

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

{

double sum = 0;

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

{

sum += Math.Abs(A[j][i]);

}

max = sum > max ? sum : max;

}

return max;

}

//Евклидовая норма (норма 3)

static double ThirdNorm(List<List<double>> A)

{

int n = A.Count;

List<List<double>> At = GetMatrixCopy(A);

MatrixTranspose(At);

List<List<double>> AtA = MatrixMultiply(At, A);

AtA = Jacobi\_Rotation(AtA);

double max = AtA[0][0];

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

if (max < AtA[i][i])

max = AtA[i][i];

return Math.Sqrt(max);

}

//LU

static List<List<double>> CreateLU(List<List<double>> A, List<List<double>> L0, List<int> swap\_col)

{

swap\_col.Add(0);

int n = A.Count;

List<List<double>> ACopy = GetMatrixCopy(A);

List<List<double>> P = GetIdentityMatrix(n);

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

{

double maxElem = Math.Abs(A[k][k]);

int indMaxElem = k;

//поиск максимального по столбцу элемента

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

{

if (Math.Abs(A[i][k]) > maxElem)

{

maxElem = Math.Abs(A[i][k]);

indMaxElem = i;

}

}

//смена строк местами (если это необходимо)

if (k != indMaxElem)

{

SwapRows(A, k, indMaxElem);

SwapRows(L0, k, indMaxElem);

SwapRows(ACopy, k, indMaxElem);

SwapRows(P, k, indMaxElem);

swap\_col[0]++;

AppendFile(String.Format("Строки {0} и {1} поменялись местами: ", k + 1, indMaxElem + 1));

}

double tmp = A[k][k];

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

{

A[k][j] /= tmp;

}

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

{

double tmp\_loc = A[i][k];

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

{

A[i][j] -= A[k][j] \* tmp\_loc;

}

}

L0[k][0] = ACopy[k][0];

int r = k;

if (k > 0)

{

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

{

if (j <= r)

{

int y = 0, x = 0;

L0[r][j] += ACopy[r][j];

while (x < j)

{

if (x == 0)

{

L0[r][j] -= ACopy[r][x] \* A[y][j];

}

else

{

L0[r][j] -= L0[r][x] \* A[y][j];

}

x++;

y++;

}

}

}

}

AppendFile("U: ");

WriteMatrix(A);

AppendFile("L: ");

WriteMatrix(L0);

AppendFile("\n\n");

}

return P;

}

//Вычисление определителя

static double Determinant(List<List<double>> L, int swap\_col)

{

double det\_L = 1, det;

for (int i = 0; i < L.Count; i++)

det\_L \*= L[i][i];

if (swap\_col % 2 != 0)

det = det\_L \* (-1);

else

det = det\_L;

return det;

}

//

static void MatrixPtoVector(List<List<double>> P)

{

List<int> result = new List<int>(P.Count);

for (int i = 0; i < P.Count; i++)

for (int j = 0; j < P[i].Count; j++)

{

if (P[i][j] != 0)

result.Add(j);

}

using (StreamWriter file = File.AppendText(outpath))

{

string str = "";

for (int i = 0; i < result.Count; i++)

{

str += String.Format(" {0} ", result[i]);

}

file.WriteLine(str);

}

}

//Выполнение программы

static void Main(string[] args)

{

List<List<double>> A = new List<List<double>>();

List<int> swap\_col = new List<int>();

string[] lines = File.ReadAllLines(inpath);

foreach (string l in lines)

{

List<double> row = new List<double>();

string[] numbers = System.Text.RegularExpressions.Regex.Split(l, " ");

foreach (string number in numbers)

{

row.Add(Double.Parse(number));

}

A.Add(row);

}

int n = A.Count;

List<List<double>>

U = GetMatrixCopy(A),

L = InitNewMatrix(n);

//Вывод матрицы А в самом начале

AppendFile("A matrix: ");

WriteMatrix(A);

//Разложение по методу Гаусса матрицы А на произведение матриц L\*U и получаем матрицу перестановок на выходе. Пошаговый вывод матриц L U

List<List<double>> P = CreateLU(U, L, swap\_col);

//Вывод матриц

AppendFile("ИТОГ");

AppendFile("A matrix:");

WriteMatrix(A);

AppendFile("U matrix:");

WriteMatrix(U);

AppendFile("L matrix:");

WriteMatrix(L);

//Поиск и вывод вектора b, инициализация вектора х

List<double> b = new List<double>();

List<double> x = new List<double>();

List<double> b\_r = new List<double>();

for (int i = 0; i < A.Count; i++)

{

b.Add(0);

b\_r.Add(0);

x.Add(i + 1);

}

GetMultiVM(A, x, b);

AppendFile("\nvector b:");

WriteVector(b);

//Вывод вектора х

AppendFile("\nvector x:");

GetMultiVM(P, b, b\_r);

x = GetX(b\_r, L, U);

WriteVectorform16(x);

//Вывод получившейся матрицы L\*U-P\*A, чтобы показать, что L\*U ~= P\*A

AppendFile("\n\n\nL\*U-P\*A");

List<List<double>> LU, PA, Z;

LU = MatrixMultiply(L, U);

PA = MatrixMultiply(P, A);

Z = MatrixDiferc(LU, PA);

WriteMatrixform16(Z);

//Вывод определителя матрицы

double det2 = Determinant(L, swap\_col[0]);

AppendFile(String.Format("\n\n\nDeterminant: {0,10:f7}", det2));

//Вывод вектора перестановок

//AppendFile("\nP:");

//WriteMatrix(P);

AppendFile("\nPvector:");

MatrixPtoVector(P);

//Вывод матрицы А

AppendFile("\nA:");

WriteMatrix(A);

//Вывод матрицы At

AppendFile("\n\n\nA^(-1):");

List<List<double>> A\_r = GetMatrixReverse(L, U, P);

WriteMatrix(A\_r);

//Промежуточный результат A\*At = E

AppendFile("\n\n\nA\*A^(-1)");

List<List<double>> A\_r\_r = MatrixMultiply(A, A\_r);

WriteMatrix(A\_r\_r);

//Вывод числа обусловленности исходной матрицы с использованием различных подчиненных норм

AppendFile(String.Format("\n\n\n" +

"Число обусловленности матрицы A=\n" +

"Норма-1: {0}",

FirstNorm(A) \* FirstNorm(A\_r)));

AppendFile(String.Format("\n" +

"Норма-2: {0}",

SecondNorm(A) \* SecondNorm(A\_r)));

AppendFile(String.Format("\n" +

"Норма-3: {0}",

ThirdNorm(A) \* ThirdNorm(A\_r)));

//A\*x-b

AppendFile("\n\n\nA\*x-b:");

List<double> A\_x = new List<double>(), A\_x\_minus\_b;

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

{

A\_x.Add(0);

}

GetMultiVM(A, x, A\_x);

A\_x\_minus\_b = VectorDiferc(A\_x, b);

WriteVectorform16(A\_x\_minus\_b);

AppendFile("\n\n");

}

}

}