

Министерство науки и высшего образования РФ  
Федеральное государственное автономное образовательное учреждение  
высшего образования «Национальный исследовательский Нижегородский  
государственный университет им. Н.И. Лобачевского»  
Институт информационных технологий, математики и механики

# **Отчет по лабораторной работе**

## **РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ**

## **АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ**

Выполнил: Власов Максим Сергеевич, студент группы 381806-1

Проверил: к. ф.-м. н., доцент кафедры ДУМЧА Эгамов А. И.

Нижний Новгород

2020

# Содержание

<b>ВВЕДЕНИЕ.....</b>	<b>3</b>
<b>1. ПОСТАНОВКА ЗАДАЧИ .....</b>	<b>4</b>
<b>2. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ .....</b>	<b>5</b>
<b>3. РУКОВОДСТВО ПРОГРАММИСТА.....</b>	<b>8</b>
3.1. СТРУКТУРА ПРОГРАММЫ .....	8
3.2. ОПИСАНИЕ АЛГОРИТМОВ .....	10
3.2.1. Метод Гаусса .....	10
3.2.2. Метод Крамера .....	11
3.2.3. Метод простых итераций .....	12
3.2.4. Метод Зейделя.....	13
3.2.5. Метод верхней релаксации.....	14
3.2.6. Последовательный алгоритм .....	15
3.2.7. LU - разложение.....	16
3.3. ОПИСАНИЕ СТРУКТУР ДАННЫХ И ФУНКЦИЙ .....	18
3.3.1. Класс <i>MainWindow</i> .....	18
3.3.2. Класс <i>LESystemSolver</i> .....	20
3.3.3. Классы-наследники <i>LESystemSolver</i> .....	21
3.3.4. Класс <i>DataRequestDialog</i> .....	22
3.3.5. Класс <i>HelpDialog</i> .....	23
3.3.6. Класс <i>AboutDialog</i> .....	23
3.3.7. Основная программа .....	23
<b>ЗАКЛЮЧЕНИЕ .....</b>	<b>24</b>
<b>СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ.....</b>	<b>25</b>

# Введение

В связи с развитием новой вычислительной техники инженерная практика наших дней все чаще и чаще встречается с математическими задачами, точное решение которых получить весьма сложно или невозможно. В этих случаях обычно прибегают к тем или иным приближенным вычислениям. Вот почему приближенные и численные методы математического анализа получили за последние годы широкое развитие и приобрели исключительно важное значение.

Данная работа направлена на изучение методов решения систем линейных алгебраических уравнений различными способами, как точечными, так и итерационными.

# 1. Постановка задачи

**Задача:** разработать и реализовать приложение, выполняющее решение систем линейных алгебраических уравнений (далее – СЛАУ) различными способами (метод Гаусса, метод Крамера, метод Зейделя, метод простых итераций, метод верхней релаксации, метод LU-разложения) с измерением времени.

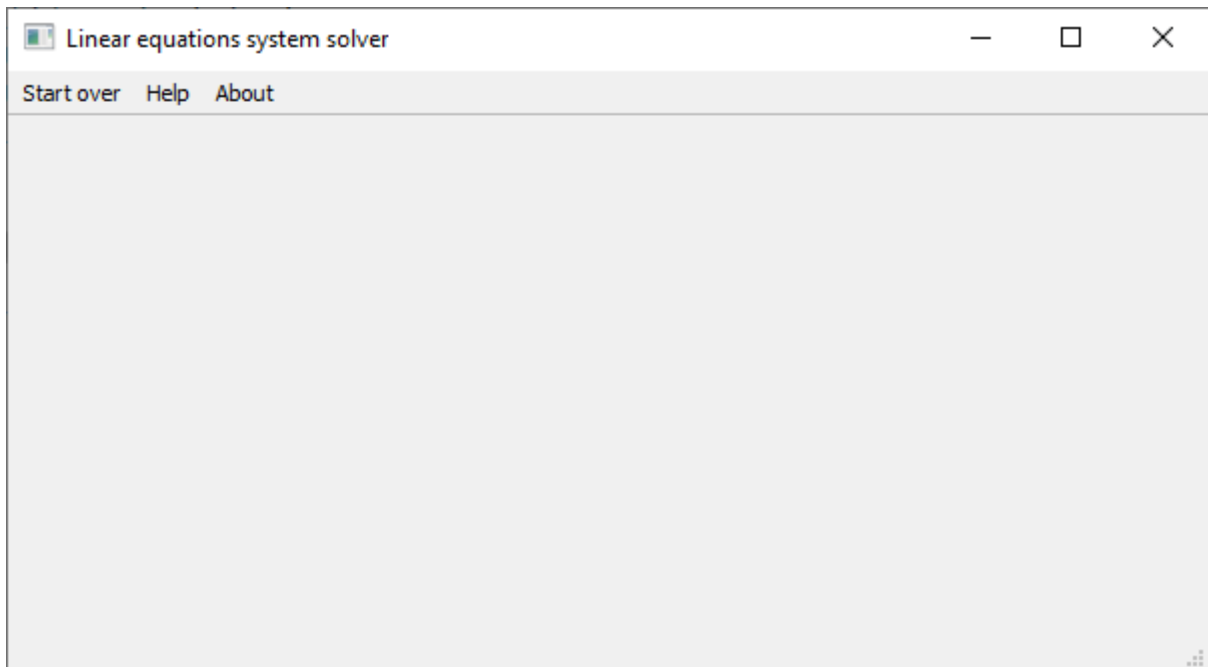
**Входные данные:** система линейных алгебраических уравнений, первое приближение решения (если необходимо).

**Выходные данные:** решение системы (если есть).

## 2. Руководство пользователя

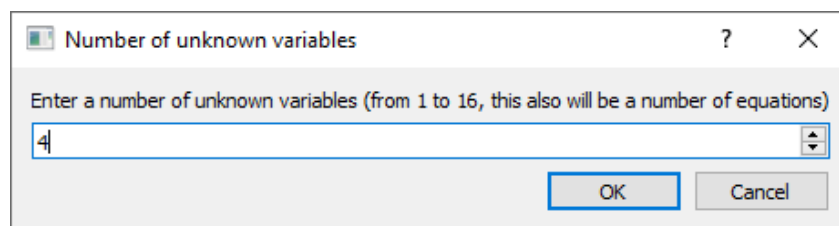
В данном руководстве содержатся пошаговые инструкции по работе с программой, для того чтобы вы могли как можно быстрее приступить к использованию приложения.

1. Запустите файл **02\_systems.exe** из папки с программой. Перед вами отобразится основное окно программы. Ознакомьтесь с пунктами меню “Help” и “About” в верхней части окна, чтобы узнать о возможностях и разработчиках программы.



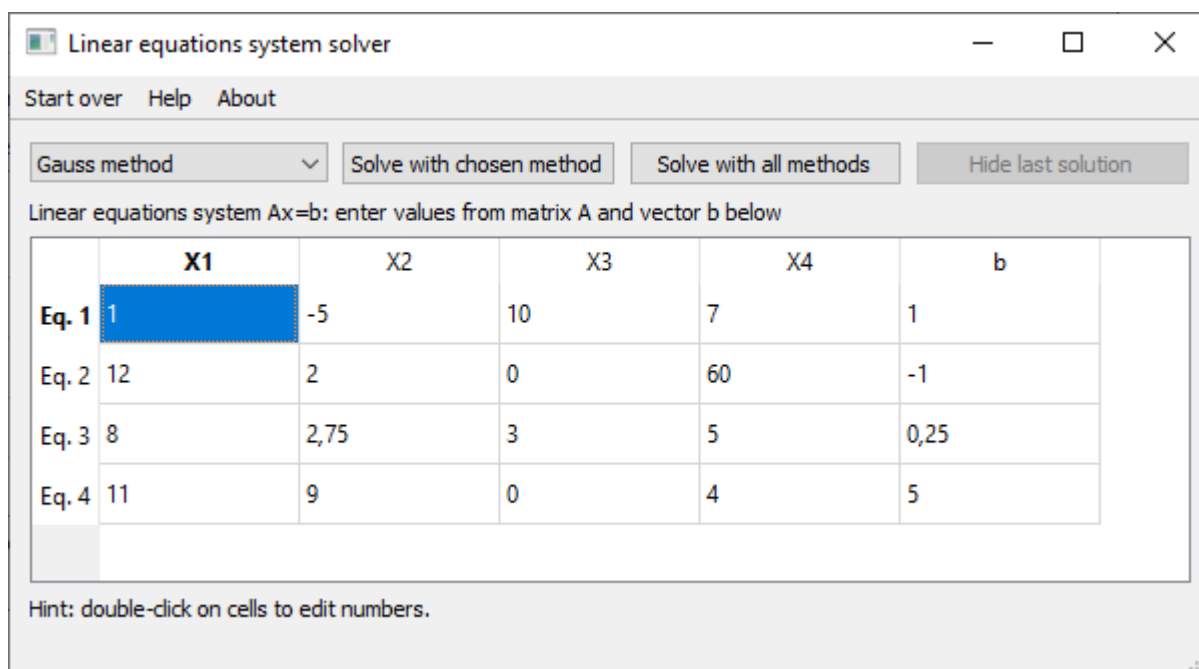
**Рис 1.** Программа после запуска.

2. Нажмите “Start over”, чтобы вызвать окно ввода числа неизвестных. Введите целое число от 2 до 16, а затем нажмите “OK”.



**Рис 2.** Окно ввода числа неизвестных.

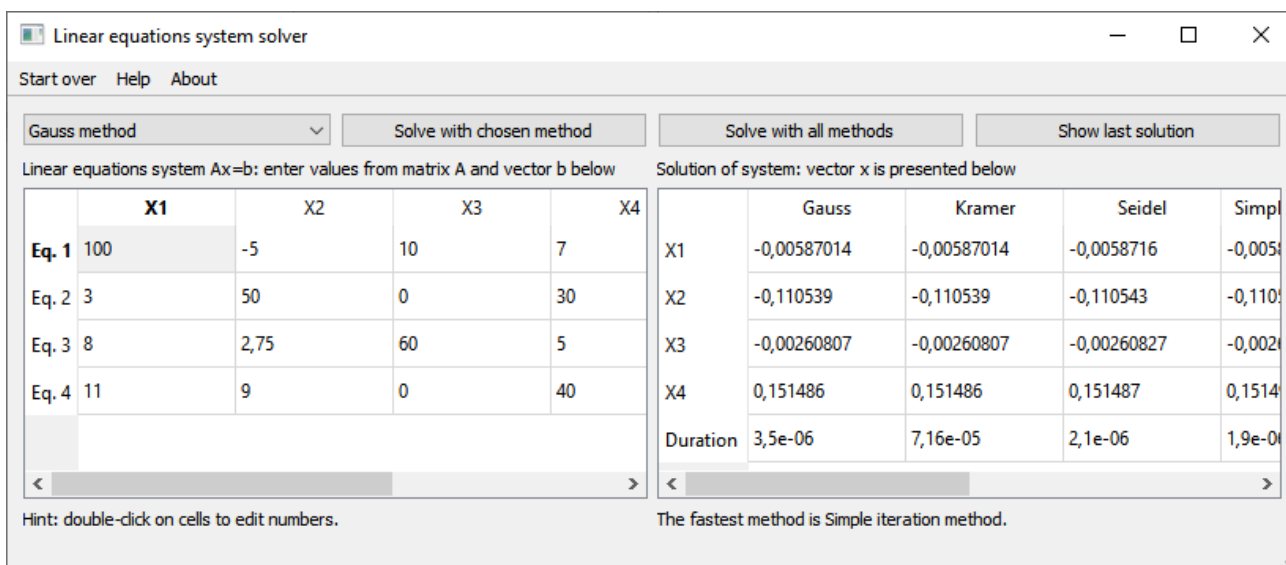
3. Перед вами отобразится таблица, представляющая собой матрицу A и столбец b в СЛАУ вида  $Ax=b$ . Заполните ее вещественными числами (для ввода дважды щелкните по нужной ячейке, для сохранения введенного числа щелкните в любом месте либо нажмите Enter). По таблице также можно перемещаться с помощью клавиш со стрелками.



	X1	X2	X3	X4	b
Eq. 1	1	-5	10	7	1
Eq. 2	12	2	0	60	-1
Eq. 3	8	2,75	3	5	0,25
Eq. 4	11	9	0	4	5

**Рис 3.** Пример введенной системы уравнений.

4. Чтобы решить СЛАУ конкретным методом, выберите его из выпадающего списка слева и нажмите “Solve with chosen method”. Чтобы решить СЛАУ всеми доступными методами, нажмите “Solve with all methods”.



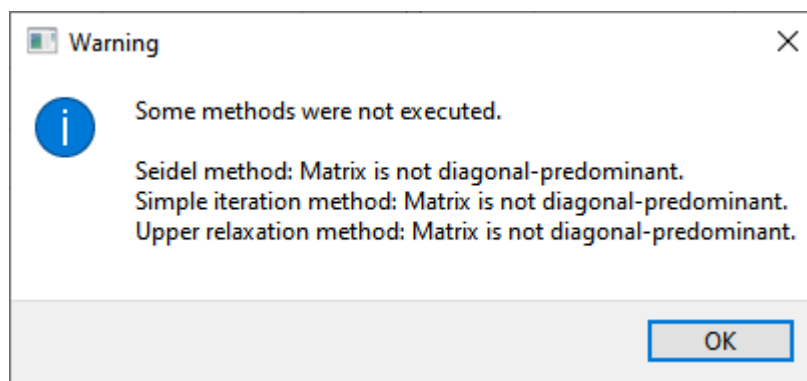
	X1	X2	X3	X4
Eq. 1	100	-5	10	7
Eq. 2	3	50	0	30
Eq. 3	8	2,75	60	5
Eq. 4	11	9	0	40

	Gauss	Kramer	Seidel	Simple iteration
X1	-0,00587014	-0,00587014	-0,0058716	-0,0058716
X2	-0,110539	-0,110539	-0,110543	-0,110543
X3	-0,00260807	-0,00260807	-0,00260827	-0,00260827
X4	0,151486	0,151486	0,151487	0,151487
Duration	3,5e-06	7,16e-05	2,1e-06	1,9e-06

**Рис 4.** Пример вычисленного решения.

5. Не все системы поддаются решению с помощью всех имеющихся методов, в случае возникновения ошибки вы увидите соответствующее сообщение.



**Рис 5.** Пример ошибки.

6. По завершении работы с программой нажмите крестик в правом верхнем углу окна.

## 3. Руководство программиста

### 3.1. Структура программы

Исходный код программы содержится в следующих модулях:

1. **Main** (включает в себя файл `main.cpp`) – основной модуль (точка входа, использующая модули ниже).
2. **MainWindow** (включает в себя файлы `mainwindow.h`, `mainwindow.cpp`, `mainwindow.ui`) – модуль Основное окно (содержит объявление и реализацию класса, отвечающего за основное окно программы).
3. **Matrix** (включает в себя файлы `matrix.h` и `matrix.cpp`) – модуль Матрица (содержит заданные типы для работы с матрицами и векторами, а также реализацию некоторых бинарных операций).
4. **LESystemSolver** (включает в себя файлы `lesystemsolver.h` и `lesystemsolver.cpp`) – модуль Решатель СЛАУ – абстрактный метод (содержит объявление и реализацию базового класса).
5. **GaussMethodSolver** (включает в себя файлы `gaussmethodsolver.h` и `gaussmethodsolver.cpp`) – модуль Решатель СЛАУ – метод Гаусса (класс, содержащий в себе реализацию алгоритма этого метода, а также вспомогательные функции).
6. **KramerMethodSolver** (включает в себя файлы `kramermethodsolver.h` и `kramermethodsolver.cpp`) – модуль Решатель СЛАУ – метод Крамера (класс, содержащий в себе реализацию алгоритма этого метода, а также вспомогательные функции).
7. **LUDecompositionMethodSolver** (включает в себя файлы `ludecompositionmethodsolver.h` и `ludecompositionmethodsolver.cpp`) – модуль Решатель СЛАУ – метод LU-разложения (класс, содержащий в себе реализацию алгоритма этого метода, а также вспомогательные функции).
8. **SeidelMethodSolver** (включает в себя файлы `seidelmethodsolver.h` и `seidelmethodsolver.cpp`) – модуль Решатель СЛАУ – метод Зейделя (класс, содержащий в себе реализацию алгоритма этого метода, а также вспомогательные функции).
9. **SimpleIterationMethodSolver** (включает в себя файлы `simpleiterationmethodsolver.h` и `simpleiterationmethodsolver.cpp`) – модуль Решатель СЛАУ – метод простых итераций (класс, содержащий в себе реализацию алгоритма этого метода, а также вспомогательные функции).



10. **UpperRelaxationMethodSolver** (включает в себя файлы upperrelaxationmethodsolver.h и upperrelaxationmethodsolver.cpp) – модуль Решатель СЛАУ – метод верхней релаксации (класс, содержащий в себе реализацию алгоритма этого метода, а также вспомогательные функции).
11. **DataRequestDialog** (включает в себя файлы datarequestdialog.h, datarequestdialog.cpp, datarequestdialog.ui) – модуль Окно запроса данных (содержит объявление и реализацию класса, отвечающего за окно запроса данных – первого приближения решения).
12. **HelpDialog** (включает в себя файлы helpdialog.h, helpdialog.cpp, helpdialog.ui) – модуль Окно справки (содержит объявление и реализацию класса, отвечающего за окно с информацией о работе с программой).
13. **AboutDialog** (включает в себя файлы aboutdialog.h, aboutdialog.cpp, aboutdialog.ui) – модуль Окно информации о программе (содержит объявление и реализацию класса, отвечающего за окно с информацией о создателях программы).

## 3.2. Описание алгоритмов

Методы численного решения систем линейных уравнений естественным образом распадаются на две группы:

- Точечные (прямые);
- Итерационные.

**Точные методы** представляют собой способы вычисления решений систем за конечное число шагов. Однако число операций в этих методах существенно зависит от размерности системы уравнений, причем данная зависимость не линейная.

**Итерационные методы** - это приближенные методы, позволяющие получать решения систем с помощью бесконечных сходящихся процессов.

Заметим, что вследствие неизбежности округлений при компьютерных вычислениях даже точные методы становятся приближенными.

В данной работе будут рассмотрены следующие методы решения СЛАУ:

- метод Гаусса,
- метод простых итераций,
- правило Крамера,
- метод релаксации,
- метод Зейделя,
- метод Якоби.

### 3.2.1. Метод Гаусса

Пусть дана система линейных уравнений

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2, \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n. \end{cases}$$

Общий подход к отысканию решений системы основывается на последовательном переходе с помощью элементарных преобразований от данной системы к такой эквивалентной ей системе, для которой решение находится просто. Такой подход, в частности, лежит в основе метода последовательного исключения неизвестных (метода Гаусса).

Алгоритм этого метода состоит в следующем.

Не уменьшая общности, будем считать, что коэффициент  $a_{11}$  системы отличен от нуля.

1. Умножим первое уравнение на  $a_{21}/a_{11}$  и вычтем результат из 2 уравнения.
  2. Умножим первое уравнение на  $a_{31}/a_{11}$  и вычтем результат из 3 уравнения.
  3. ...
  4. Умножим первое уравнение на  $a_{n1}/a_{11}$  и вычтем результат из последнего уравнения.
- Тогда система примет вид

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1, \\ a'_{22}x_2 + \dots + a'_{2n}x_n = b'_2, \\ \dots \\ a'_{n2}x_2 + \dots + a'_{nn}x_n = b'_n. \end{cases}$$

На этом первый шаг прямого хода Гаусса заканчивается. Следующий шаг осуществляется аналогично, считая, что на нем  $a'_{22} \neq 0$ . В результате неизвестное  $x_2$  исключается из всех уравнений, кроме первого и второго, и т.д.

Если на каком-то шаге прямого хода получается уравнение вида

$$0 \cdot x_1 + 0 \cdot x_1 + \dots + 0 \cdot x_1 = b_i \text{ при } b_i \neq 0.$$

Тогда рассматриваемая система несовместна, и дальнейшее ее решение прекращается.

После  $n$  шагов система принимает треугольный вид

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1, \\ a_{22}x_2 + \dots + a_{2n}x_n = b_2, \\ \dots \\ a_{nn}x_n = b_n. \end{cases}$$

Штрихи над коэффициентами были опущены для дальнейшего упрощения записи.

Далее идет обратный ход метода Гаусса.

1. Из последнего уравнения системы находим значение неизвестного  $x_n$ .
  2. Подставляем в предпоследнее уравнение системы  $x_n$  и вычисляем  $x_{n-1}$ .
  3. ...
  4. Подставляя в первое уравнение системы  $x_n, x_{n-1}, \dots, x_2$ , находим  $x_1$ .
- Таким образом, получаем решение системы линейных уравнений.

### 3.2.2. Метод Крамера

Пусть дана система линейных уравнений

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2, \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n. \end{cases}$$

Данная система называется **крамеровской**, если  $\Delta$  матрицы  $\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}$  отличен от нуля.

Такая система имеет единственное решение, и оно находится по формулам Крамера

$$x_j = \frac{\Delta_j}{\Delta}, j = 1, 2, \dots, n,$$

где определитель  $\Delta_j$  получается из определителя  $\Delta$  заменой  $j$ -ого столбца столбцом свободных членов системы.

### 3.2.3. Метод простых итераций

Пусть дана система линейных уравнений вида  $Ax = b$ , где

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}; x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}; b = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix};$$

и  $A$  – невырожденная  $n$  – ого порядка.

Предполагаем, что  $a_{ii} \neq 0, i = \overline{1, n}$ . Выразим  $x_1$  из первого уравнения,  $x_1$  из второго уравнения и т.д. Получаем систему

$$\begin{cases} x_1 = \frac{b_1}{a_{11}} - \frac{a_{12}}{a_{11}}x_2 - \cdots - \frac{a_{1n}}{a_{11}}x_n \\ \vdots \\ x_n = \frac{b_n}{a_{nn}} - \frac{a_{n1}}{a_{nn}}x_1 - \frac{a_{n2}}{a_{nn}}x_2 - \cdots - \frac{a_{n,n-1}}{a_{nn}}x_{n-1} \end{cases}$$

Обозначим:

$$\frac{b_i}{a_{ii}} = \beta_i; -\frac{a_{ij}}{a_{ii}} = \alpha_{ij}, i \neq j; \alpha_{ii} = 0; i = \overline{1, n}, j = \overline{1, n}$$

Получаем:

$$\alpha = \begin{pmatrix} \alpha_{11} & \cdots & \alpha_{1n} \\ \vdots & \ddots & \vdots \\ \alpha_{n1} & \cdots & \alpha_{nn} \end{pmatrix}; \beta = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix};$$

Если записать это в матричном виде получим:  $x^{(i)} = \beta + \alpha x^{(i-1)}$ .

За нулевое приближение примем столбец свободных членов.

$$\begin{pmatrix} x_1^{(0)} \\ \vdots \\ x_n^{(0)} \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix};$$

$$\begin{pmatrix} x_1^{(1)} \\ \vdots \\ x_n^{(1)} \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix} + \begin{pmatrix} \alpha_{11} & \cdots & \alpha_{1n} \\ \vdots & \ddots & \vdots \\ \alpha_{n1} & \cdots & \alpha_{nn} \end{pmatrix} \begin{pmatrix} x_1^{(0)} \\ \vdots \\ x_n^{(0)} \end{pmatrix} - \text{первое приближение};$$

$$\begin{pmatrix} x_1^{(2)} \\ \vdots \\ x_n^{(2)} \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix} + \begin{pmatrix} \alpha_{11} & \cdots & \alpha_{1n} \\ \vdots & \ddots & \vdots \\ \alpha_{n1} & \cdots & \alpha_{nn} \end{pmatrix} \begin{pmatrix} x_1^{(1)} \\ \vdots \\ x_n^{(1)} \end{pmatrix} - \text{второе приближение};$$

...

$$\begin{pmatrix} x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix} + \begin{pmatrix} \alpha_{11} & \cdots & \alpha_{1n} \\ \vdots & \ddots & \vdots \\ \alpha_{n1} & \cdots & \alpha_{nn} \end{pmatrix} \begin{pmatrix} x_1^{(i-1)} \\ \vdots \\ x_n^{(i-1)} \end{pmatrix} - i - \text{ое приближение};$$

Таким образом, получаем последовательность приближенных решений

$$x^{(0)}, x^{(1)}, \dots, x^{(k)}, \dots$$

Если эта последовательность имеет предел, то он является решением системы. Процесс быстро сходится, если диагональные коэффициенты исходной системы значительно преобладают по абсолютной величине над остальными коэффициентами. Более точно достаточные условия описывает следующая теорема.

**Теорема 1.** *Процесс итераций для системы линейных уравнений сходится к единственному ее решению, если какая-либо норма матрицы этой системы меньше единицы, в частности, если выполняется хотя бы одно из следующих условий:*

$$\|\alpha\|_1 = \max_j \sum_i |\alpha_{ij}| < 1,$$

$$\|\alpha\|_\infty = \max_i \sum_j |\alpha_{ij}| < 1,$$

$$\|\alpha\|_E = \sqrt{\sum_{i=1}^n \sum_{j=1}^n |\alpha_{ij}|^2} < 1.$$

### 3.2.4. Метод Зейделя

Другим итерационным методом решения систем линейных уравнений является метод Зейделя. Он представляет собой некоторое изменение метода простых итераций. В нем при вычислении  $(k + 1)$  – ого приближения неизвестной  $x_i$  используются уже вычисленные значения  $(k + 1)$  – ого приближения для неизвестных  $x_1, x_2, \dots, x_{i-1}$ . Если для приведенной системы

$$\begin{cases} x_1 = b_{11}x_1 + b_{12}x_2 + \cdots + b_{1n}x_n + h_1, \\ x_2 = b_{21}x_1 + b_{22}x_2 + \cdots + b_{2n}x_n + h_2, \\ \dots \\ x_n = b_{n1}x_1 + b_{n2}x_2 + \cdots + b_{nn}x_n + h_n. \end{cases}$$

уже найдено  $k$  – е приближение  $x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)}$ , то  $(k + 1)$  – е приближение находится по формулам

$$\begin{cases} x_1^{(k+1)} = b_{11}x_1^{(k)} + b_{12}x_2^{(k)} + \dots + b_{1n}x_n^{(k)} + h_1, \\ x_2^{(k+1)} = b_{21}x_1^{(k+1)} + b_{22}x_2^{(k)} + \dots + b_{2n}x_n^{(k)} + h_2, \\ x_3^{(k+1)} = b_{31}x_1^{(k+1)} + b_{32}x_2^{(k+1)} + \dots + b_{3n}x_n^{(k)} + h_3, \\ \dots \\ x_n^{(k+1)} = b_{n1}x_1^{(k+1)} + \dots + b_{n,n-1}x_{n-1}^{(k+1)} + b_{nn}x_n^{(k)} + h_n. \end{cases} \quad (1)$$

$$k = 0, 1, 2, \dots$$

Если матрицу  $B$  итерационного процесса (1) Зейделя представить в виде  $B = B_1 + B_2$ , где

$$B_1 = \begin{pmatrix} 0 & 0 & \dots & 0 \\ b_{21} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & 0 \end{pmatrix}, \quad B_2 = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ 0 & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & b_{nn} \end{pmatrix},$$

то систему (1) можно записать в матричной форме следующим образом:

$$x^{(k+1)} = B_1 x^{(k+1)} + B_2 x^{(k)} + h, \quad (2)$$

или, что тоже самое, в виде

$$(E - B_1)x^{(k+1)} = B_2 x^{(k)} + h. \quad (3)$$

Матрица  $E - B_1$  является левой нижней треугольной матрицей с единицами по главной диагонали. Поэтому она имеет обратную матрицу  $(E - B_1)^{-1}$ . Умножив слева обе части равенства (3) на матрицу  $(E - B_1)^{-1}$ , приведем к равенству

$$x^{(k+1)} = (E - B_1)^{-1} B_2 x^{(k)} + (E - B_1)^{-1} h. \quad (4)$$

Таким образом, итерационный процесс (1) Зейделя эквивалентен процессу (4) простой итерации. Для сходимости обоих этих матриц достаточно, что какая-либо норма матрицы  $(E - B_1)^{-1} B_2$  была меньше единицы.

Можно доказать, что достаточным условиям сходимости процесса Зейделя (4) являются те же условия, что и для сходимости процесса простой итерации.

### 3.2.5. Метод верхней релаксации

Рассмотрим систему  $Ax = b$  с симметричной, положительно определенной матрицей  $A$  размера  $n \times n$ . Обозначим через  $D$  диагональную матрицу  $n \times n$ , такую, что ее главная диагональ совпадает с главной диагональю матрицы  $A$ . Через  $L$  обозначим нижнюю треугольную матрицу  $n \times n$ , такую, что ее ненулевые (поддиагональные) элементы также

совпадают с элементами  $A$ , а главная диагональ является нулевой. Аналогично обозначим через  $R$  верхнюю треугольную матрицу  $n \times n$ , ненулевые (наддиагональные) элементы которой совпадают с элементами  $A$ , а главная диагональ также является нулевой. В этом случае для  $A$  справедливо представление в виде

$$A = L + D + R \quad (2)$$

Метод верхней релаксации является представителем стационарных одношаговых итерационных методов линейной алгебры и записывается в виде

$$\frac{(D + \omega L)(x^{(n+1)} + x^{(n)})}{\omega} + Ax^{(n)} = b$$

Здесь  $x^{(n)}$  – приближение, полученное на итерации с номером  $n$ ,  $x^{(n)}$  – следующее приближение,  $\omega$  – число (*параметр* метода), матрицы  $A, L, D$  и *вектор*  $b$  определены выше.

Необходимым условием сходимости метода релаксации с любого начального приближения  $x^0$  к точному решению задачи  $x^*$  является выполнение условия  $\omega \in (0,2)$ . Если же *матрица*  $A$  симметрична и положительно определена, то выполнение данного условия является также и достаточным. При этом если  $\omega \in (0,1)$ , то говорят о *методе нижней релаксации*, а при  $\omega \in (1,2)$  – о *методе верхней релаксации*, при  $\omega = 1$  метод релаксации будет совпадать с известным *методом Зейделя*.

Скорость сходимости метода верхней релаксации определяется выбором параметра  $\omega$ . Известно, что при решении некоторых классов разреженных систем уравнений, метод Зейделя требует  $O(n^2)$  итераций, а при надлежащем выборе итерационного параметра  $\omega$  метод будет сходиться за  $O(n)$  итераций.

В общем случае нет аналитической формулы для вычисления оптимального параметра  $\omega_{opt}$ , обеспечивающего наилучшую *сходимость*.

### 3.2.6. Последовательный алгоритм

Получим формулы для отыскания  $x^{(n+1)}$  по предыдущему приближению  $x^{(n)}$  в явном виде.

$$\begin{aligned} (D + \omega L)(x^{(n+1)} + x^{(n)}) + \omega Ax^{(n)} &= \omega b \\ Dx^{(n+1)} + \omega Lx^{(n+1)} - Dx^{(n)} - \omega Lx^{(n)} + \omega Ax^{(n)} &= \omega b \\ Dx^{(n+1)} &= -\omega Lx^{(n+1)} + Dx^{(n)} - \omega(A - L)x^{(n)} + \omega b \end{aligned}$$

С учетом того, что  $A - L = D + R$ , получаем

$$Dx^{(n+1)} = -\omega Lx^{(n+1)} + (1 - \omega)Dx^{(n)} - \omega Rx^{(n)} + \omega b$$

Далее нетрудно записать явные формулы для отыскания компонент нового вектора  $x^{(n+1)}$ :

$$a_{ii}x_i^{(n+1)} = -\omega \sum_{j=1}^{i-1} a_{ij}x_j^{(n+1)} + (1 - \omega)a_{ii}x_i^{(n)} - \omega \sum_{j=i+1}^N a_{ij}x_j^{(n)} + \omega b_i, \quad (3)$$

Как следует из формулы (3), при подсчете  $i$ -й компоненты нового приближения все компоненты, индекс которых меньше  $i$ , берутся из нового приближения  $x^{(n+1)}$ , а все компоненты, индекс которых больше либо равен  $i$  – из старого приближения  $x^{(n)}$ . Таким образом, после того, как  $i$ -я компонента нового приближения вычислена,  $i$ -я компонента старого приближения нигде использоваться не будет. Напротив, для подсчета следующих компонент вектора  $x^{(n+1)}$  компоненты с индексом, меньшим или равным  $i$ , будут использоваться "в новой версии". В силу этого обстоятельства для реализации метода достаточно хранить только одно (текущее) приближение  $x^{(n)}$ , а при расчете следующего приближения  $x^{(n+1)}$  использовать формулу (3) для всех компонент по порядку и постепенно обновлять вектор  $x^{(n)}$ .

### 3.2.7. LU - разложение

**LU – разложение** – представление матрицы  $A$  в виде произведения двух матриц,  $A = LU$  где  $L$  – нижняя треугольная матрица, а  $U$  – верхняя треугольная матрица.

$$L = \begin{pmatrix} l_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ l_{n1} & \cdots & l_{nn} \end{pmatrix}, U = \begin{pmatrix} u_{11} & \cdots & u_{1n} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & u_{nn} \end{pmatrix}.$$

Данное разложение используется для решения систем линейных уравнений, обращения матриц и вычисления определителя.

**Теорема 1.** Любую квадратную матрицу  $A = (a_{ij})$   $n$  – ого порядка, у которой все угловые диагональные миноры

$$\Delta_1 = a_{11}, \Delta_2 = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}, \dots, \Delta_n = |A|$$

отличны от нуля, можно представить, причем единственным образом, в виде LU – разложения.

Алгоритм LU – разложения.

1. Создаем матрицы  $L$  и  $U$ , где  $l_{j1} = \frac{a_{j1}}{u_{11}}, u_{1j} = a_{1j} \quad j = 1, 2, \dots, n$ .
2. Вычисляем коэффициенты для  $i = 2, 3, \dots, n$



$$u_{ij} = a_{ij} - \sum_{k=1}^i l_{ik} u_{kj}, j = i, \dots, n, k = 1, \dots, i,$$

$$l_{ji} = \frac{1}{u_{ii}} (a_{ji} - \sum_{k=1}^i l_{jk} u_{ki}), j = i + 1, \dots, n, k = 1, \dots, i.$$

Решение системы  $Ax = b$  при помощи LU – разложения.

1. Находим LU – разложение матрицы  $A$ .
2. Переходим к системе  $Lz = b$ , где  $z = Ux$ . Решаем ее.
3. Решаем систему  $z = Ux$ .

Применение матрицы  $L$  равносильно проведению прямого хода метода Гаусса, а применение матрицы  $U$  равносильно обратному ходу метода Гаусса.

### 3.3. Описание структур данных и функций

Приложение построено на фреймворке Qt для C++. Все описанные ниже классы окон и диалогов, а также моделей данных, являются мета-сущностями Qt и содержат, помимо обычных полей и методов: реализацию механизма обмена данными, сокрытую в Q\_OBJECT; специальные поля-структуры ui, инкапсулирующие указатели на элементы интерфейса; и не имеющие реализации особые методы-сигналы.

#### 3.3.1. Класс MainWindow

Объявление класса:

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

public slots:
    void showAboutDialog();
    void showHelpDialog();
    void startOver();
    void showWorkspace();
    void hideWorkspace();
    void enableWorkspace();
    void disableWorkspace();
    void solveWithChosenMethod();
    void solveWithAllMethods();
    void toggleSolution();

protected:
    void resizeEvent(QResizeEvent *event) override;
    static QString methodName(int method);

private:
    Ui::MainWindow *ui;
    std::list<QWidget*> m_workspace;
    int m_eq_count;
    SystemTableModel *m_system;
    SolutionTableModel *m_solution;
    std::array<LESystemSolver*, METHODS_COUNT> m_solvers;
};
```

### 3.3.1.1. Поля класса

- `std::list<QWidget*> m_workspace` - хранит список активных виджетов.
- `int m_eq_count` - хранит количество уравнений в СЛАУ.
- `SystemTableModel *m_system` - хранит указатель модель данных СЛАУ.
- `SolutionTableModel *m_solution` - хранит указатель на модель данных решения СЛАУ.
- `std::array<LESystemSolver*, METHODS_COUNT> m_solvers` - хранит массив Решателей СЛАУ.

### 3.3.1.2. Методы класса

- `explicit MainWindow(QWidget *parent = nullptr)` - конструктор инициализации.
- `~MainWindow()` - деструктор.
- `void showAboutDialog()` - открывает окно «О программе»
- `void showHelpDialog()` - открывает окно «Помощь»
- `void startOver()` - запускает процесс ввода СЛАУ заново
- `void showWorkspace()` - включает отображение активных виджетов
- `void hideWorkspace()` - выключает отображение активных виджетов
- `void enableWorkspace()` - включает активные виджеты
- `void disableWorkspace()` - выключает активные виджеты
- `void solveWithChosenMethod()` - запускает решение выбранным методом
- `void solveWithAllMethods()` - запускает решение СЛАУ всеми методами
- `void toggleSolution()` - переключает видимость решения
- `void resizeEvent(QResizeEvent *event) override` - обрабатывает изменение размеров главного окна программы
- `QString methodName(int method)` - возвращает строку с названием метода решения СЛАУ по его внутреннему номеру

### 3.3.2. Класс LSystemSolver

Объявление класса:

```
class LSystemSolver
{
public:
    LSystemSolver() = default;
    virtual ~LSystemSolver() = default;

    virtual Column solve(const Matrix& A,
                        const Column& b,
                        const Column& x = Column(),
                        double epsilon = 0) = 0;
    virtual bool needApproximation() = 0;

    static bool converge(const Column& xk, const Column& xkp, double epsilon);
    static bool diagonalPredominant(const Matrix& A);
    static bool hasZerosDiagonal(const Matrix& A);
    static double determinant(const Matrix& A);
    static double secondVectorNorm(const Column& v);
};
```

#### 3.3.2.1. Методы класса

- `LSystemSolver() = default` – конструктор по умолчанию.
- `~LSystemSolver() = default` – деструктор.
- `Column solve(const Matrix& A, const Column& b, const Column& x = Column(), double epsilon = 0) = 0` – реализует алгоритм решения СЛАУ определенным методом
- `bool needApproximation() = 0` – указывает, нужно ли первое приближение для выполнения алгоритма
- `bool converge(const Column& xk, const Column& xkp, double epsilon)` – вычисляет, сходится ли столбец `xk` к `xkp` с заданной точностью.
- `bool diagonalPredominant(const Matrix& A)` – определяет, является ли матрица диагонально преобладающей.
- `bool hasZerosDiagonal(const Matrix& A)` – определяет, имеет ли матрица хотя бы один ноль на главной диагонали.
- `double determinant(const Matrix& A)` – вычисляет определитель матрицы.
- `double vectorNorm(const Column& v)` – вычисляет норму вектора.

### 3.3.3. Классы-наследники LESystemSolver

Перечисленные ниже классы имеют идентичное объявление, различие только в реализации конкретных методов решения СЛАУ:

- GaussMethodSolver
- KramerMethodSolver
- LUDecompositionMethodSolver
- SeidelMethodSolver
- SimpleIterationMethodSolver
- UpperRelaxationMethodSolver

Объявление классов:

```
class XXXMethodSolver : public LESystemSolver
{
public:
    XXXMethodSolver() = default;
    ~XXXMethodSolver() = default;

    Column solve(const Matrix& A,
                const Column& b,
                const Column& x = Column(),
                double epsilon = 0) override;
    bool needApproximation() override;
};
```

Описание методов: см. Класс LESystemSolver.

### 3.3.4. Класс DataRequestDialog

Объявление класса:

```
class DataRequestDialog : public QDialog
{
    Q_OBJECT
public:
    explicit DataRequestDialog(const Column& column, QWidget *parent = nullptr);
    ~DataRequestDialog();
    const Column& resultColumn() const;
private:
    Ui::DataRequestDialog *ui;
    FirstApproximationTableModel *m_model;
};
```

#### 3.3.4.1. Методы класса

- `explicit DataRequestDialog(const Column& column, QWidget *parent = nullptr)` - конструктор инициализации;
- `~DataRequestDialog()` - деструктор;
- `const Column& resultColumn() const` - возвращает введенный столбец (первое приближение решения СЛАУ).

### 3.3.5. Класс HelpDialog

Объявление класса:

```
class HelpDialog : public QDialog
{
    Q_OBJECT
public:
    explicit HelpDialog(QWidget *parent = nullptr);
    ~HelpDialog();
private:
    Ui::HelpDialog *ui;
};
```

#### 3.3.5.1. Методы класса

- `explicit HelpDialog(QWidget *parent = nullptr)` - конструктор инициализации.
- `~HelpDialog()` - деструктор.

### 3.3.6. Класс AboutDialog

Объявление класса:

```
class AboutDialog : public QDialog
{
    Q_OBJECT
public:
    explicit AboutDialog(QWidget *parent = nullptr);
    ~AboutDialog();
private:
    Ui::AboutDialog *ui;
};
```

#### 3.3.6.1. Методы класса

- `explicit AboutDialog(QWidget *parent = nullptr)` - конструктор инициализации.
- `~AboutDialog()` - деструктор.

### 3.3.7. Основная программа

```
int main();
```

**Назначение:** основная функция (точка входа).

## Заключение

В результате работы над лабораторной работой были изучены методы решения систем линейных уравнений точными и итерационными методами, получен опыт реализации алгоритмов решения систем линейных уравнений, написано приложение для их решения. Программа полностью соответствует описанным методам и решает поставленную задачу решения систем линейных алгебраических уравнений.



## Список используемых источников

1. Волков Е.А. Численные методы. [Электронный ресурс]: учеб. — Электрон. дан. — СПб.: Лань, 2008 — 256 с. — Режим доступа: <http://e.lanbook.com/book/54> — Загл. с экрана.
2. Д.П. Кострамов, А.П. Фаворский Вводные лекции по численным методам: Учеб. Пособие. — М.: Логос, 2004. — 184 с.: ил.
3. Шевцов, Г.С. Численные методы линейной алгебры. [Электронный ресурс]: учеб. пособие / Г.С. Шевцов, О.Г. Крюкова, Б.И. Мызникова. — Электрон. дан. — СПб.: Лань, 2011. — 496 с.
4. Qt 5.15 Reference Pages [Электронный ресурс]: Документация к фреймворку Qt. URL: <https://doc.qt.io/qt-5/reference-overview.html>