Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

**РЕШЕНИЕ ЗАДАЧ ЛИНЕЙНОЙ АЛГЕБРЫ**

**Отчет по лабораторной работе №2**

**По дисциплине**

**«Численные методы»**

Студент гр. 430-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Лузинсан

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Проверил: ст. преп. каф. АСУ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Е. Косова

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Томск 2022

Оглавление

[1 Цели и задачи 3](#_Toc99941421)

[1.1 Формат входных данных 3](#_Toc99941422)

[1.2 Формат выходных данных 3](#_Toc99941423)

[2 Теория 4](#_Toc99941424)

[2.1 Точные методы 6](#_Toc99941425)

[2.1.1 Метод Гаусса 6](#_Toc99941426)

[2.1.2 Метод декомпозиции 7](#_Toc99941427)

[2.2 Итерационные методы 8](#_Toc99941428)

[2.2.1 Метод простой итерации 8](#_Toc99941429)

[2.2.2 Метод Зейделя 9](#_Toc99941430)

[2.3 Вычисление обратных матриц 9](#_Toc99941431)

[3 Ход выполнения лабораторной работы 11](#_Toc99941432)

[4 Тестирование 12](#_Toc99941433)

[4.1 Unit 1 12](#_Toc99941434)

[4.2 Unit 2 12](#_Toc99941435)

[Вывод 14](#_Toc99941436)

[Список использованных источников 15](#_Toc99941437)

[**Листинг программы** 16](#_Toc99941438)

# Цели и задачи

В данной лабораторной работе необходимо реализовать один из трех обязательных точных методов (в зависимости от номера варианта):

1. Метод Гаусса;

2. Метод декомпозиции;

3. Метод ортогонализации.

Дополнительно можно реализовать еще один итерационный метод – Зейделя или простой итерации.

При помощи данных методов необходимо реализовать решение следующих задач:

1. Решение СЛАУ.

2. Поиск определителя матрицы (только для методов Гаусса и декомпозиции).

3. Поиск обратной матрицы.

Вариант – 2.

## 1.1 Формат входных данных

|  |  |  |
| --- | --- | --- |
| m | – тип задачи (в том порядке, в котором они перечислены выше); | |
| n | – порядок матрицы; | |
| a11…a1n [b1]  a21…a2n [b2]  …………..  an1…ann [bn] | | – коэффициенты матрицы и вектор свободных коэффициентов (при решении СЛАУ, т.е. при m = 1). |

## 1.2 Формат выходных данных

При решении СЛАУ на экран выводятся:

|  |  |
| --- | --- |
| x\* | – вектор решения; |
| ε | – вектор невязки; |

При поиске определителя – его значение. При вычислении обратной матрицы:

|  |  |
| --- | --- |
| X | – обратная матрица; |

Формат выходного файла зависит от метода и типа задачи:

• Если используется метод Гаусса, то в любом случае в выходной файл выводятся матрицы A(1), A(2) , …, A(n) . Если решалась система СЛАУ, то еще и вектора b(1), b(2) , …, b(n) . Если вычислялась обратная матрица – вектора e1(n), e2(n), …, en(n).

• Если используется метод декомпозиции, то в любом случае выводятся матрицы B и C. Если решалась система СЛАУ, то вектор y. Если вычислялась обратная матрица – вектора y1, y2, …, yn.

Для итерационных методов выводятся матрицы α и вектора β (для каждой решаемой СЛАУ).

# Теория

К решению задач линейной алгебры в численных методах относят решение систем линейных алгебраических уравнений и вычисление различных характеристик матриц – определителей, вычисление собственных чисел и собственных векторов.

Все перечисленные характеристики матриц, так или иначе, находятся при помощи решения некоторых СЛАУ. СЛАУ выглядит следующим образом:

|  |  |
| --- | --- |
|  | (2.1) |

где A – матрица размером n×m, x – вектор неизвестных длиной m, b – вектор свободных коэффициентов длиной n. Все вектора являются столбцами.

Если n < m, то СЛАУ называется недоопределенной, а если n > m – то переопределенной. Мы будем рассматривать только нормально определенные системы с n = m (т.е. имеющие квадратную матрицу A).

Точность решения СЛАУ можно оценить, вычислив вектор невязки:

|  |  |
| --- | --- |
|  | (2.2) |
|  | (2.2.1) |

где x\* – приближенное решение СЛАУ.

Для получения скалярной оценки можно использовать норму (2.2.1).

Учитывая, что точное решение уравнения (2.1) для квадратной матрицы можно найти аналитически, т.е с помощью формулы 2.3:

|  |  |
| --- | --- |
|  | (2.3) |

можно сделать вывод, что единственное решение существует только тогда, когда существует обратная матрица. А для этого, в свою очередь, требуется, чтобы выполнялось условие 2.4:

|  |  |
| --- | --- |
|  | (2.4) |

Существует три класса методов решения СЛАУ:

1. Прямые (точные). Дают решение задачи за конечное число итераций, при этом, если все операции выполняются точно, то и решение получается точным. При реализации на ЭВМ погрешность, конечно же, появляется. К прямым методам относятся методы Гаусса, декомпозиции (Халецкого), ортогонализации и др. Прямые методы применяются для решения систем порядка 103.
2. Итерационные. Дают решение с некоторой точностью как предел последовательных приближений. К итерационным методам относятся методы релаксации, простой итерации, Зейделя, градиентные методы и др. Итерационные методы применяются для систем порядка 107.
3. Вероятностные. Основаны на случайных испытаниях некоторой блуждающей частицы, моделирующей решение задачи и применении закона больших чисел. В основном, это метод Монте-Карло и его модификации.

## Точные методы

### Метод Гаусса

Метод Гаусса заключается в последовательном исключении неизвестных посредством преобразования исходной матрицы к системе с треугольной матрицей на этапе прямого хода, из которой затем последовательно обратным ходом получаются значения всех неизвестных.

Прямой ход состоит в преобразовании коэффициентов матрицы по формулам:

|  |  |
| --- | --- |
|  | (2.5) |
|  | (2.6) |

После прямого хода СЛАУ примет следующий вид:

|  |  |
| --- | --- |
|  | (2.7) |

Формулы для обратного хода метода Гаусса:

|  |  |
| --- | --- |
|  | (2.8) |

Определитель исходной матрицы A можно вычислить по формуле:

|  |  |
| --- | --- |
|  | (2.9) |

Метод Гаусса обладает следующим недостатком. Если обратить внимание на формулу (2.5), то видно, что в ней происходит операция деления на диагональные элементы матриц A(k) . Если в процессе решения требуемый диагональный элемент получится равным нулю, то этот метод даст сбой, даже если условие (2.4) выполняется. В этом случае требуется перестановка строк исходной матрицы A (и соответствующих элементов вектора b). В данной практической работе делать этого не требуется, т.к. алгоритм значительно усложняется (учитывая количество вариантов перестановки).

### Метод декомпозиции

Сначала исходная матрица A раскладывается на две треугольные матрицы B и C таким образом, что A = BC. Формулы для получения элементов матриц B и C:

|  |  |
| --- | --- |
|  | (2.10) |
|  | (2.11) |

Диагональные элементы матрицы C равны 1, остальные элементы матриц B и C нулевые:

Важен порядок вычисления элементов матриц B и C. Сначала вычисляется первый столбец матрицы B, затем первая строка матрицы C, затем второй столбец B, затем вторая строка C и т.д.

После этого сначала решается СЛАУ By = d, а затем – СЛАУ Cx = y по формулам (2.1.3 и 2.1.4):

|  |  |
| --- | --- |
|  | (2.12) |
|  | (2.13) |

Определитель исходной матрицы A можно вычислить по формуле 2.1.5:

|  |  |
| --- | --- |
|  | (2.14) |

Метод декомпозиции обладает тем же недостатком, что и метод Гаусса. В формуле (2.11) происходит деление на диагональные элементы матрицы B. Если в процессе решения требуемый диагональный элемент получится равным нулю, то этот метод также даст сбой. Аналогично, тогда может помочь только перестановка строк исходной СЛАУ, но делать этого, в рамках данной практической работы, мы не будем.

## Итерационные методы

### Метод простой итерации

Преобразуем исходную систему к виду:

|  |  |
| --- | --- |
|  | (2.15) |

где α – матрица размера nn, β – вектор размера n:

|  |  |
| --- | --- |
|  | (2.16) |

Полагая в качестве начального приближения решения x(0) = β, строим итерационный процесс по формуле 2.17:

|  |  |
| --- | --- |
|  | (2.17) |

Итерации заканчиваются, когда выполняется условие 2.18:

|  |  |
| --- | --- |
|  | (2.18) |

где ε – требуемая точность решения. Из (2.16) следует, что диагональные элементы исходной матрицы должны быть ненулевыми. Более того, на самом деле требования к ним еще жестче. Итерационный процесс (2.17) сходится, если норма матрицы α меньше 1. Для этого требуется, чтобы у исходной матрицы СЛАУ A числа, стоящие на главной диагонали, были больше суммы остальных чисел в соответствующей строке матрицы (все числа нужно брать по модулю), как показано в условии 2.19:

|  |  |
| --- | --- |
|  | (2.19) |

### Метод Зейделя

Метод Зейделя является модификацией метода простой итерации. Поэтому преобразование (2.15), (2.16), а также критерий останова (2.18) верны и для него. Несколько по-другому строится итерационный процесс (2.20):

|  |  |
| --- | --- |
|  | (2.20) |

Ограничение (2.19) также применимо. Но, в силу модификаций, метод Зейделя сходится также для любой СЛАУ с симметричной положительно определенной матрицей. Чтобы сделать матрицу таковой, необходимо ее транспонировать и умножить на саму себя. Тогда аналогичные преобразования необходимо проделать и с правой частью СЛАУ (2.21):

|  |  |
| --- | --- |
|  | (2.21) |

Получаем систему A'x = b', которую решаем методом Зейделя.

## Вычисление обратных матриц

Обозначим как X неизвестные элементы обратной матрицы. Следовательно, нам необходимо решить систему 2.22:

|  |  |
| --- | --- |
|  | (2.22) |

где E – единичная матрица:

Все матрицы имеют размер n×n. Решение матричной системы (2.22) можно представить в виде решения n СЛАУ (2.23):

|  |  |
| --- | --- |
|  | (2.23) |

где xi, ei – i-й столбец обратной и единичной матрицы соответственно.

Обратите внимание, что при вычислении обратной матрицы СЛАУ решается n раз, где n – порядок матрицы. При этом все треугольные матрицы в методах Гаусса и декомпозиции получаются одинаковыми, меняется только вектор свободных коэффициентов. Это нужно использовать для оптимизации вычислений в программе – все треугольные матрицы должны вычисляться только один раз.

# Ход выполнения лабораторной работы

Реализация методов содержится в классе InputData. В этом классе осуществляется как считывание данных из файла, так и непосредственное нахождение результирующего вектора решения системы линейных алгебраических уравнений.

Класс содержит методы Гаусса, декомпозиции, простой итерации, а также Зейделя. А также при вычислении решения отдельным методом сразу находится определитель и записывается в поле объекта.

Реализация каждого численного метода сделана в соответствии с теоретическим материалом методических указаний дисциплины.

# Тестирование

## Unit 1

Эффективность методов на примере решения СЛАУ порядка 3 представлены в виде таблицы 4.1.

Таблица 4.1 – Решение СЛАУ третьего порядка:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Метод | F(x) | x\* |  |  | | ||A|| | X-1 | | AX-E | ||AX-E|| | |
| Гаусса |  |  |  | 0 | | -24 |  | |  | 3.34e-16 | |
| Декомпозиции |  |  | 0 | | -24 |  | |  | 3.34e-16 | |
| Простой итерации | Не сходится | - | | | | | | | | |
| Зейделя |  |  | | 1.6e-11 | |  |  |  | | 4.4772e-10 |

## Unit 2

Эффективность методов на примере решения СЛАУ порядка 4 представлены в виде таблицы 4.2.

Таблица 4.2 – Решение СЛАУ четвёртого порядка:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | F(x) | x\* |  |  | ||A|| |
| Гаусса |  |  |  | 8.89911e-16 | 7.22553e+06 |
| Декомпозиции |  |  | 8.8991e-16 | 7.2255e+06 |
| Простой итерации |  |  | 1.98488e-05 | - |
| Зейделя |  |  | 1.67263e-13 | - |

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | X-1 | AX-E | ||AX-E|| |
| Гаусса |  |  | 1.3411e-16 |
| Декомпозиции |  |  | 1.3194e-16 |
| Простой итерации |  |  | 1.5232e-05 |
| Зейделя |  |  | 1.5507e-12 |

# Вывод

В ходе выполнения данной лабораторной работы я изучила теоретические сведения, связанные с численными методами решения систем линейных алгебраических уравнений.

Рассмотрела сильные и слабые стороны методов.

# Список использованных источников

1. Вычислительные методы : учебное пособие / А.А.Мицель. — Томск : Эль Контент, 2013. — 198 с. (дата обращения: 01.03.2022).

**Листинг программы**

#ifndef INPUTDATA\_H

#define INPUTDATA\_H

#include <iostream>

#include <fstream>

#include <cmath>

#include "Matrix.h"

#include "Vector.h"

#define MAX\_ITER 10000

#define EPS 1e-12

namespace luMath

{

double invert\_unit\_matrix\_initer(size\_t m, size\_t n, size\_t r, size\_t c);

double unit\_matrix\_initer(size\_t m, size\_t n, size\_t r, size\_t c);

double zero\_matrix\_initer(size\_t m, size\_t n, size\_t r, size\_t c);

char getSymbol(std::initializer\_list<char> list,

std::string notification\_message = "",

std::string error\_message = "Недопустимое значение, попробуйте ещё раз.\n->");

double getDouble(double min = -DBL\_MAX,

double max = DBL\_MAX,

std::string notification\_message = "",

std::string error\_message = "Недопустимое значение, попробуйте ещё раз.\n->");

template <class T>

class InputData

{

public:

enum class METHOD

{

GAUSS=1,

DECOMPOSOTION,

SIMPLEITERATION,

SEIDEL

};

enum class TASK

{

ROOT=1,

DETERMINANT,

INVERS

};

private:

std::ofstream\* \_fout;

// основные данные

Matrix<T>\* \_expandedMatrix;

Matrix<T>\* \_inverseMatrix;

Matrix<T>\* A;

Vector<T>\* b;

Vector<T>\* x;

int m; // размерность квадратной матрицы

// проверка и точность

Vector<T>\* ResidualVector;

Matrix<T>\* ResidualMatrix;

METHOD \_method;

TASK \_task;

T \_determinant = 0;

int \_NAfterComma;

public:

InputData()

{

std::ifstream\* \_fin = new std::ifstream("input.txt");

\_fout = new std::ofstream("output.txt");

int c;

\*\_fin >> c; // считывается метод

\_method = static\_cast<METHOD>(c);

\*\_fin >> c; // считывается тип задачи

\_task = static\_cast<TASK>(c);

\*\_fin >> m;

std::cout << "\n\tПорядок матрицы: " << m;

T\* array = new T[m \* (m + 1)];

for (int i = 0; i < (m + 1) \* m; i++)

\*\_fin >> array[i];

delete \_fin;

\_expandedMatrix = new Matrix<T>(m, m + 1, array);

\_inverseMatrix = new Matrix<T>(m);

delete[] array;

std::cout << "\n\tКоэффициенты считанной матрицы и вектор свободных коэффициентов (последний стоблик):\n\n" << \*\_expandedMatrix;

setA(\*\_expandedMatrix, m);

setB(\*\_expandedMatrix, m);

x = new Vector<T>(m);

(\*x).transposition();

ResidualVector = new Vector<T>(m);

(\*ResidualVector).transposition();

ResidualMatrix = new Matrix<T>(m);

}

InputData(const Matrix<T>& matrix, std::ofstream\* out)

{

m = matrix.getRows();

A = new Matrix<T>(m);

\*A = matrix;

\_inverseMatrix = new Matrix<T>(m);

ResidualMatrix = new Matrix<T>(m);

\_fout = out;

}

METHOD getMethod() { return \_method; }

TASK getTask() { return \_task; }

// инициализация квадратной матрицы

void setA(const Matrix<T>& matrix, size\_t size)

{

A = new Matrix<T>(size);

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

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

(\*A)[i][j] = matrix[i][j];

}

// инициализация вектора свободных коэффициентов - initialization of the vector of free coefficients

void setB(const Matrix<T>& matrix, size\_t size) // size - number of rows

{

b = new Vector<T>(size);

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

(\*b)[i] = matrix[i][size];

(\*b).transposition();

}

~InputData()

{

delete \_expandedMatrix;

delete \_inverseMatrix;

delete A;

delete x;

delete b;

delete ResidualVector;

delete ResidualMatrix;

delete \_fout;

}

Matrix<T>& getExpandedMatrix() { return \*\_expandedMatrix; }

//вектор невязки

void setResidualVector(const Matrix<T>& \_A, const Vector<T>& \_x, const Vector<T>& \_b)

{

\*\_fout << "\nВектор невязки e\*:\n" << (\_A \* \_x) - \_b;

(\*ResidualVector) = (\_A \* \_x) - \_b;

}

const Matrix<T>& getInverseMatrix() const { return \*\_inverseMatrix; }

const Matrix<T>& getMainMatrix() const { return \*A; }

void setInverseMatrixByMethod(Vector<T>(\*Method)(const Matrix<T>&, const Vector<T>&, T& determinant, std::ofstream&))

{

Vector<Vector<T>> x\_temp(m);

Vector<Vector<T>> E(m);

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

{

E[i] = Vector<T>(m);

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

if (i == j)

E[i][j] = 1;

else

E[i][j] = 0;

E[i].transposition();

}

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

{

x\_temp[i] = Method(\*A, E[i], \_determinant,\*\_fout);

std::cout << "\nx'[" << i << "]=\n" << x\_temp[i];

}

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

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

(\*\_inverseMatrix)[i][j] = x\_temp[j][i];

\*\_fout << "\nОбратная матрица:\n" << std::setw(10) << (\*\_inverseMatrix);

\*ResidualMatrix = (\*\_inverseMatrix) \* (\*A) - Matrix<double>(m, unit\_matrix\_initer);

\*\_fout << "\nМатрица невязки:\n" << std::setprecision(5) << std::setw(15) << \*ResidualMatrix;

\*\_fout << "\nЕвклидова норма матрицы невязки: " << (\*ResidualMatrix).getModule() << "\n";

}

void setRoot(Vector<T>(\*Method)(const Matrix<T>&, const Vector<T>&, T& determinant, std::ofstream&))

{

(\*x) = Method(\*A, \*b, \_determinant, \*\_fout);

}

void getRoot(Vector<T>(\*Method)(const Matrix<T>&, const Vector<T>&, T& determinant, std::ofstream&))

{

if (Method == InputData::GaussMethod)

\*\_fout << "\nМетод Гаусса:\n";

else if (Method == InputData::DecompositionMethod)

\*\_fout << "\nМетод Декомпозиции:\n";

else if (Method == InputData::SimpleIterationMethod)

\*\_fout << "\nМетод Простой Итерации\n";

else if (Method == InputData::SeidelMethod)

\*\_fout << "\nМетод Зейделя\n";

setRoot(Method);

\*\_fout << "\nРезультат:\n" << (\*x);

setResidualVector(\*A, \*x, \*b);

\*\_fout << "\nЕвклидова норма вектора невязки: " << (\*ResidualVector).getModule() << "\n";

}

T getDeterminant()

{

if (\_determinant == 0)

{

switch (\_method)

{

case METHOD::GAUSS:

setRoot(InputData::GaussMethod);

break;

case METHOD::DECOMPOSOTION:

setRoot(InputData::DecompositionMethod);

break;

case METHOD::SIMPLEITERATION:

setRoot(InputData::SimpleIterationMethod);

break;

}

}

\*\_fout << "\nОпределитель: " << \_determinant << "\n";

return \_determinant;

}

static Vector<T> GaussMethod(const Matrix<T>& A, const Vector<T>& b, T& determinant, std::ofstream& out = std::cout)

{

Matrix<T> expandedMatrix(A.getRows(), A.getCols() + 1);

for (int i = 0; i < expandedMatrix.getRows(); i++)

for (int j = 0; j < expandedMatrix.getCols(); j++)

if (j == expandedMatrix.getCols() - 1)

expandedMatrix[i][j] = b[i];

else

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

return GaussMethod(expandedMatrix, determinant,out);

}

static Vector<T> GaussMethod(const Matrix<T>& expandedMatrix, T& determinant, std::ofstream& out=std::cout)

{

// удобнее использовать расширенную матрицу, поэтому зафиксируем её

Matrix<T> tempMatrix(expandedMatrix);

determinant = 1;

// Прямой ход метода Гаусса - преобразование матрицы к треугольному виду

for (int i = 0; i < tempMatrix.getRows(); i++) // проходим по всем строкам

{

out << "i = "<< i << "\n" << std::setw(10) << tempMatrix << "\n";

T coeff = tempMatrix[i][i]; // запоминаем коэффициент по диагонали

determinant \*= coeff;

for (int j = i; j < tempMatrix.getRows() + 1; j++) // проходим по всем элементам текущей строки, включая вектор коэффициентов

{

// если это элемент на диагонали, то он вырождается в единицу,

// а если любой другой на текущей строке, то просто делится на этот коэффициент

tempMatrix[i][j] /= coeff;

}

out << '\n' << std::setw(10) << tempMatrix;

for (int j = i + 1; j < tempMatrix.getRows(); j++)

{

coeff = tempMatrix[j][i]; // запоминаем коэффициент умножения

for (int k = i; k < tempMatrix.getCols(); k++) // проходим по всем элементам строки, некоторые элементы которой будут обнуляться

tempMatrix[j][k] -= coeff \* tempMatrix[i][k]; // вычитаем из текущей строки верхнюю i-ю строку помноженную на coeff

// в результате получим некоторое количество нулей под единицей

}

}

Vector<T> result(tempMatrix.getRows());

result.transposition();

// Обратный ход метода Гаусса

for (int i = result.getLength() - 1; i >= 0; i--)

{

T sumCoeff = 0;

for (int j = i + 1; j < result.getLength(); j++)

sumCoeff += tempMatrix[i][j] \* result[j];

result[i] = tempMatrix[i][result.getLength()] - sumCoeff;

}

return result;

}

static Vector<T> DecompositionMethod(const Matrix<T>& A, const Vector<T>& b, T& determinant, std::ofstream& out = std::cout)

{

// Раскладываем матрицу \_matrix на матрицы B и C так, что A = B \* C

Matrix<T> B(A.getRows(), unit\_matrix\_initer);

Matrix<T> C(A.getRows(), unit\_matrix\_initer);

unsigned m = B.getRows();

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

{

for (int i = j; i < m; i++)// проходим по элементам столбца матрицы B

{

T sumCoeff = 0;

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

sumCoeff += B[i][k] \* C[k][j];

B[i][j] = A[i][j] - sumCoeff;

}

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

{

T sumCoeff = 0;

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

sumCoeff += B[j][k] \* C[k][i];

C[j][i] = (1 / B[j][j]) \* (A[j][i] - sumCoeff);

}

out << "\n\ti = "<< j<<"\n\tМатрица B : \n" << std::setprecision(5) << std::setw(15) << B << "\n\tМатрица C : \n" << std::setw(15) << C;

}

Vector<T> y(m);

determinant = 1;

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

{

T sumCoeff = 0;

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

sumCoeff += B[i][k] \* y[k];

y[i] = (b[i] - sumCoeff) / B[i][i];

determinant \*= B[i][i];

}

out << "\ny' = \n" <<y.transposition() << "\n";

Vector<T> x(m);

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

{

T sumCoeff = 0;

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

sumCoeff += C[i][k] \* x[k];

x[i] = y[i] - sumCoeff;

}

x.transposition();

return x;

}

static bool converge(const Matrix<T>& A, std::ofstream& out = std::cout)

{

int n = A.getCols();

double sum = 0;

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

{

sum = 0;

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

if(i!=j)

sum += A[i][j];

if (abs(A[i][i]) <= abs(sum))

{

out << "\n\tНеобходимость не выполняется\n";

return false;

}

}

return true;

}

static Vector<T> SimpleIterationMethod(const Matrix<T>& A, const Vector<T>& B, T& determinant, std::ofstream& out = std::cout)

{

int m = A.getRows();

Matrix<double> a(m);

Vector<double> b(m);

b.transposition();

if (!converge(A,out))

return B;

Vector<double> x0(b);

x0.transposition();

Vector<double> x1(b);

x1.transposition();

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

{

b[i] = B[i] / A[i][i];

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

if (i == j)

a[i][j] = 0;

else

a[i][j] = -A[i][j] / A[i][i];

}

out << "\nb=\n" << std::setw(10) << b << "\na=\n" << std::setw(10) << a;

x0 = b;

x1 = x0;

do

{

x0 = x1;

x1 = b + a \* x1;

} while ((x1-x0).getModule() >= abs( (1-a.getModule()) / a.getModule() \* 0.000001) );

return x1;

}

static Vector<T> SeidelMethod(const Matrix<T>& A, const Vector<T>& B, T& determinant, std::ofstream& out = std::cout)

{

int m = A.getRows();

Matrix<T> a0(A);

Vector<T> b0(B);

if (!converge(A, out))

{

out << "\nПреобразуем матрицу к симметрично положительно определённой.";

Matrix<T> a\_T(a0.transposition());

out << "\na^T:\n" << a\_T;

a0 = a\_T \* a0;

out << "\n a^T \* a = \n" << a0;

b0 = a\_T \* b0;

out << "\n a^T \* b = \n" << b0;

}

Vector<double> x0(b0);

x0.transposition();

Vector<double> x1(b0);

x1.transposition();

Matrix<T> a(A);

Vector<T> b(B);

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

{

b[i] = b0[i] / a0[i][i];

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

if (i == j)

a[i][j] = 0;

else

a[i][j] = -a0[i][j] / a0[i][i];

}

out << "\nb=\n" << std::setw(10) << b << "\na=\n" << std::setw(10) << a;

x0 = b;

x1 = x0;

unsigned count = 0;

do

{

x0 = x1;

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

{

double sum1 = 0;

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

sum1 += a[i][j] \* x1[j];

double sum2 = 0;

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

sum2 += a[i][j] \* x0[j];

x1[i] = b[i] + sum1 + sum2;

}

count++;

} while ((x1 - x0).getModule() >= abs((1 - a.getModule()) / a.getModule() \* EPS) && count < MAX\_ITER);

return x1;

}

};

}

#endi