

**Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования «Северо-Осетинский государственный университет
имени Коста Левановича Хетагурова»**

Факультет Математики и информационных технологий
Кафедра Прикладной математики

**Отчет
о прохождении учебной практики
по получению первичных профессиональных умений и навыков**

Исполнитель:

студент 2 курса очной формы обучения,
направления подготовки бакалавриата
01.03.02 Прикладная математика и
информатика
Гамосов Станислав Станиславович

Руководитель практики:

к.ф.-м.н.,
доцент кафедры прикладной математики,
Биткина Виктория Васильевна

Владикавказ 2020 г.

Оглавление

ВВЕДЕНИЕ	3
ОСНОВНЫЕ ПОНЯТИЯ	4
ОСНОВНАЯ ЧАСТЬ.....	6
1. Прямые методы решения СЛАУ	7
1.1. Правило Крамера	7
1.2. Метод обратных матриц	8
1.3. Метод Гаусса.....	10
1.4. Метод прогонки	13
1.5. Метод квадратного корня	15
2. Итерационные методы решения СЛАУ	21
2.1. Метод простой итерации	25
2.2. Метод Зейделя	29
ЗАКЛЮЧЕНИЕ.....	33
СПИСОК ЛИТЕРАТУРЫ.....	34
ПРИЛОЖЕНИЯ	35
1. Реализация вспомогательного класса Vector	35
2. Реализация вспомогательного класса Matrix	37
3. Реализация вспомогательного класса Permutations.....	44
4. Реализация основного класса SLE(СЛАУ)	47
5. Метод Крамера.....	48
6. Метод обратной матрицы	49
7. Метод Гаусса.....	50
8. Метод прогонки	51
9. Метод квадратных корней	52
10. Метод простых итераций.....	54
11. Метод Зейделя	55

ВВЕДЕНИЕ

Проблемы, соответствующие современным задачам на составление и решение систем уравнений с несколькими неизвестными, встречаются еще в вавилонских и египетских рукописях II века до н.э., а также в трудах древнегреческих, индийских и китайских мудрецов. В китайском трактате "Математика в девяти книгах" словесно изложены правила решения систем уравнений, были замечены некоторые закономерности при решении.

Система может состоять из алгебраических уравнений, линейных алгебраических уравнений, нелинейных уравнений, дифференциальных уравнений.

Методы решения системы уравнений зависят от типа системы. Например, решения систем линейных алгебраических уравнений хорошо известны (метод Крамера, метод Гаусса, матричный метод, метод итераций и т.д.). Для нелинейных же систем общего аналитического решения не найдено, они решаются разного рода численными методами. Аналогично дело обстоит и с системами дифференциальных уравнений.

Системы линейных уравнений широко используются в задачах экономики, физики, химии и других науках.

Решение систем линейных алгебраических уравнений - одна из основных задач вычислительной линейной алгебры. Хотя задача решения именно системы линейных уравнений сравнительно редко представляет самостоятельный интерес для прикладных задач, но от умения эффективно решать данные системы часто зависит сама возможность математического моделирования самых разнообразных процессов с применением ЭВМ.

В этой работе рассматриваются численные методы решения систем линейных алгебраических уравнений. В частности: метод Крамера, метод Гаусса, метод обратных матриц, метод квадратного корня, метод прогонки, метод простой итерации и метод Зейделя.

ОСНОВНЫЕ ПОНЯТИЯ

Системы линейных алгебраических уравнений (СЛАУ) являются важной математической моделью линейной алгебры. На их базе ставятся такие практические математические задачи, как:

- непосредственное решение линейных систем;
- вычисление определителей матриц;
- вычисление элементов обратных матриц;
- определение собственных значений и собственных векторов матриц.

Решение линейных систем является одной из самых распространенных задач вычислительной математики. К их решению сводятся многочисленные практические задачи нелинейного характера, решения дифференциальных уравнений и др.

Вторая и третья задачи являются также и компонентами технологии решения самих линейных систем.

Обычно СЛАУ n -го порядка записывается в виде

$$\sum_{j=1}^n a_{ij} x_j = b_i; \quad i = \overline{1, n}$$

или в развернутой форме

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

или в векторной форме

$$A\bar{x} = \bar{b}. \quad (2)$$

В соотношениях (2):

A называется основной матрицей системы с n^2 элементами;

$\bar{x} = (x_1, x_2, \dots, x_n)^T$ – вектор-столбец неизвестных;

$\bar{b} = (b_1, b_2, \dots, b_n)^T$ – вектор-столбец свободных членов.

Первоначальным при решении СЛАУ (1) является анализ вида исходной матрицы A и вектора-столбца свободных членов \bar{b} в (2).

Если все свободные члены равны нулю, т.е. $\bar{b} = 0$, то система $A\bar{x} = 0$ называется *однородной*. Если же $\bar{b} \neq 0$, или хотя бы одно $b_i \neq 0$ ($i = \overline{1, n}$), то система (2) называется *неоднородной*.

Квадратная матрица A называется *невыврожденной*, или *неособенной*, если ее определитель $|A| \neq 0$. При этом система (1) имеет единственное решение.

При $|A| = 0$ матрица A называется *вырожденной*, или *особенной*, а система (1) не имеет решения, либо имеет бесконечное множество решений.

Если $|A| \approx 0$ система (1) называется *плохо обусловленной*, т.е. решение очень чувствительно к изменению коэффициентов системы.

В ряде случаев получаются системы уравнений с матрицами специальных видов: диагональные, трехдиагональные (частный случай ленточных), симметричные ($a_{ij} = a_{ji}$), единичные (частный случай диагональной), треугольные и др.

Решение системы (2) заключается в отыскании вектора-столбца $\bar{x} = (x_1, x_2, \dots, x_n)^T$, который обращает каждое уравнение системы в тождество.

Существуют две величины, характеризующие степень отклонения полученного решения от точного, которые появляются в связи с округлением и ограниченностью разрядной сетки ЭВМ, – погрешность ε и «невязка» r :

$$\begin{cases} \varepsilon = \bar{x} - \overline{x^*}; \\ r = \bar{B} - A\bar{x}^*; \end{cases} \quad (3)$$

где $\overline{x^*}$ – вектор решения. Как правило, значения вектора \bar{x} – неизвестны.

Доказано, что если $\varepsilon \approx 0$, то и $r = 0$. Обратное утверждение не всегда верно. Однако если система не плохо обусловлена, для оценки точности решения используют невязку r .

ОСНОВНАЯ ЧАСТЬ

Методы решения СЛАУ делятся на две группы:

- прямые (точные) методы;
- итерационные (приближенные) методы.

К **прямым** методам относятся такие методы, которые, в предположении, что вычисления ведутся без округлений, позволяют получить точные значения неизвестных. Они просты, универсальны и используются для широкого класса систем. Однако они не применимы к системам больших порядков ($n < 200$) и к плохо обусловленным системам из-за возникновения больших погрешностей. К ним можно отнести: *правило Крамера*, методы *обратных матриц*, *Гаусса*, *прогонки*, *квадратного корня* и др.

К **приближенным** относятся методы, которые даже в предположении, что вычисления ведутся без округлений, позволяют получить решение системы лишь с заданной точностью. Это итерационные методы, т.е. методы последовательных приближений. К ним относятся методы *простой итерации*, *Зейделя*.

1. Прямые методы решения СЛАУ

1.1. Правило Крамера

1.1.1. Теоретическая часть

Рассмотрим систему (1). Как отмечалось выше, если определитель этой системы не равен нулю, то будет иметь место единственное решение. Это необходимое и достаточное условие. Тогда по правилу Крамера

$$x_k = \frac{D_k}{D}, \quad k = \overline{1, n}, \quad (4)$$

где D_k – определитель, получающийся из D при замене элементов $a_{1k}, a_{2k}, \dots, a_{nk}$ k -го столбца (соответствующими) свободными членами b_1, b_2, \dots, b_n из (1), или

$$D_k = \sum_{i=1}^n A_{ik} b_i, \quad k = \overline{1, n},$$

где A_{ik} алгебраическое дополнение элемента a_{ik} в определителе D . Стоит существенная проблема вычисления определителей высоких порядков.

1.1.2 Практическая часть

Рассмотрим пример:

$$\begin{cases} 2x_1 + x_2 + x_3 = 2 \\ x_1 - x_2 = -2 \\ 3x_1 - x_2 + 2x_3 = 2 \end{cases}$$

Вычислим определитель матрицы:

$$\Delta = \begin{vmatrix} 2 & 1 & 1 \\ 1 & -1 & 0 \\ 3 & -1 & 2 \end{vmatrix} = 2 * (-1) * 2 + 1 * (-1) * 1 + 1 * 0 * 3 - 3 * (-1) * 1 - (-1) * 0 * 2 - 1 * 1 * 2 = -4 \neq 0$$

Система не равна нулю, следовательно, система имеет единственное решение. Вычислим определители:

$$\Delta_1 = \begin{vmatrix} 2 & 1 & 1 \\ -2 & -1 & 0 \\ 2 & -1 & 2 \end{vmatrix} = 2 * (-1) * 2 + (-2) * (-1) * 1 + 1 * 0 * 2 - 2 * (-1) * 1 - (-1) * 0 * 2 - 1 * 1 * 2 = 4$$

$$\Delta_2 = \begin{vmatrix} 2 & 2 & 1 \\ 1 & -2 & 0 \\ 3 & 2 & 2 \end{vmatrix} = 2 * (-2) * 2 + 1 * 2 * 1 + 2 * 0 * 3 - 3 * (-2) * 1 -$$

$$2 * 0 * 2 - 1 * 2 * 2 = -4$$

$$\Delta_3 = \begin{vmatrix} 2 & 1 & 2 \\ -1 & -1 & -2 \\ 3 & -1 & 2 \end{vmatrix} = 2 * (-1) * 2 + 1 * (-1) * 2 + 1 * (-2) * 3 -$$

$$3 * (-1) * 2 - (-1) * (-2) * 2 - 1 * 1 * 2 = -12$$

Таким образом:

$$\begin{cases} x_1 = \frac{\Delta_1}{\Delta} = \frac{4}{-4} = -1 \\ x_2 = \frac{\Delta_2}{\Delta} = \frac{-4}{-4} = 1 \\ x_3 = \frac{\Delta_3}{\Delta} = \frac{-12}{-4} = 3 \end{cases}$$

1.2. Метод обратных матриц

1.2.1. Теоретическая часть

Дана система $A\bar{x} = \bar{b}$. Умножим левую и правую части этого выражения на A^{-1} :

$$A^{-1} A \bar{x} = A^{-1} \bar{b}; \quad \bar{x} = A^{-1} \bar{b}.$$

При его реализации стоит проблема нахождения обратной матрицы A^{-1} .

1.2.2 Практическая часть

Рассмотрим пример:

$$\begin{cases} 2x_1 - 4x_2 + 3x_3 = 1 \\ x_1 - 2x_2 + 4x_3 = 3 \\ 3x_1 - x_2 + 5x_3 = 2 \end{cases}$$

Запишем в матричном уравнении вида $A * X = B$,

где $A = \begin{pmatrix} 2 & -4 & 3 \\ 1 & -2 & 4 \\ 3 & -1 & 5 \end{pmatrix}$, $X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$, $B = \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix}$ Выразим X:

$$X = A^{-1} * B$$

Найдём определитель матрицы A:

$$\Delta = \begin{vmatrix} 2 & 4 & 3 \\ 1 & 2 & 4 \\ 3 & -1 & 5 \end{vmatrix} = 2 * (-1) * 5 + 3 * (-4) * 4 + 3 * (-1) * 1 - 3 * (-2) * 3 - 1 * (-4) * 5 - 2 * 4 * (-1) = -25 \neq 0$$

Система неравна нулю, следовательно, система имеет единственное решение. Найдем обратную матрицу A^{-1} с помощью союзной матрицы. Вычислим алгебраические дополнения A_{ij} к соответствующим элементам матрицы A :

$$\begin{aligned} A_{11} &= (-1)^{1+1} \cdot \begin{vmatrix} -2 & 4 \\ -1 & 5 \end{vmatrix} = -10 + 4 = -6; & A_{12} &= (-1)^{1+2} \cdot \begin{vmatrix} 1 & 4 \\ 3 & 5 \end{vmatrix} = -(5 - 12) = 7; \\ A_{13} &= (-1)^{1+3} \cdot \begin{vmatrix} 1 & -2 \\ 3 & -1 \end{vmatrix} = -1 + 6 = 5; & A_{21} &= (-1)^{2+1} \cdot \begin{vmatrix} -4 & 3 \\ -1 & 5 \end{vmatrix} = -(-20 + 3) = 17; \\ A_{22} &= (-1)^{2+2} \cdot \begin{vmatrix} 2 & 3 \\ 3 & 5 \end{vmatrix} = 10 - 9 = 1; & A_{23} &= (-1)^{2+3} \cdot \begin{vmatrix} 2 & -4 \\ 3 & -1 \end{vmatrix} = -(-2 + 12) = -10 \\ A_{31} &= (-1)^{3+1} \cdot \begin{vmatrix} -4 & 3 \\ -2 & 4 \end{vmatrix} = -16 + 6 = -10; \\ A_{32} &= (-1)^{3+2} \cdot \begin{vmatrix} 2 & 3 \\ 1 & 4 \end{vmatrix} = -(8 - 3) = -5; & A_{33} &= (-1)^{3+3} \cdot \begin{vmatrix} 2 & -4 \\ 1 & -2 \end{vmatrix} = -4 + 4 = 0. \end{aligned}$$

Запишем союзную матрицу A^* , составленную из алгебраических дополнений элементов матрицы A :

$$A^* = \begin{pmatrix} 6 & 7 & 5 \\ 17 & 1 & -10 \\ 10 & -5 & 0 \end{pmatrix}$$

Далее запишем обратную матрицу согласно формуле $A^{-1} = \frac{1}{\Delta} * A^*$.

Получим:

$$A^{-1} = -\frac{1}{25} * \begin{pmatrix} 6 & 17 & -10 \\ 7 & 1 & -5 \\ 5 & -10 & 0 \end{pmatrix}$$

Умножая обратную матрицу A^{-1} на столбец свободных членов B , получим искомое решение исходной системы:

$$X = A^{-1} * B = -\frac{1}{25} * \begin{pmatrix} 6 & 17 & -10 \\ 7 & 1 & -5 \\ 5 & -10 & 0 \end{pmatrix} * \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$$

Ответ: $x_1 = -1$, $x_2 = 0$, $x_3 = 1$

1.3. Метод Гаусса

1.3.1. Теоретическая часть

Этот метод является наиболее распространенным методом решения СЛАУ. В его основе лежит идея последовательного исключения неизвестных, в основном, приводящая исходную систему к треугольному виду, в котором все коэффициенты ниже главной диагонали равны нулю. Существуют различные вычислительные схемы, реализующие этот метод. Наибольшее распространение имеют схемы с выбором главного элемента либо по строке, либо по столбцу, либо по всей матрице. С точки зрения простоты реализации, хотя и с потерей точности, перед этими схемами целесообразней применять так называемую схему единственного деления. Рассмотрим ее суть.

Посредством первого уравнения системы (1) исключается x_1 из последующих уравнений. Далее посредством второго уравнения исключается x_2 из последующих уравнений и т.д. Этот процесс называется **прямым ходом Гаусса**. Исключение неизвестных повторяется до тех пор, пока в левой части последнего n -го уравнения не останется одно неизвестное x_n

$$a'_{mn}x_n = b', \quad (5)$$

где a'_{mn} и b' – коэффициенты, полученные в результате линейных (эквивалентных) преобразований.

Прямой ход реализуется по формулам

$$\left. \begin{aligned} a^*_{mi} &= a_{mi} - a_{ki} \frac{a_{mk}}{a_{kk}}, \quad k = \overline{1, n-1}; \quad i = \overline{k, n}; \\ b^*_m &= b_m - b_k \frac{a_{mk}}{a_{kk}}, \quad m = \overline{k+1, n} \end{aligned} \right\} \quad (6)$$

где m – номер уравнения, из которого исключается x_k ;

k – номер неизвестного, которое исключается из оставшихся $(n - k)$ уравнений, а также обозначает номер уравнения, с помощью которого исключается x_k ;

i – номер столбца исходной матрицы;

a_{kk} – главный (ведущий) элемент матрицы.

Во время счета необходимо следить, чтобы $a_{kk} \neq 0$. В противном случае прибегают к перестановке строк матрицы.

Обратный ход метода Гаусса состоит в последовательном вычислении x_n, x_{n-1}, \dots, x_1 , начиная с (5) по алгоритму

$$x_n = b' / a'_{nn}; \quad x_k = \frac{1}{a'_{kk}} \left[b'_k - \sum_{i=k+1}^n a'_{ki} x_i \right], \quad k = \overline{n-1, 1}. \quad (7)$$

Точность полученного решения оценивается посредством «невязки» (3). В векторе невязки $(r_1, r_2, \dots, r_n)^T$ отыскивается максимальный элемент и сравнивается с заданной точностью ε . Приемлемое решение будет, если $r_{\max} < \varepsilon$. В противном случае следует применить схему уточнения решения.

Уточнение корней

Полученные методом Гаусса приближенные значения корней можно уточнить.

Пусть для системы $A\bar{x} = \bar{b}$ найдено приближенное решение $\overline{x_0}$, не удовлетворяющее по «невязке». Положим тогда $\bar{x} = \overline{x_0} + \bar{\delta}$. Для получения поправки $\bar{\delta} = (\delta_1, \delta_2, \dots, \delta_n)^T$ корня $\overline{x_0}$ следует рассмотреть новую систему

$$A(\overline{x_0} + \bar{\delta}) = \bar{b} \quad \text{или} \quad A\bar{\delta} = \bar{\varepsilon},$$

где $\bar{\varepsilon} = \bar{b} - A\overline{x_0}$ – невязка для исходной системы.

Таким образом, решая линейную систему с прежней матрицей A и новым свободным членом $\bar{\varepsilon} = (\varepsilon_1, \varepsilon_2, \dots, \varepsilon_n)^T$, получим поправки $(\delta_1, \delta_2, \dots, \delta_n)$.

Пример решения СЛАУ по методу Гаусса (с точностью до трех знаков). Нужно уточнить корни до 10^{-4} :

$$\begin{cases} 6x_1 - x_2 - x_3 = 11,33; \\ -x_1 + 6x_2 - x_3 = 32; \\ -x_1 - x_2 + 6x_3 = 42. \end{cases}$$

В результате $x_1^{(0)} = 4,67$; $x_2^{(0)} = 7,62$; $x_3^{(0)} = 9,05$. Невязки равны $\varepsilon_1^{(0)} = -0,02$; $\varepsilon_2^{(0)} = 0$; $\varepsilon_3^{(0)} = -0,01$. Получено уточнение $\delta_1^{(0)} = -0,0039$; $\delta_2^{(0)} = -0,0011$; $\delta_3^{(0)} = -0,0025$. Следовательно, $x_1 = 4,6661$; $x_2 = 7,6189$; $x_3 = 9,0475$. Невязки будут $\delta_1 = -2 \cdot 10^{-4}$; $\delta_2 = -2 \cdot 10^{-4}$; $\delta_3 = 0$.

1.3.2 Практическая часть

Рассмотрим пример:

$$\begin{cases} 4x_1 + 2x_2 - x_3 = 1 \\ 5x_1 + 3x_2 - 2x_3 = 2 \\ 3x_1 + 2x_2 - 3x_3 = 0 \end{cases}$$

Запишем расширенную матрицу системы и с помощью элементарных преобразований приведем ее к ступенчатому виду:

$$\left(\begin{array}{ccc|c} 4 & 2 & -1 & 1 \\ 5 & 3 & -2 & 2 \\ 3 & 2 & -3 & 0 \end{array} \right)$$

К первой строке прибавляем вторую строку, умноженную на -1 :

$$\left(\begin{array}{ccc|c} 1 & -1 & 1 & -1 \\ 5 & 3 & -2 & 2 \\ 3 & 2 & -3 & 0 \end{array} \right)$$

Ко второй строке прибавим первую строку, умноженную на 5. К третьей строке прибавим первую строку, умноженную на 3.

$$\left(\begin{array}{ccc|c} 1 & -1 & 1 & -1 \\ 0 & -2 & 3 & -3 \\ 0 & -1 & 0 & -3 \end{array} \right)$$

Первую строку умножим на -1 , в принципе, это для красоты. У третьей строки также сменили знак и переставим её на второе место, таким образом, на второй «ступеньке у нас появилась нужная единица.

$$\left(\begin{array}{ccc|c} 1 & -1 & 1 & -1 \\ 0 & 1 & 0 & 3 \\ 0 & -2 & 3 & -3 \end{array} \right)$$

К третьей строке прибавим вторую строку, умноженную на 2.

$$\left(\begin{array}{ccc|c} 1 & 1 & -1 & 1 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 3 & 3 \end{array} \right)$$

Третью строку разделим на 3.

$$\left(\begin{array}{ccc|c} 1 & 1 & -1 & 1 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 1 \end{array} \right)$$

$$x_3 = 1, \quad x_2 = 3, \quad x_1 + x_2 - x_3 = 1 \Rightarrow x_1 + 3 - 1 = 1 \Rightarrow x_1 = -1$$

$$\text{Ответ: } x_1 = -1, \quad x_2 = 3, \quad x_3 = 1$$

1.4. Метод прогонки

1.4.1. Теоретическая часть

Данный метод также является модификацией метода Гаусса для частного случая *разреженных* систем – систем с матрицей трехдиагонального типа.

Каноническая форма их записи

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i; \quad i = \overline{1, n}; \quad a_1 = c_n = 0, \quad (9)$$

или в развернутом виде

$$\left\{ \begin{array}{lcl} b_1 x_1 + c_1 x_2 & & = d_1; \\ a_2 x_1 + b_2 x_2 + c_2 x_3 & & = d_2; \\ & a_3 x_2 + b_3 x_3 + c_3 x_4 & = d_3; \\ & & \dots \end{array} \right. \quad (10)$$

$$a_{n-1} x_{n-2} + b_{n-1} x_{n-1} + c_{n-1} x_n = d_{n-1};$$

$$a_n x_{n-1} + b_n x_n = d_n.$$

При этом, как правило, все коэффициенты $b_i \neq 0$.

Метод реализуется в два этапа – прямой и обратный ходы.

Прямой ход. Каждое неизвестное x_i выражается через x_{i+1}

$$x_i = A_i \cdot x_{i+1} + B_i \quad \text{для } i = 1, 2, \dots, n-1, \quad (11)$$

посредством прогоночных коэффициентов A_i и B_i . Определим алгоритм их вычисления.

Из первого уравнения системы (10) находим x_1

$$x_1 = -\frac{c_1}{b_1} x_2 + \frac{d_1}{b_1}.$$

Из уравнения (11) при $i=1$: $x_1 = A_1 \cdot x_2 + B_1$. Следовательно

$$A_1 = -\frac{c_1}{b_1}; \quad B_1 = \frac{d_1}{b_1}. \quad (12)$$

Из второго уравнения системы (10) определяем x_2 через x_3 , подставляя найденное значение x_1

$$a_2 (A_1 x_2 + B_1) + b_2 x_2 + c_2 x_3 = d_2,$$

откуда

$$x_2 = \frac{-c_2 x_3 + d_2 - a_2 B_1}{a_2 A_1 + b_2}; \quad (12^*)$$

и согласно (11) при $i = 2$: $x_2 = A_2 \cdot x_3 + B_2$, следовательно

$$A_2 = -\frac{c_2}{e_2}; \quad B_2 = \frac{d_2 - a_2 B_1}{e_2}, \text{ где } e_2 = a_2 \cdot A_1 + b_2.$$

Ориентируясь на соотношения индексов при коэффициентах (12) и (12*) можно получить эти соотношения для общего случая

$$A_i = -\frac{c_i}{e_i}; \quad B_i = \frac{d_i - a_i B_{i-1}}{e_i}, \text{ где } e_i = a_i \cdot A_{i-1} + b_i (i=2,3, \dots, n-1). \quad (13)$$

Обратный ход. Из последнего уравнения системы (10) с использованием (11) при $i = n-1$

$$x_n = \frac{d_n - a_n B_{n-1}}{b_n + a_n A_{n-1}}. \quad (14)$$

Далее посредством (11) и прогоночных коэффициентов (12), (13) последовательно вычисляем $x_{n-1}, x_{n-2}, \dots, x_1$.

При реализации метода прогонки нужно учитывать, что при условии

$$|b_i| \geq |a_i| + |c_i|, \quad (15)$$

или хотя бы для одного b_i имеет место строгое неравенство (15), деление на «0» исключается, и система имеет единственное решение.

Заметим, что условие (15) является достаточным, но не необходимым. В ряде случаев для хорошо обусловленных систем (10) метод прогонки может быть устойчивым и при несоблюдении условия (15).

1.4.2. Практическая часть

Рассмотрим пример:

$$\begin{cases} 2x_1 - x_2 = 3 \\ 5x_1 + 4x_2 + 2x_3 = 6 \\ x_2 - 3x_3 = 2 \end{cases}$$

Прямой ход:

$$1) \ y_1 = b_1 = 2, \ \alpha_1 = \frac{-c_1}{y_1} = \frac{1}{2}, \ \beta_1 = \frac{d_1}{y_1} = \frac{3}{2}$$

$$2) \ y_2 = b_2 + a_2\alpha_1 = 4 + 5 * 0,5 = 6,5, \ \alpha_2 = \frac{-c_2}{y_2} = \frac{-2}{6,5}, \ \beta_2 = \frac{d_2 - a_2\beta_1}{y_2} = \frac{6 - 5 * 1,5}{6,5} = -0,23$$

$$3) \ y_3 = b_3 + a_3\alpha_2 = -3 - 0,3 = -3,3, \ \beta_2 = \frac{d_3 - a_3\beta_2}{y_3} = \frac{2 + 0,23}{-3,3} = -0,68$$

Обратный ход:

$$1) \ x_3 = \beta_3 = -0,68$$

$$2) \ x_2 = \alpha_2 * x_3 + \beta_2 = -0,02$$

$$3) \ x_1 = \alpha_2 * x_2 + \beta_1 = 1,49$$

1.5. Метод квадратного корня

1.5.1. Теоретическая часть

Данный метод используется для решения линейной системы

$$A\bar{x} = \bar{b}, \tag{16}$$

у которой матрица A симметрическая, т.е. $A^T = A$, $a_{ij} = a_{ji} \ (i=j=1, \dots, n)$.

Решение системы (16) осуществляется в два этапа.

Прямой ход. Преобразование матрицы A и представление ее в виде произведения двух взаимно транспонированных треугольных матриц:

$$A = S^T \cdot S, \tag{17}$$

где

$$S = \begin{vmatrix} s_{11} & s_{12} & \cdots & s_{1n} \\ 0 & s_{22} & \cdots & s_{2n} \\ & & \cdots & \\ 0 & 0 & \cdots & s_{nn} \end{vmatrix}; \quad S^T = \begin{vmatrix} s_{11} & 0 & \cdots & 0 \\ s_{12} & s_{22} & \cdots & 0 \\ & & \cdots & \\ s_{1n} & s_{2n} & \cdots & s_{nn} \end{vmatrix}.$$

Перемножая S^T и S , и приравнявая матрице A , получим следующие формулы для определения s_{ij} :

$$\begin{cases} s_{11} = \sqrt{a_{11}}, & s_{1j} = a_{1j} / s_{11}, \quad (j > 1); \\ s_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} s_{ki}^2}, & (1 \leq i \leq n); \\ s_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} s_{ki} s_{kj}}{s_{ii}}, & (i < j); \\ s_{ij} = 0, & (i > j). \end{cases} \quad (18)$$

После нахождения матрицы S систему (16) заменяем двумя ей эквивалентными системами с треугольными матрицами (17)

$$S^T \bar{y} = \bar{b}, \quad S \bar{x} = \bar{y}. \quad (19)$$

Обратный ход. Записываем системы (19) в развернутом виде:

$$\begin{cases} s_{11} y_1 = b_1; \\ s_{12} y_1 + s_{22} y_2 = b_2; \\ \dots \\ s_{1n} y_1 + s_{2n} y_2 + \dots + s_{nn} y_n = b_n; \end{cases} \quad (20)$$

$$\begin{cases} s_{11} x_1 + s_{12} x_2 + \dots + s_{1n} x_n = y_1; \\ s_{22} x_2 + \dots + s_{2n} x_n = y_2; \\ \dots \\ s_{nn} x_n = y_n. \end{cases} \quad (21)$$

Используя (20) и (21) последовательно находим

$$y_1 = \frac{b_1}{s_{11}}, \quad y_i = (b_i - \sum_{k=1}^{i-1} s_{ki} y_k) / s_{ii}; \quad (i > 1); \quad (22)$$

$$x_n = \frac{y_n}{s_{nn}}, \quad x_i = (y_i - \sum_{k=i+1}^n s_{ik} x_k) / s_{ii}; \quad (i < n). \quad (23)$$

Метод квадратных корней дает большой выигрыш во времени по сравнению с рассмотренными ранее прямыми методами, так как, во-первых, существенно уменьшает число умножений и делений (почти в два раза), во-вторых, позволяет накапливать сумму произведений без записи промежуточных результатов.

Машинная реализация метода предусматривает его следующую трактовку. Исходная матрица A системы (16) представляется в виде произведения трех матриц

$$A = S^T \cdot D \cdot S,$$

где D – диагональная матрица с элементами $d_{ii} = \pm 1$; S – верхняя треугольная ($s_{ik} = 0$, если $i > k$, причем $s_{ii} > 0$); S^T – транспонированная нижняя треугольная.

Требование выполнения условия $s_{ii} > 0$ необходимо для полной определенности разложения. Это и определяет необходимость введения диагональной матрицы D .

Рассмотрим алгоритм разложения матрицы A с использованием матрицы D на примере матрицы второго порядка.

Пусть A – действительная симметричная матрица

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}.$$

Будем искать S и D в виде

$$S = \begin{bmatrix} s_{11} & s_{12} \\ 0 & s_{22} \end{bmatrix}, \quad D = \begin{bmatrix} d_{11} & 0 \\ 0 & d_{22} \end{bmatrix}, \quad \text{где } d_{ii} = \pm 1.$$

Тогда

$$S^T D S = \begin{bmatrix} s_{11}^2 d_{11} & s_{11} s_{12} d_{11} \\ s_{11} s_{12} d_{11} & s_{12}^2 d_{11} + s_{22}^2 d_{22} \end{bmatrix}.$$

Из условия равенства $A = S^T D S$, получим три уравнения

$$s_{11}^2 d_{11} = a_{11}; \quad s_{11} s_{12} d_{11} = a_{12}; \quad s_{12}^2 d_{11} + s_{22}^2 d_{22} = a_{22}.$$

Из первого уравнения находим

$$d_{11} = \text{sign } a_{11}; \quad s_{11} = \sqrt{|a_{11}|}.$$

Далее, если $a_{11} \neq 0$, то $s_{12} = a_{12} / (s_{11} d_{11})$, и, наконец

$$s_{22}^2 d_{22} = a_{22} - s_{12}^2 d_{11},$$

$$\text{т.е. } d_{22} = \text{sign } (a_{22} - s_{12}^2 d_{11}); \quad s_{22} = \sqrt{|a_{22} - s_{12}^2 d_{11}|}.$$

Здесь и для общего случая матрицу S можно по аналогии с числами трактовать как корень квадратный из матрицы A , отсюда и название метода.

Итак, если $S^T D S$ известно, то решение исходной системы $Ax = b$ сводится к последовательному решению систем:

$$S^T D y = b \quad Sx = y. \quad (23)$$

Нахождение элементов матрицы S (извлечение корня из A) осуществляется по рекуррентным формулам, избежав проблемы оперирования комплексными числами:

$$d_k = \text{sign} \left(a_{kk} - \sum_{i=1}^{k-1} d_i s_{ik}^2 \right);$$

$$s_{kk} = \sqrt{\left| a_{kk} - \sum_{i=1}^{k-1} d_i s_{ik}^2 \right|}; \quad (24)$$

$$s_{kj} = \left(a_{kj} - \sum_{i=1}^{k-1} d_i s_{ik} s_{ij} \right) / (s_{kk} d_k);$$

$$k = 1, 2, \dots, n; \quad j = k+1, k+2, \dots, n.$$

В этих формулах сначала полагаем $k = 1$ и последовательно вычисляем

$$d_1 = \text{sign } (a_{11}); \quad s_{11} = \sqrt{|a_{11}|}$$

и все элементы первой строки матрицы S ($s_{1j}, j>1$), затем полагаем $k = 2$, вычисляем s_{22} и вторую строку матрицы s_{1j} для $j>2$ и т.д.

Решение систем (23) ввиду треугольности матрицы S осуществляется по формулам, аналогичным обратному ходу метода Гаусса:

$$y_1 = \frac{b_1}{s_{11}d_1}, \quad y_i = (b_i - \sum_{k=1}^{i-1} d_k s_{ki} y_k) / (s_{ii} d_i); \quad i=2,3,\dots,n;$$

$$x_n = \frac{y_n}{s_{nn}}, \quad x_i = (y_i - \sum_{k=i+1}^n s_{ik} x_k) / s_{ii}; \quad i=n-1, n-2, \dots, 1.$$

Метод квадратного корня почти вдвое эффективнее метода Гаусса, т.к. полезно использует симметричность матрицы.

Проиллюстрируем метод квадратного корня, решая систему трех уравнений:

$$\begin{cases} x_1 + x_2 + x_3 = 3; \\ x_1 + 2x_2 + 2x_3 = 5; \\ x_1 + 2x_2 + 3x_3 = 6; \end{cases} \quad A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 3 \end{bmatrix}, \quad b = \begin{bmatrix} 3 \\ 5 \\ 6 \end{bmatrix}.$$

Нетрудно проверить, что матрица A есть произведение двух треугольных матриц (здесь $d_{ii} = 1$):

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = S^T \cdot S.$$

Исходную систему запишем в виде

$$S^T \cdot S \cdot \vec{x} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 6 \end{bmatrix}.$$

Обозначим

$$S \cdot \vec{x} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}.$$

Тогда для вектора \vec{y} получим систему $S^T y = b$:

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 6 \end{bmatrix}, \quad \text{откуда} \quad y_1 = 3; y_2 = 2; y_3 = 1.$$

Зная \vec{y} , решаем систему $Sx = y$:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}, \quad \text{откуда} \quad x_1 = 1; x_2 = 1; x_3 = 1.$$

1.5.2 Практическая часть

Рассмотрим пример:

$$\begin{cases} 2x_1 + x_2 + 4x_3 = 16 \\ x_1 + x_2 + 3x_3 = 12 \\ 4x_1 + 3x_2 + 14x_3 = 52 \end{cases}$$

Составим из нее матрицу M :

$$M = \begin{pmatrix} 2 & 1 & 4 \\ 1 & 1 & 3 \\ 4 & 3 & 14 \end{pmatrix}$$

Представим матрицу M в виде: $M = A^T A$

$$A = \begin{pmatrix} \sqrt{2} & \frac{1}{\sqrt{2}} & \frac{4}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & \sqrt{2} \\ 0 & 0 & 4 \end{pmatrix}$$

$$A^T = \begin{pmatrix} \sqrt{2} & 0 & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{4}{\sqrt{2}} & \sqrt{2} & 2 \end{pmatrix}$$

Решим систему $A^T * t = b$:

$$\begin{pmatrix} \sqrt{2} & 0 & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{4}{\sqrt{2}} & \sqrt{2} & 2 \end{pmatrix} * \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} = \begin{pmatrix} 16 \\ 12 \\ 52 \end{pmatrix}$$

$$\begin{cases} \sqrt{2}t_1 = 16 \\ \frac{1}{\sqrt{2}}t_1 + \frac{1}{\sqrt{2}}t_2 = 12 \\ \frac{4}{\sqrt{2}}t_1 + \sqrt{2}t_2 + 2t_3 = 52 \end{cases}$$

$$\begin{cases} t_1 = 8\sqrt{2} \\ t_2 = 4\sqrt{2} \\ t_3 = 6 \end{cases}$$

Решим систему $A * x = t$:

$$\begin{pmatrix} \sqrt{2} & 0 & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{4}{\sqrt{2}} & \sqrt{2} & 2 \end{pmatrix} * \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 8\sqrt{2} \\ 4\sqrt{2} \\ 16 \end{pmatrix}$$

Ответ:

$$\begin{cases} x_1 = 1 \\ x_2 = 2 \\ x_3 = 3 \end{cases}$$

2. Итерационные методы решения СЛАУ

Напомним, что достоинством итерационных методов является их применимость к плохо обусловленным системам и системам высоких порядков, их самоуправляемость и простота реализации на ЭВМ. Итерационные методы для начала вычисления требуют задания какого-либо начального приближения к искомому решению.

Следует заметить, что условия и скорость сходимости итерационного процесса существенно зависят от свойств матрицы A системы и от выбора начальных приближений.

Для применения метода итераций исходную систему (1) или (2) необходимо привести к виду

$$\bar{x} = G\bar{x} + \bar{f} \quad (25)$$

и затем итерационный процесс выполняется по рекуррентным формулам

$$\bar{x}^{(k+1)} = G\bar{x}^{(k)} + \bar{f}, \quad k = 0, 1, 2, \dots \quad (25^*)$$

Матрица G и вектор \bar{f} получены в результате преобразования системы (1).

Для сходимости (25*) необходимо и достаточно, чтобы $|\lambda_i(G)| < 1$, где $\lambda_i(G)$ – все собственные значения матрицы G . Сходимость будет также и в случае, если $\|G\| < 1$, ибо $|\lambda_i(G)| < \forall \|G\|$ (\forall – любой).

Символ $\| \dots \|$ означает норму матрицы. При определении ее величины чаще всего останавливаются на проверке двух условий:

$$\|G\| = \max_{1 \leq i \leq n} \sum_{j=1}^n |g_{ij}|, \quad \text{или} \quad \|G\| = \max_{1 \leq j \leq n} \sum_{i=1}^n |g_{ij}|, \quad (26)$$

где $G = \{g_{ij}\}_1^n$. Сходимость гарантирована также, если исходная матрица A имеет диагональное преобладание, т.е.

$$|a_{ii}| > \sum_{i,j=1; i \neq j}^n |a_{ij}|, \quad A = \{a_{ij}\}_1^n. \quad (27)$$

Если (26) или (27) выполняются, метод итерации сходится при любом начальном приближении $\bar{x}^{(0)}$. Чаще всего вектор $\bar{x}^{(0)}$ берут или нулевым, или единичным, или сам вектор \bar{f} из (25).

Имеется много подходов к преобразованию исходной системы (2) с матрицей A для обеспечения вида (25) или условий сходимости (26) и (27).

Например, (25) можно получить следующим образом.

Пусть $A = B + C$, $\det B \neq 0$;

тогда $(B+C) \bar{x} = \bar{b} \Rightarrow B \bar{x} = -C \bar{x} + \bar{b} \Rightarrow B^{-1} B \bar{x} = -B^{-1} C \bar{x} + B^{-1} \bar{b}$,

откуда $\bar{x} = -B^{-1} C \bar{x} + B^{-1} \bar{b}$.

Положив $-B^{-1} C = G$, $B^{-1} \bar{b} = \bar{f}$ и получим (25).

Из условий сходимости (26) и (27) видно, что представление $A = B + C$ не может быть произвольным.

Если матрица A удовлетворяет требованиям (27), то в качестве матрицы B можно выбрать нижнюю треугольную

$$B = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ & \cdots & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad a_{ii} \neq 0.$$

Или

$$\begin{aligned} A\bar{x} = \bar{b}; \quad \Rightarrow \quad A\bar{x} - \bar{b} = 0; \quad \Rightarrow \quad \bar{x} + (A\bar{x} - \bar{b}) = \bar{x}; \quad \Rightarrow \\ \bar{x} = \bar{x} + \alpha(A\bar{x} - \bar{b}) = \bar{x} + \alpha A\bar{x} - \alpha\bar{b} = (E + \alpha A)\bar{x} - \alpha\bar{b} = G\bar{x} + \bar{f}. \end{aligned}$$

Подбирая параметр α можно добиться, чтобы $\|G\| = \|E + \alpha A\| < 1$.

Если имеет место преобладание (27), тогда преобразование к (25) можно осуществить просто, решая каждое i -е уравнение системы (1) относительно x_i по следующим рекуррентным формулам:

$$\begin{aligned} x_i^k = -\frac{1}{a_{ii}} \left[\sum_{j=1; j \neq i}^n a_{ij} x_j^{k-1} - b_i \right] = \sum_{j=1}^n g_{ij} x_j^{k-1} + f_i; \\ g_{ij} = -a_{ij} / a_{ii}; \quad g_{ii} = 0; \quad f_i = b_i / a_{ii}, \end{aligned} \quad (27^*)$$

т.е. $G = \{g_{ij}\}_1^n$.

Если же в матрице A нет диагонального преобладания, его нужно добиться посредством каких-либо ее линейных преобразований, не нарушающих их равносильности.

Для примера рассмотрим систему

$$\begin{cases} 2x_1 - 1,8x_2 + 0,4x_3 = 1; & (I) \\ 3x_1 + 2x_2 - 1,1x_3 = 0; & (II) \\ x_1 - x_2 + 7,3x_3 = 0; & (III) \end{cases} \quad (28)$$

Как видно в уравнениях (I) и (II) нет диагонального преобладания, а в (III) есть, поэтому его оставляем неизменным.

Добьемся диагонального преобладания в уравнении (I). Умножим (I) на α , (II) на β , сложим оба уравнения и в полученном уравнении выберем α и β так, чтобы имело место диагональное преобладание:

$$(2\alpha + 3\beta) x_1 + (-1,8\alpha + 2\beta) x_2 + (0,4\alpha - 1,1\beta)x_3 = \alpha .$$

Взяв $\alpha = \beta = 5$, получим $25x_1 + x_2 - 3,5x_3 = 5$.

Для преобразования второго уравнения (II) с преобладанием, (I) умножим на γ , (II) умножим на δ , и из (II) вычтем (I). Получим

$$(3\delta - 2\gamma) x_1 + (2\delta + 1,8\gamma) x_2 + (-1,1\delta - 0,4\gamma)x_3 = -\gamma .$$

Положим $\delta = 2$, $\gamma = 3$, получим $0x_1 + 9,4x_2 - 3,4x_3 = -3$. В результате получим систему:

$$\begin{cases} 25x_1 + x_2 - 3,5x_3 = 5; \\ 9,4x_2 - 3,4x_3 = -3; \\ x_1 - x_2 + 7,3x_3 = 0. \end{cases} \quad (29)$$

Такой прием можно применять для широкого класса матриц.

Далее разделим в (29) каждое уравнение на диагональный элемент, получим

$$\begin{cases} x_1 + 0,04x_2 - 0,14x_3 = 0,2; \\ x_2 - 0,36x_3 = -0,32; \\ 0,14x_1 - 0,14x_2 + x_3 = 0. \end{cases} \quad \text{или} \quad \begin{cases} x_1 = -0,04x_2 + 0,14x_3 + 0,2; \\ x_2 = 0,36x_3 - 0,32; \\ x_3 = -0,14x_1 + 0,14x_2. \end{cases}$$

Взяв в качестве начального приближения, например, вектор $\bar{x}^{(0)} = (0,2; -0,32; 0)^T$. Будем решать эту систему по технологии (25*):

$$\begin{aligned} x_1^{(k+1)} &= -0,04x_2^{(k)} + 0,14x_3^{(k)} + 0,2; \\ x_2^{(k+1)} &= 0,36x_3^{(k)} - 0,32; \\ x_3^{(k+1)} &= -0,14x_1^{(k)} + 0,14x_2^{(k)}. \end{aligned} \quad k = 0, 1, 2, \dots$$

Процесс вычисления прекращается, когда два соседних приближения вектора решения совпадают по точности, т.е.

$$\left| \bar{x}^{(k+1)} - \bar{x}^{(k)} \right| < \varepsilon .$$

2.1. Метод простой итерации

2.1.1 Теоретическая часть

Технология итерационного решения вида (25*) названа методом *простой итерации*.

Оценка абсолютной погрешности для метода простой итерации

$$\left\| \bar{x}^* - \bar{x}^{(k+1)} \right\| \leq \|G\|^{k+1} \cdot \left\| \bar{x}^{(0)} \right\| + \frac{\|G\|^{k+1}}{1 - \|G\|} \cdot \left\| \bar{f} \right\|,$$

напомним, символ $\| \dots \|$ означает норму.

Пример 1. Методом простой итерации с точностью $\varepsilon=0,001$ решить систему линейных уравнений

$$\begin{cases} x_1 = 0,32x_1 - 0,05x_2 + 0,11x_3 - 0,08x_4 + 2,15; \\ x_2 = 0,11x_1 + 0,16x_2 - 0,28x_3 - 0,06x_4 - 0,83; \\ x_3 = 0,08x_1 - 0,15x_2 \quad \quad \quad + 0,12x_4 + 1,16; \\ x_4 = -0,21x_1 + 0,13x_2 - 0,27x_3 \quad \quad \quad + 0,44. \end{cases}$$

Число шагов, дающих ответ с точностью до $\varepsilon = 0,001$, можно определить из соотношения

$$\left\| \bar{x}^* - \bar{x}^{(k)} \right\| \leq \frac{\|G\|^{k+1}}{1 - \|G\|} \cdot \left\| \bar{f} \right\| \leq 0,001.$$

Оценим сходимость по формуле (26). Здесь $\|G\| = \max_{1 \leq i \leq 4} \sum_{j=1}^4 |g_{ij}| = \max\{0,56;$

$0,61; 0,35; 0,61\} = 0,61 < 1$; $\left\| \bar{f} \right\| = 2,15$. Значит, сходимость обеспечена.

В качестве начального приближения возьмем вектор свободных членов, т.е. $\bar{x}^{(0)} = (2,15; -0,83; 1,16; 0,44)^T$. Подставим значения вектора $\bar{x}^{(0)}$ в формулы (25*):

$$\begin{aligned} x_1^{(1)} &= 0,32 \cdot 2,15 + 0,05 \cdot 0,83 + 0,11 \cdot 1,16 - 0,08 \cdot 0,44 + 2,15 = 2,9719; \\ x_2^{(1)} &= 0,11 \cdot 2,15 - 0,16 \cdot 0,83 - 0,28 \cdot 1,16 - 0,06 \cdot 0,44 - 0,83 = -1,0775; \\ x_3^{(1)} &= 0,08 \cdot 2,15 + 0,15 \cdot 0,83 \quad \quad \quad + 0,12 \cdot 0,44 + 1,16 = 1,5093; \\ x_4^{(1)} &= -0,21 \cdot 2,15 - 0,13 \cdot 0,83 - 0,27 \cdot 1,16 \quad \quad \quad + 0,44 = -0,4326. \end{aligned}$$

Продолжая вычисления, результаты занесем в таблицу:

k	x_1	x_2	x_3	x_4
0	2,15	-0,83	1,16	0,44
1	2,9719	-1,0775	1,5093	-0,4326
2	3,3555	-1,0721	1,5075	-0,7317
3	3,5017	-1,0106	1,5015	-0,8111
4	3,5511	-0,9277	1,4944	-0,8321
5	3,5637	-0,9563	1,4834	-0,8298
6	3,5678	-0,9566	1,4890	-0,8332
7	3,5760	-0,9575	1,4889	-0,8356
8	3,5709	-0,9573	1,4890	-0,8362
9	3,5712	-0,9571	1,4889	-0,8364
10	3,5713	-0,9570	1,4890	-0,8364

Сходимость в тысячных долях имеет место уже на 10-м шаге.

Ответ: $x_1 \approx 3,571$; $x_2 \approx -0,957$; $x_3 \approx 1,489$; $x_4 \approx -0,836$.

Это решение может быть получено и с помощью формул (27*).

Пример 2. Для иллюстрации алгоритма с помощью формул (27*), рассмотрим решение системы (только две итерации):

$$\begin{cases} 4x_1 - x_2 - x_3 = 2; \\ x_1 + 5x_2 - 2x_3 = 4; \\ x_1 + x_2 + 4x_3 = 6; \end{cases} \quad A = \begin{bmatrix} 4 & -1 & -1 \\ 1 & 5 & -2 \\ 1 & 1 & 4 \end{bmatrix}; \quad \vec{b} = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}. \quad (30)$$

Преобразуем систему к виду (25) согласно (27*):

$$\begin{cases} x_1 = (2 + x_2 + x_3)/4; \\ x_2 = (4 - x_1 + 2x_3)/5; \\ x_3 = (6 - x_1 - x_2)/4; \end{cases} \Rightarrow \begin{cases} x_1^{(k+1)} = (2 + x_2^{(k)} + x_3^{(k)})/4; \\ x_2^{(k+1)} = (4 - x_1^{(k)} + 2x_3^{(k)})/5; \\ x_3^{(k+1)} = (6 - x_1^{(k)} - x_2^{(k)})/4. \end{cases} \quad (31)$$

Возьмем начальное приближение $\bar{x}^{(0)} = (0; 0; 0)^T$. Тогда для $k = 0$ очевидно, что значение $\bar{x}^{(1)} = (0,5; 0,8; 1,5)^T$. Подставим эти значения в (31), т.е. при $k=1$, получим $\bar{x}^{(2)} = (1,075; 1,3; 1,175)^T$.

$$\text{Ошибка } \varepsilon_2 = \max_{1 \leq i \leq 2} |x_i^{(2)} - x_i^{(1)}| = \max(0,575; 0,5; 0,325) = 0,575.$$

2.1.2 Практическая часть

Методом простых итераций с точностью $\varepsilon=0,01$ решить систему линейных алгебраических уравнений:

$$\begin{cases} 2x_1 + 2x_2 + 10x_3 = 14, \\ 10x_1 + x_2 + x_3 = 12, \\ 2x_1 + 10x_2 + x_3 = 13. \end{cases}$$

Так как $|2| < |2| + |10|$, $|1| < |10| + |1|$, $|1| < |2| + |10|$, то условие не выполняется. Переставим уравнения так, чтобы выполнялось условие преобладания диагональных элементов:

$$\begin{cases} 10x_1 + x_2 + x_3 = 12, \\ 2x_1 + 10x_2 + x_3 = 13, \\ 2x_1 + 2x_2 + 10x_3 = 14. \end{cases}$$

Получаем $|10| > |1| + |1|$, $|10| > |2| + |1|$, $|10| > |2| + |2|$. Выразим из первого уравнения x_1 , из второго x_2 , из третьего x_3 :

$$\begin{cases} x_1 = -0,1 \cdot x_2 - 0,1 \cdot x_3 + 1,2, \\ x_2 = -0,2 \cdot x_1 - 0,1 \cdot x_3 + 1,3, \\ x_3 = -0,2 \cdot x_1 - 0,2 \cdot x_2 + 1,4; \end{cases} \alpha = \begin{pmatrix} 0 & -0,1 & -0,1 \\ -0,2 & 0 & -0,1 \\ -0,2 & -0,2 & 0 \end{pmatrix}, \quad \beta = \begin{pmatrix} 1,2 \\ 1,3 \\ 1,4 \end{pmatrix}.$$

Заметим, что $\|\alpha\|_1 = \max\{0,2; 0,3; 0,4\} = 0,4 < 1$, следовательно, условие сходимости выполнено.

Зададим $x^{(0)} = \beta = \begin{pmatrix} 1,2 \\ 1,3 \\ 1,4 \end{pmatrix}$. В поставленной задаче $\varepsilon = 0,01$

Выполним расчеты по формуле

$$x^{(k+1)} = \begin{pmatrix} 0 & -0,1 & -0,1 \\ -0,2 & 0 & -0,1 \\ -0,2 & -0,2 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \end{pmatrix} + \begin{pmatrix} 1,2 \\ 1,3 \\ 1,4 \end{pmatrix}, \quad k = 0, 1, \dots \text{ или}$$

$$\begin{cases} x_1^{(k+1)} = -0,1 \cdot x_2^{(k)} - 0,1 \cdot x_3^{(k)} + 1,2, \\ x_2^{(k+1)} = -0,2 \cdot x_1^{(k)} - 0,1 \cdot x_3^{(k)} + 1,3, \\ x_3^{(k+1)} = -0,2 \cdot x_1^{(k)} - 0,2 \cdot x_2^{(k)} + 1,4. \end{cases}$$

до выполнения условия окончания и результаты занесем в табл.

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$\ x^{(k)} - x^{(k-1)}\ _1$
0	1,2000	1,3000	1,4000	—
1	0,9300	0,9200	0,900	0,5
2	1,0180	1,0240	1,0300	0,13
3	0,9946	0,9934	0,9916	0,0384
4	1,0015	1,0020	1,0024	0,0108
5	0,9996	0,9995	0,9993	0,0027 < ε

Расчет закончен, поскольку выполнено условие окончания $\|x^{(k+1)} - x^{(k)}\| = 0,0027 < \varepsilon = 0,01$.

Приведем результаты расчетов для другого начального приближения $x^0 = (1,2 \ 0 \ 0)$ и $\varepsilon = 0,001$

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$\ x^{(k)} - x^{(k-1)}\ _1$
0	1,2000	0	0	—
1	1,2000	1,0600	1,1600	1,1600
2	0,9780	0,9440	0,9480	0,2220
3	1,0108	1,0096	1,0156	0,0676
4	0,9975	0,9963	0,9959	0,0133
5	1,0008	0,0009	1,0012	0,0053
6	0,9998	0,9997	0,9997	0,0015
7	1,0001	1,0001	1,0001	0,0004 < ε

Приближенное решение задачи: $x^* \approx (1,0001 \ 1,0001 \ 1,0001)$

2.2. Метод Зейделя

2.2.1. Теоретическая часть

Данный метод является модификацией метода простой итерации и для системы (25) $\bar{x} = G\bar{x} + \bar{f}$ имеет следующую технологию

$$\begin{aligned}x_1^{(k+1)} &= g_{11}x_1^k + \dots + g_{1n}x_n^k + f_1; \\x_2^{(k+1)} &= g_{21}x_1^{(k+1)} + \dots + g_{2n}x_n^{(k)} + f_2; \\x_3^{(k+1)} &= g_{31}x_1^{(k+1)} + \dots + g_{3n}x_n^{(k)} + f_3; \\&\dots \\x_n^{(k+1)} &= g_{n1}x_1^{(k+1)} + \dots + g_{nn}x_n^{(k)} + f_n.\end{aligned}\tag{32}$$

Суть его состоит в том, что при вычислении очередного приближения $x_i^{(k)}$ ($2 \leq i \leq n$) в системе (32) и в формуле (27*), если имеет место соотношение (27), вместо $x_i^{k-1}, \dots, x_{i-1}^{k-1}$ используются уже вычисленные ранее x_1^k, \dots, x_{i-1}^k , т.е. (27*) преобразуется к виду

$$x_i^k = \sum_{j=1}^{i-1} g_{ij}x_j^k + \sum_{j=i+1}^n g_{ij}x_j^{k-1} + f_i, \quad i = 1, \dots, n.\tag{33}$$

Это позволяет ускорить сходимость итераций почти в два раза. Оценка точности аналогична методу простой итерации.

Пример. Методом Зейделя решить систему линейных уравнений с точностью $\varepsilon = 0,0001$, приводя ее к виду, удобному для итераций.

$$\begin{cases} 4.5x_1 - 1.8x_2 + 3.6x_3 = -1.7 & (I) \\ 3.1x_1 + 2.3x_2 - 1.2x_3 = 3.6 & (II) \\ 1.8x_1 + 2.5x_2 + 4.6x_3 = 2.2 & (III) \end{cases}\tag{34}$$

Условия (27) для системы не удовлетворяются, поэтому приведем ее к виду соответствующему данному требованию.

$$\begin{cases} 7.6x_1 + 0.5x_2 + 2.4x_3 = 1.9, & (I + II) \\ 2.2x_1 + 9.1x_2 + 4.4x_3 = 9.7, & (2III + II - I) \\ -1.3x_1 + 0.2x_2 + 5.8x_3 = -1.4, & (III - II) \end{cases} \quad (35)$$

$$\begin{cases} 10x_1 = 2.4x_1 - 0.5x_2 - 2.4x_3 + 1.9; \\ 10x_2 = -2.2x_1 + 0.9x_2 - 4.4x_3 + 9.7; \\ 10x_3 = 1.3x_1 - 0.2x_2 + 4.2x_3 - 1.4; \end{cases}$$

$$\begin{cases} x_1 = 0.24x_1 - 0.05x_2 - 0.24x_3 + 0.19; \\ x_2 = -0.22x_1 + 0.09x_2 - 0.44x_3 + 0.97; \\ x_3 = 0.13x_1 - 0.02x_2 + 0.42x_3 - 0.14. \end{cases}$$

Здесь $\|G\| = \max_{1 \leq i \leq n} \sum_{j=1}^n |g_{ij}| = \max\{0.53; 0.75; 0.57\} = 0.75 < 1$, значит, процесс

Зейделя сходится.

По технологии счета (32) $\overline{x^0} = \{0.19; 0.97; -0.14\}$.

$$x_1^{(1)} = 0.24 * 0.19 - 0.05 * 0.97 + 0.24 * 0.14 + 0.19 = 0.2207;$$

$$x_2^{(1)} = -0.22 * 0.2207 + 0.09 * 0.97 + 0.44 * 0.14 + 0.97 = 1.0703;$$

$$x_3^{(1)} = 0.13 * 0.2207 - 0.02 * 1.0703 - 0.42 * 0.14 - 0.14 = -0.1915;$$

k	x_1	x_2	x_3	k	x_1	x_2	x_3
0	0.19	0.97	-0.14	5	0.2467	1.1135	-0.2237
1	0.2207	1.0703	-0.1915	6	0.2472	1.1143	-0.2241
2	0.2354	1.0988	-0.2118	7	0.2474	1.1145	-0.2243
3	0.2424	1.1088	-0.2196	8	0.2475	1.1145	-0.2243
4	0.2454	1.1124	-0.2226				

Ответ: $x_1 = 0.248$; $x_2 = 1.115$; $x_3 = -0.224$.

Замечание. Если для одной и той же системы методы простой итерации и Зейделя сходятся, то метод Зейделя предпочтительнее. Однако на практике имеет место ситуация, когда области сходимости этих методов могут быть различными, т.е. метод простой итерации сходится, а метод

Зейделя расходится и наоборот. Для обоих методов, если $\|G\|$ близка к единице, скорость сходимости очень малая.

2.2.2 Практическая часть

Методом простых итераций с точностью $\varepsilon=0,01$ решить систему линейных алгебраических уравнений:

$$\begin{cases} 2x_1 + 2x_2 + 10x_3 = 14, \\ 10x_1 + x_2 + x_3 = 12, \\ 2x_1 + 10x_2 + x_3 = 13. \end{cases}$$

Так как $|2| < |2| + |10|$, $|1| < |10| + |1|$, $|1| < |2| + |10|$, то условие не выполняется. Переставим уравнения так, чтобы выполнялось условие преобладания диагональных элементов:

$$\begin{cases} 10x_1 + x_2 + x_3 = 12, \\ 2x_1 + 10x_2 + x_3 = 13, \\ 2x_1 + 2x_2 + 10x_3 = 14. \end{cases}$$

Получаем $|10| > |1| + |1|$, $|10| > |2| + |1|$, $|10| > |2| + |2|$. Выразим из первого уравнения x_1 , из второго x_2 , из третьего x_3 :

$$\begin{cases} x_1 = -0,1 \cdot x_2 - 0,1 \cdot x_3 + 1,2, \\ x_2 = -0,2 \cdot x_1 - 0,1 \cdot x_3 + 1,3, \\ x_3 = -0,2 \cdot x_1 - 0,2 \cdot x_2 + 1,4; \end{cases} \alpha = \begin{pmatrix} 0 & -0,1 & -0,1 \\ -0,2 & 0 & -0,1 \\ -0,2 & -0,2 & 0 \end{pmatrix}, \quad \beta = \begin{pmatrix} 1,2 \\ 1,3 \\ 1,4 \end{pmatrix}.$$

Заметим, что $\|\alpha\|_1 = \max\{0,2; 0,3; 0,4\} = 0,4 < 1$, следовательно, условие сходимости выполнено.

Зададим $x^{(0)} = (1,2 \ 0 \ 0)$. В поставленной задаче $\varepsilon = 0,001$

Выполним расчеты по формуле

$$\begin{cases} x_1^{(k+1)} = -0,1 \cdot x_2^{(k)} - 0,1 \cdot x_3^{(k)} + 1,2, \\ x_2^{(k+1)} = -0,2 \cdot x_1^{(k+1)} - 0,1 \cdot x_3^{(k)} + 1,3, \ (k = 0, 1, \dots) \\ x_3^{(k+1)} = -0,2 \cdot x_1^{(k+1)} - 0,2 \cdot x_2^{(k+1)} + 1,4, \end{cases}$$

и результаты занесем в таблице.

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$\ x^{(k)} - x^{(k-1)}\ _1$
0	1,2000	0	0	—
1	1,2000	1,0600	0,9480	1,0600
2	0,9992	1,0054	0,9991	0,1008
3	0,9996	1,0002	1,0000	0,0052
4	1,0000	1,0000	1,0000	$0,0004 < \varepsilon$

Очевидно, найденное решение $x = (1 \ 1 \ 1)$ является точным.

Расчет завершен, поскольку выполнено условие окончания $\|x^{(k+1)} - x^{(k)}\| = 0,0004 < \varepsilon = 0,001$

ЗАКЛЮЧЕНИЕ

В своей работе было рассмотрены различные методы решения СЛАУ. Так же и реализован каждый из вышеописанных методов на языке С#.

Вся работа позволила узнать и научиться пользоваться несколькими методами для вычисления систем линейных уравнений, а также были применены в практике различные методы программирования для реализации каждого алгоритма.

СПИСОК ЛИТЕРАТУРЫ

1. Абаффи Й., Спедикато Э. Математические методы для линейных и нелинейных уравнений: проекционные ABSалгоритмы. — М.: Мир, 1996.
2. Годунов С. К. Современные аспекты линейной алгебры. — Новосибирск: Научн. книга, 1997.
3. Голуб Дж., Ван Лоун Ч. Матричные вычисления. — М.: Мир, 1999.
4. Ильин В. П. Методы неполной факторизации для решения линейных систем. — М.: Физматлит, 1995.
5. Ортега Дж. Введение в параллельные и векторные методы решения линейных систем. — М.: Мир, 1991.
6. Алексеева МЮ, Практическое применение элементов арт-терапии в работе учителя - М., 2016-88 с
7. Бадалян Л О. Невропатология М.: «Академия», 2012 -317 с.
8. Бетенски М Что ты видишь? Новые методы арт-терапии. Серия «Ступени психотерапии». СПб. Эксмо-пресс, 2012 -250 с.
9. Богатеева З. А. Чудесные поделки из бумаги. М. «Просвещение», 2011.-208 с.
10. Ванюхина Г. А Воспитание фонетико-фонематического восприятия. // Логопед 2007 №4 - С. 4-14.

ПРИЛОЖЕНИЯ

1. Реализация вспомогательного класса Vector

```
public class Vector
{
    private double[] data;           //Коэффициенты вектора
    private int count;               //Количество элементов в векторе
    public int Count { get => this.count; }
    public double Norm { get => FindNorm(); }

    public Vector(int count)
    {
        this.count = count;
        data = new double[count];
    }

    public Vector(double[] initArray)
    {
        data = new double[initArray.Length];
        for (int i = 0; i < data.Length; i++)
            data[i] = initArray[i];
        count = data.Length;
    }

    public Vector(Matrix initArray)
    {
        if (initArray.Columns == 1)
        {
            double[] temp = new double[initArray.Rows];
            for (int i = 0; i < initArray.Rows; i++)
                temp[i] = initArray[i, 0];
            data = (double[])temp.Clone();
            count = data.Length;
        }
    }

    /// Копирование вектора
    public Vector Copy()
    {
        Vector v = new Vector(data);
        return v;
    }

    /// Обращение к элементу вектора
    /// <param name="row">Индекс элемента</param>
    public double this[int row]
    {
        get { return data[row]; }
        set { data[row] = value; }
    }

    /// Вычитание двух векторов
    /// <param name="left">Первый вектор</param>
    /// <param name="right">Второй вектор</param>
    public static Vector operator -(Vector left, Vector right)
    {
        Vector v = new Vector(left.count);
        for (int i = 0; i < left.count; i++)
            v[i] = left[i] - right[i];
        return v;
    }
}
```

```

/// Сложение двух векторов
/// <param name="left">Первый вектор</param>
/// <param name="right">Второй вектор</param>
public static Vector operator +(Vector left, Vector right)
{
    Vector v = new Vector(left.count);
    for (int i = 0; i < left.count; i++)
        v[i] = left[i] + right[i];
    return v;
}

/// Обмен элементов
/// <param name="i">Индекс первого элемента</param>
/// <param name="j">Индекс второго элемента</param>
public void Swap(int i, int j)
{
    if (i == j) return;
    double temp = data[i];
    data[i] = data[j];
    data[j] = temp;
}

/// Вывод вектора
public void Print()
{
    for (int i = 0; i < count; i++)
        Console.WriteLine("x" + (i + 1) + " = " + data[i]);
    Console.WriteLine();
}

/// Поиск нормы вектора
private double FindNorm()
{
    double max = double.MinValue;
    for (int i = 0; i < Count; i++)
        max = Math.Max(max, Math.Abs(data[i]));

    return max;
}
}

```

2. Реализация вспомогательного класса Matrix

```
public class Matrix
{
    private double[,] data; //Значения матрицы
    private double precalculatedDeterminant = double.NaN; //Определитель

    private int rows; //Кол-во строк
    private int columns; //Кол-во столбцов
    public int Rows { get => this.rows; }
    public int Columns { get => this.columns; }
    public double[,] Data { get => this.data; }

    //Проверка на квадратность матрицы
    public bool IsSquare { get => this.Rows == this.Columns; }
    //Проверка на трехдиагональность матрицы
    public bool IsTridiagonal { get => CheckTridiagonalMatrix(); }
    //Проверка на симметричность
    public bool IsSymmetry { get => CheckSymmetryMatrix(); }
    //Горизонтальная норма
    public double NormColumn { get => CalculateNormColumn(); }
    //Вертикальная норма
    public double NormRow { get => CalculateNormRow(); }

    public Matrix(int rows, int columns)
    {
        this.rows = rows;
        this.columns = columns;
        this.data = new double[rows, columns];
        this.ProcessFunctionOverData((i, j) => this.data[i, j] = 0);
    }

    public Matrix(double[,] initArray)
    {
        this.rows = initArray.GetLength(0);
        this.columns = initArray.GetLength(1);
        this.data = new double[this.rows, this.columns];
        for(int i = 0; i < rows; i++)
            for (int j = 0; j < columns; j++)
                this.data[i, j] = initArray[i, j];
    }

    /// Функция Обработки Данных
    /// <param name="func">Операция для изменения элементов</param>
    public void ProcessFunctionOverData(Action<int, int> func)
    {
        for (var i = 0; i < this.Rows; i++)
            for (var j = 0; j < this.Columns; j++)
                func(i, j);
    }
}
```

```

/// Обращение к элементу матрицы
/// <param name="x">Строка</param>
/// <param name="y">Столбец</param>
public double this[int x, int y]
{
    get
    {
        return this.data[x, y];
    }
    set
    {
        this.data[x, y] = value;
        this.precalculatedDeterminant = double.NaN;
    }
}

/// Сложение двух матриц
/// <param name="matrixOne">Первая матрица</param>
/// <param name="matrixTwo">Вторая матрица</param>
public static Matrix operator +(Matrix matrixOne, Matrix matrixTwo)
{
    if (matrixOne.Rows != matrixTwo.Rows || matrixOne.Columns !=
matrixTwo.Columns)
        throw new ArgumentException("Эти матрицы не могут быть сложены");

    var result = new Matrix(matrixOne.Rows, matrixOne.Columns);

    result.ProcessFunctionOverData((i, j)
        => result[i, j] = matrixOne[i, j] + matrixTwo[i, j]);

    return result;
}

/// Вычитание двух матриц
/// <param name="matrix1">Первая матрица</param>
/// <param name="matrix2">Вторая матрица</param>
public static Matrix operator -(Matrix matrix1, Matrix matrix2)
{
    return matrix1 + (matrix2 * -1);
}

/// Операция умножения матрицы на число
/// <param name="matrix">Матрица</param>
/// <param name="number">Число</param>
public static Matrix operator *(Matrix matrix, double number)
{
    var result = new Matrix(matrix.Rows, matrix.Columns);
    result.ProcessFunctionOverData((i, j) => result[i, j] = matrix[i, j] *
number);
    return result;
}

```

```

/// Операция умножения двух матриц
/// <param name="matrixOne">Первая матрица</param>
/// <param name="matrixTwo">Вторая матрица</param>
public static Matrix operator *(Matrix matrixOne, Matrix matrixTwo)
{
    if (matrixOne.Columns != matrixTwo.Rows)
        throw new ArgumentException("Эти матрицы не могут быть умножены");

    var result = new Matrix(matrixOne.Rows, matrixTwo.Columns);
    result.ProcessFunctionOverData((i, j) =>
    {
        for (var k = 0; k < matrixOne.Columns; k++)
        {
            result[i, j] += matrixOne[i, k] * matrixTwo[k, j];
        }
    });
    return result;
}

/// Создание единичной матрицы
/// <param name="size">Размер матрицы</param>
public static Matrix CreateIdentityMatrix(int size)
{
    var result = new Matrix(size, size);
    for (var i = 0; i < size; i++)
        result[i, i] = 1;

    return result;
}

/// Транспонирование матрицы
public Matrix CreateTransposeMatrix()
{
    var result = new Matrix(this.Columns, this.Rows);
    result.ProcessFunctionOverData((i, j) => result[i, j] = this[j, i]);
    return result;
}

/// Определитель
public double CalculateDeterminant()
{
    if (!double.IsNaN(this.precalculatedDeterminant))
        return this.precalculatedDeterminant;

    if (!this.IsSquare)
        throw new InvalidOperationException(
            "Определитель существует только у квадратных матриц");

    if (this.Columns == 1)
        return this[0, 0];
    if (this.Columns == 2)
        return this[0, 0] * this[1, 1] - this[0, 1] * this[1, 0];

    double result = 0;
    for (var j = 0; j < this.Columns; j++)
        result += (j % 2 == 1 ? 1 : -1) * this[1, j] *

this.CreateMatrixWithoutColumn(j).CreateMatrixWithoutRow(1).CalculateDeterminant();

    this.precalculatedDeterminant = result;
    return result;
}

```

```

/// Обратная матрица
public Matrix CreateInvertibleMatrix()
{
    if (this.Rows != this.Columns)
        return null;
    double determinant = CalculateDeterminant();
    if (Math.Abs(determinant) < Constants.DoubleComparisonDelta)
        return null;

    Matrix result = new Matrix(Rows, Rows);
    ProcessFunctionOverData((i, j) =>
    {
        result[i, j] = ((i + j) % 2 == 1 ? -1 : 1)
            * CalculateMinor(i, j) / determinant;
    });

    result = result.CreateTransposeMatrix();
    return result;
}

/// Подсчет минора
/// <param name="i"></param>
/// <param name="j"></param>
private double CalculateMinor(int i, int j)
{
    return
CreateMatrixWithoutColumn(j).CreateMatrixWithoutRow(i).CalculateDeterminant();
}

/// Создать матрицу без строки
/// <param name="row">Строка</param>
private Matrix CreateMatrixWithoutRow(int row)
{
    if (row < 0 || row >= this.Rows)
        throw new ArgumentException("Строка не существует");

    var result = new Matrix(this.Rows - 1, this.Columns);
    result.ProcessFunctionOverData((i, j) => result[i, j] = i < row ? this[i, j]
: this[i + 1, j]);
    return result;
}

/// Создать матрицу без столбца
/// <param name="row">Столбец</param>
private Matrix CreateMatrixWithoutColumn(int column)
{
    if (column < 0 || column >= this.Columns)
        throw new ArgumentException("Столбец не существует");
    var result = new Matrix(this.Rows, this.Columns - 1);
    result.ProcessFunctionOverData((i, j) => result[i, j] = j < column ?
this[i, j] : this[i, j +
1]);
    return result;
}

```



```

/// Вывод матрицы
public void Print()
{
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
            Console.Write("{0} ", data[i, j]);
        Console.WriteLine();
    }
}

/// Метод позволяющий заменить нужный столбец
/// <param name="vector">Вектор столбца</param>
/// <param name="index">Номер столбца</param>
public Matrix InsertColumn(Vector vector, int index)
{
    if(this.Columns != vector.Count)
        throw new ArgumentException("Расхождение с размерами");

    Matrix rez = new Matrix(this.data);
    for (int i = 0; i < rez.rows; i++)
        rez[i, index] = vector[i];
    return rez;
}

/// Копирование матрицы
public Matrix Copy()
{
    Matrix temp = new Matrix(this.Rows, this.Columns);
    for (int i = 0; i < this.Rows; i++)
        for (int j = 0; j < this.Columns; j++)
            temp[i, j] = this[i, j];

    return temp;
}

/// Основное условие сходимости итерационного метода
private bool BasicConditionIteratAlgo()
{
    double sum = 0;
    for (int i = 0; i < Rows; i++)
    {
        for (int j = 0; j < Columns; j++)
            if (i != j)
                sum += Math.Abs(this[i, j]);
        if (Math.Abs(this[i, i]) < sum)
            return false;
    }
    return true;
}

```

```

/// Проверка основного условия итерационного метода
/// <param name="matrix">Проверяемая матрица коэффициентов системы</param>
/// <param name="vector">Проверяемый вектор свободных членов системы</param>
public bool ChecIteratAlgo(Vector vector)
{
    int last = 1;
    do
    {
        last = Permutations.NextPermutation(this, vector, last);
        if (this.BasicConditionIteratAlgo())
            return true;
    } while (last != 1);

    return false;
}

/// Проверка на трехдиагональную матрицу
private bool CheckTridiagonalMatrix()
{
    for (int i = 0; i < Rows; i++)
    {
        for (int j = 0; j < Columns; j++)
        {
            if (i == j || i == j - 1 || i == j + 1)
                continue;

            if (this[i, j] != 0)
                return false;
        }
    }

    return true;
}

/// Проверка на симметричность матрицы
private bool CheckSymmetryMatrix()
{
    if (!this.IsSquare)
        return false;
    for (int i = 0; i < Rows; i++)
        for (int j = i; j < Columns; j++)
            if (this[i, j] != this[j, i])
                return false;

    return true;
}

/// Подсчет нормы по строкам
private double CalculateNormColum()
{
    double max = double.MinValue;
    for (int i = 0; i < Rows; i++)
    {
        double temp = 0;
        for (int j = 0; j < Columns; j++)
            temp += data[i, j];
        max = Math.Max(max, temp);
    }
    return max;
}

```

```

/// Подсчет нормы столбцам
private double CalculateNormRow()
{
    double max = double.MinValue;
    for (int i = 0; i < Rows; i++)
    {
        double temp = 0;
        for (int j = 0; j < Columns; j++)
            temp += data[j, i];
        max = Math.Max(max, temp);
    }
    return max;
}

/// Сортировка строк
/// <param name="matrix">Сортируемая матрица</param>
/// <param name="value">Вектор</param>
/// <param name="index">Индекс</param>
public static void SortRows(Matrix matrix, Vector value, int index)
{
    double maxElement = matrix[index, index];
    int maxElementIndex = index;
    for (int i = index + 1; i < matrix.Columns; i++)
    {
        if (matrix[i, index] > maxElement)
        {
            maxElement = matrix[i, index];
            maxElementIndex = i;
        }
    }

    if (maxElementIndex > index)
    {
        double temp;

        temp = value[maxElementIndex];
        value[maxElementIndex] = value[index];
        value[index] = temp;

        for (int i = 0; i < matrix.Rows; i++)
        {
            temp = matrix[maxElementIndex, i];
            matrix[maxElementIndex, i] = matrix[index, i];
            matrix[index, i] = temp;
        }
    }
}
}

```

3. Реализация вспомогательного класса Permutations

```
public class Permutations
{
    private static bool LEFT_TO_RIGHT = true;
    private static bool RIGHT_TO_LEFT = false;

    /// Служебная функция для нахождения положения наибольшего мобильного целого
числа
    private static int SearchArr(int[] a, int n, int mobile)
    {
        for (int i = 0; i < n; i++)
            if (a[i] == mobile)
                return i + 1;

        return 0;
    }

    /// Поиск наибольшего подвижного числа
    private static int GetMobile(int[] a, bool[] dir, int n)
    {
        int mobile_prev = 0, mobile = 0;

        for (int i = 0; i < n; i++)
        {
            if (dir[a[i] - 1] == RIGHT_TO_LEFT && i != 0)
                if (a[i] > a[i - 1] && a[i] > mobile_prev)
                {
                    mobile = a[i];
                    mobile_prev = mobile;
                }

            if (dir[a[i] - 1] == LEFT_TO_RIGHT && i != n - 1)
                if (a[i] > a[i + 1] && a[i] > mobile_prev)
                {
                    mobile = a[i];
                    mobile_prev = mobile;
                }
        }

        if (mobile == 0 && mobile_prev == 0)
            return 0;
        else
            return mobile;
    }
}
```

```

    /// Вывод одной перестановки
    public static void PrintOnePerm(int[] a, bool[] dir, int n, Matrix matrix, Vector
vector)
    {
        int mobile = GetMobile(a, dir, n);
        int pos = SearchArr(a, n, mobile);

        if (dir[a[pos] - 1] == RIGHT_TO_LEFT)
        {
            int temp = a[pos - 1];
            a[pos - 1] = a[pos - 2];
            a[pos - 2] = temp;
            matrix.SwapRows(pos - 1, pos - 2);
            vector.Swap(pos - 1, pos - 2);
        }
        else if (dir[a[pos] - 1] == LEFT_TO_RIGHT)
        {
            int temp = a[pos];
            a[pos] = a[pos - 1];
            a[pos - 1] = temp;
            matrix.SwapRows(pos, pos - 1);
            vector.Swap(pos, pos - 1);
        }

        for (int i = 0; i < n; i++)
        {
            if (a[i] > mobile)
            {
                if (dir[a[i] - 1] == LEFT_TO_RIGHT)
                    dir[a[i] - 1] = RIGHT_TO_LEFT;

                else if (dir[a[i] - 1] == RIGHT_TO_LEFT)
                    dir[a[i] - 1] = LEFT_TO_RIGHT;
            }
        }
    }
    /// Вывод следующей перестановки
    public static int NextPermutation(Matrix matrix, Vector vector, int last)
    {
        int n = vector.Count;
        if (last == FactFactor(n))
        {
            return 1;
        }

        int[] a = new int[n];
        bool[] dir = new bool[n];

        for (int i = 0; i < n; i++)
            a[i] = i + 1;

        for (int i = 0; i < n; i++)
            dir[i] = RIGHT_TO_LEFT;

        for (int i = last; i < FactFactor(n); i++)
            PrintOnePerm(a, dir, n, matrix, vector);
        last++;
        return last;
    }
}

```

```

/// Факториал числа
public static int FactFactor(int n)
{
    if (n < 0)
        return 0;
    if (n == 0)
        return 1;
    if (n == 1 || n == 2)
        return n;
    bool[] u = new bool[n + 1];
    List<Tuple<int, int>> p = new List<Tuple<int, int>>();
    for (int i = 2; i <= n; ++i)
        if (!u[i])
        {
            int k = n / i;
            int c = 0;
            while (k > 0)
            {
                c += k;
                k /= i;
            }

            p.Add(new Tuple<int, int>(i, c));

            int j = 2;
            while (i * j <= n)
            {
                u[i * j] = true;
                ++j;
            }
        }

    int r = 1;
    for (int i = p.Count() - 1; i >= 0; --i)
        r *= (int)Math.Pow(p[i].Item1, p[i].Item2);
    return r;
}
}

```

4. Реализация основного класса SLE(СЛАУ)

```
public class SLE
{
    private Matrix data;           //Матрица коэффициентов системы
    private Vector value;         //Вектор свободных членов

    public SLE(Matrix data, Vector value)
    {
        this.data = data;
        this.value = value;
    }

    /// Вывод системы
    public void PrintSystem()
    {
        for (int i = 0; i < data.Rows; i++)
        {
            for (int j = 0; j < data.Columns; j++)
            {
                Console.Write("{0, 3}x{1} ", j == 0 ? data[i, j] :
                                Math.Abs(data[i, j]), (j + 1));

                if (j != data.Columns - 1)
                    Console.Write("{0, 3} ", data[i, j + 1] >= 0 ? "+" : "-");
            }
            Console.WriteLine("= {0, 3}", value[i]);
        }
    }
}
```

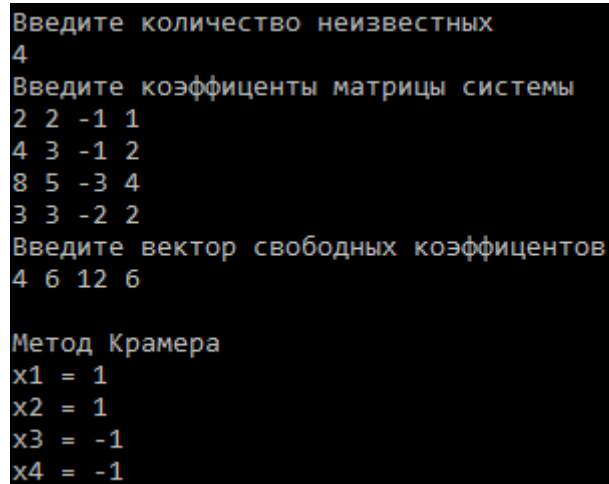
5. Метод Крамера

```
public void Kramer()  
{  
    if (!data.IsSquare) { Console.WriteLine("Матрица не квадратная" + '\n'); return; }  
  
    if (data.CalculateDeterminant() == 0)  
    {  
        Console.WriteLine("Матрица является вырожденной, метод Крамера  
                            не подходит для решения этого СЛАУ" +  
                            '\n');  
        return;  
    }  
  
    Vector x = new Vector(value.Count);  
  
    for (int i = 0; i < value.Count; i++)  
        x[i] = data.InsertColumn(value, i).CalculateDeterminant() /  
        data.CalculateDeterminant();  
  
    x.Print();  
}
```

Система

$$\begin{cases} 2x_1 + 2x_2 - x_3 + x_4 = 4 \\ 4x_1 + 3x_2 - x_3 + 2x_4 = 6 \\ 8x_1 + 5x_2 - 3x_3 + 4x_4 = 12 \\ 3x_1 + 3x_2 - 2x_3 + 2x_4 = 6 \end{cases} \quad \begin{pmatrix} 2 & 2 & -1 & 1 \\ 4 & 3 & -1 & 2 \\ 8 & 5 & -3 & 4 \\ 3 & 3 & -2 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \\ 12 \\ 6 \end{pmatrix}$$

Результат работы программы



```
Введите количество неизвестных  
4  
Введите коэффициенты матрицы системы  
2 2 -1 1  
4 3 -1 2  
8 5 -3 4  
3 3 -2 2  
Введите вектор свободных коэффициентов  
4 6 12 6  
  
Метод Крамера  
x1 = 1  
x2 = 1  
x3 = -1  
x4 = -1
```


6. Метод обратной матрицы

```
public void InvertibleMatrix()  
{  
    if (!data.IsSquare) { Console.WriteLine("Матрица не квадратная" + '\n'); return; }  
    (data.CreateInvertibleMatrix() * value).Print();  
}
```

Система

$$\begin{cases} 2x_1 + 2x_2 - x_3 + x_4 = 4 \\ 4x_1 + 3x_2 - x_3 + 2x_4 = 6 \\ 8x_1 + 5x_2 - 3x_3 + 4x_4 = 12 \\ 3x_1 + 3x_2 - 2x_3 + 2x_4 = 6 \end{cases} \quad \begin{pmatrix} 2 & 2 & -1 & 1 \\ 4 & 3 & -1 & 2 \\ 8 & 5 & -3 & 4 \\ 3 & 3 & -2 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \\ 12 \\ 6 \end{pmatrix}$$

Результат работы программы

```
Введите количество неизвестных  
4  
Введите коэффициенты матрицы системы  
2 2 -1 1  
4 3 -1 2  
8 5 -3 4  
3 3 -2 2  
Введите вектор свободных коэффициентов  
4 6 12 6  
  
Метод обратной матрицы  
x1 = 1  
x2 = 1  
x3 = -1  
x4 = -1
```

7. Метод Гаусса

```
public void Gauss()
{
    if (!data.IsSquare) { Console.WriteLine("Матрица не квадратная" + '\n'); return; }
    Matrix gaussMatrix = data.Copy();
    Vector gaussVector = value.Copy();

    for (int i = 0; i < gaussMatrix.Columns - 1; i++)
    {
        Matrix.SortRows(gaussMatrix, gaussVector, i);
        for (int j = i + 1; j < gaussMatrix.Columns; j++)
        {
            if (gaussMatrix[i, i] != 0)
            {
                double multElement = gaussMatrix[j, i] / gaussMatrix[i, i];
                for (int k = i; k < gaussMatrix.Columns; k++)
                {
                    gaussMatrix[j, k] -= gaussMatrix[i, k] * multElement;
                    gaussVector[j] -= gaussVector[i] * multElement;
                }
            }
        }
    }
    Vector x = new Vector(gaussMatrix.Rows);
    for (int i = (int)(gaussMatrix.Columns - 1); i >= 0; i--)
    {
        x[i] = gaussVector[i];
        for (int j = (int)(gaussMatrix.Columns - 1); j > i; j--)
            x[i] -= gaussMatrix[i, j] * x[j];

        if (gaussMatrix[i, i] == 0 && gaussVector[i] != 0)
        {
            Console.WriteLine("Решений нет");
            return;
        }
        x[i] /= gaussMatrix[i, i];
    }
    x.Print();
}
```

Система

$$\begin{cases} 2x_1 + 2x_2 - x_3 + x_4 = 4 \\ 4x_1 + 3x_2 - x_3 + 2x_4 = 6 \\ 8x_1 + 5x_2 - 3x_3 + 4x_4 = 12 \\ 3x_1 + 3x_2 - 2x_3 + 2x_4 = 6 \end{cases} \quad \begin{pmatrix} 2 & 2 & -1 & 1 \\ 4 & 3 & -1 & 2 \\ 8 & 5 & -3 & 4 \\ 3 & 3 & -2 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \\ 12 \\ 6 \end{pmatrix}$$

Результат работы программы

```
Введите количество неизвестных
4
Введите коэффициенты матрицы системы
2 2 -1 1
4 3 -1 2
8 5 -3 4
3 3 -2 2
Введите вектор свободных коэффициентов
4 6 12 6

Метод Гаусса
x1 = 1
x2 = 1
x3 = -1
x4 = -1
```

8. Метод прогонки

```
public void TridiagonalMatrixAlgorithm()
{
    if (!data.IsSquare) { Console.WriteLine("Матрица не квадратная" + '\n'); return; }
    if (!data.IsTridiagonal) { Console.WriteLine("Матрица не трехдиагональная" + '\n');
return; }
    double[] A = new double[data.Rows]; double[] B = new double[data.Rows];
    double[] C = new double[data.Rows]; double[] D = new double[data.Rows];

    for (int i = 0; i < data.Rows; i++)
        B[i] = data[i, i];
    A[0] = 0;
    for (int i = 1; i < data.Rows; i++)
        A[i] = data[i, i - 1];

    C[data.Rows - 1] = 0;
    for (int i = 0; i < data.Rows - 1; i++)
        C[i] = data[i, i + 1];
    for (int i = 0; i < value.Count; i++)
        D[i] = value[i];
    double[] alpha = new double[data.Rows];
    double[] beta = new double[data.Rows];
    alpha[0] = -C[0] / B[0];
    beta[0] = D[0] / B[0];

    for (int i = 1; i <= data.Rows - 1; i++)
        alpha[i] = (-C[i]) / (B[i] + A[i] * alpha[i - 1]);
    for (int i = 1; i <= data.Rows - 1; i++)
        beta[i] = (D[i] - A[i] * beta[i - 1]) / (B[i] + A[i] * alpha[i - 1]);
    Vector x = new Vector(data.Rows);
    x[data.Rows - 1] = beta[data.Rows - 1];
    for (int i = data.Rows - 2; i >= 0; i--)
        x[i] = alpha[i] * x[i + 1] + beta[i];

    x.Print();
}
```

Система

$$\begin{cases} 3x_1 - 2x_2 = 13 \\ x_1 + 4x_2 - 2x_3 = -9 \\ 3x_2 + 8x_3 - 4x_4 = 26 \\ 1x_4 = -4 \end{cases} \quad \begin{pmatrix} 3 & -2 & 0 & 0 \\ 1 & 4 & -2 & 0 \\ 0 & 3 & 8 & -4 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 13 \\ -9 \\ 26 \\ -4 \end{pmatrix}$$

Результат работы программы

```
Введите количество неизвестных
4
Введите коэффициенты матрицы системы
3 -2 0 0
1 4 -2 0
0 3 8 -4
0 0 0 1
Введите вектор свободных коэффициентов
13 -9 26 -4

Метод прогонки
x1 = 3
x2 = -2
x3 = 2
x4 = -4
```

9. Метод квадратных корней

```
public void CholeskyDecomposition()
{
    if (!data.IsSquare) { Console.WriteLine("Матрица не квадратная" + '\n'); return; }
    if (!data.IsSymmetry) { Console.WriteLine("Матрица не симметричная" + '\n'); return; }
}

Matrix u = new Matrix(data.Rows, data.Columns);
Vector x = new Vector(data.Rows);
Vector y = new Vector(data.Rows);

u[0, 0] = Math.Sqrt(data[0, 0]);
for (int i = 1; i < data.Rows; i++)
    u[0, i] = data[0, i] / u[0, 0];

for (int i = 1; i < data.Rows; i++)
{
    for (int j = 0; j < data.Columns; j++)
    {
        if (i == j)
        {
            double sum = 0;
            for (int k = 0; k < data.Rows; k++)
                sum += u[k, i] * u[k, i];

            u[i, j] = Math.Sqrt(data[i, j] - sum);
        }
        else if (i < j)
        {
            double sum = 0;
            for (int k = 0; k < i; k++)
                sum += u[k, i] * u[k, j];
            u[i, j] = (data[i, j] - sum) / u[i, i];
        }
    }
}

Matrix uT = u.CreateTransposeMatrix();
double temp = 0;

for (int i = 0; i < data.Rows; i++)
{
    temp = 0;
    for (int j = 0; j < i; j++)
        temp += uT[i, j] * y[j];

    y[i] = (value[i] - temp) / uT[i, i];
}

for (int i = data.Rows - 1; i >= 0; i--)
{
    temp = 0;
    for (int j = data.Rows - 1; j > i; j--)
        temp += u[i, j] * x[j];

    x[i] = (y[i] - temp) / u[i, i];
}

x.Print();
}
```

Система

$$\begin{cases} 2x_1 + x_2 + 4x_3 = 16 \\ x_1 + x_2 + 3x_3 = 12 \\ 4x_1 + 3x_2 + 14x_3 = 52 \end{cases}$$

$$\begin{pmatrix} 2 & 1 & 4 \\ 1 & 1 & 3 \\ 4 & 3 & 14 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 16 \\ 12 \\ 52 \end{pmatrix}$$

Результат работы программы

```
Введите количество неизвестных
3
Введите коэффициенты матрицы системы
2 1 4
1 1 3
4 3 14
Введите вектор свободных коэффициентов
16 12 52

Метод квадратных корней
x1 = 0,9999999999999999
x2 = 2
x3 = 3
```

10. Метод простых итераций

```
public void Itera(double eps)
{
    if (!data.IsSquare) { Console.WriteLine("Матрица не квадратная" + '\n'); return; }

    Matrix initMatrix = data.Copy();
    Vector initValue = value.Copy();
    if (!initMatrix.ChecIteratAlgo(initValue))
    { Console.WriteLine("Метод неподходит для решение данной системы" + '\n'); return; }

    Matrix alpha = initMatrix.Copy();
    Vector x = new Vector(initMatrix.Rows);
    Vector x0 = initValue.Copy();
    int count = 0;

    for (int i = 0; i < initMatrix.Rows; i++)
    {
        for (int j = 0; j < initMatrix.Columns; j++)
            if (i != j)
                alpha[i, j] /= -alpha[i, i];
        x0[i] /= alpha[i, i];
        alpha[i, i] = 0;
    }

    if (alpha.NormColumn >= 1 || alpha.NormRow >= 1)
    { Console.WriteLine("Метод неподходит для решение данной системы" + '\n'); return; }

    Vector beta = x0.Copy();
    do
    {
        Vector temp = x.Copy();
        x = alpha * x0 + beta;
        x0 = temp.Copy();
        count++;
    } while ((x0 - x).Norm >= eps);

    Console.WriteLine("Кол-во итераций: {0}", count);
    x.Print();
}
```

Система

$$\begin{cases} 10x_1 + x_2 + 4x_3 = 12 \\ 2x_1 + 2x_2 + 10x_3 = 13 \\ 2x_1 + 10x_2 + x_3 = 14 \end{cases} \quad \begin{pmatrix} 10 & 1 & 1 \\ 2 & 2 & 10 \\ 2 & 10 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 12 \\ 13 \\ 14 \end{pmatrix}$$

Результат работы программы

```
Введите количество неизвестных
3
Введите коэффициенты матрицы системы
10 1 1
2 2 10
2 10 1
Введите вектор свободных коэффициентов
12 13 14

Метод простых итераций
Кол-во итераций: 12
x1 = 1,000622
x2 = 1,111508
x3 = 0,876692
```

11. Метод Зейделя

```
public void Seidel(double eps)
{
    if (!data.IsSquare) { Console.WriteLine("Матрица не квадратная" + '\n'); return; }

    Matrix initMatrix = data.Copy();
    Vector initValue = value.Copy();
    if (!initMatrix.ChecIteratAlgo(initValue))
    { Console.WriteLine("Метод не подходит для решение данной системы" + '\n'); ; return; }
}

Matrix L = new Matrix(initMatrix.Rows, initMatrix.Columns);
Matrix U = new Matrix(data.Rows, data.Columns);
Matrix E = Matrix.CreateIdentityMatrix(data.Rows);

Matrix alpha = initMatrix.Copy();
Vector x = new Vector(initMatrix.Rows);
Vector x0 = initValue.Copy();
int count = 0;

for (int i = 0; i < initMatrix.Rows; i++)
{
    for (int j = 0; j < initMatrix.Columns; j++)
        if (i != j)
            alpha[i, j] /= -alpha[i, i];
    x0[i] /= alpha[i, i];
    alpha[i, i] = 0;
}

if (alpha.NormColum >= 1 || alpha.NormRow >= 1)
{ Console.WriteLine("Метод не подходит для решение данной системы" + '\n'); return; }

for (int i = 0; i < initMatrix.Rows; i++)
{
    for (int j = 0; j < initMatrix.Columns; j++)
    {
        if (i <= j)
            U[i, j] = alpha[i, j];
        else
            L[i, j] = alpha[i, j];
    }
}

if (((E - L).CreateInvertibleMatrix() * U).NormColum >= 1 ||
    ((E - L).CreateInvertibleMatrix() * U).NormRow >= 1)
{ Console.WriteLine("Метод не подходит для решение данной системы" + '\n'); return; }

Vector beta = x0.Copy();
do
{
    Vector temp = x.Copy();
    x = ((E - L).CreateInvertibleMatrix() * U) * x + ((E -
L).CreateInvertibleMatrix()) * beta;
    x0 = temp.Copy();
    count++;
} while ((x0 - x).Norm >= eps);

Console.WriteLine("Кол-во итераций: {0}", count);
x.Print();
}
```

Система

$$\begin{cases} 10x_1 + x_2 + 4x_3 = 12 \\ 2x_1 + 2x_2 + 10x_3 = 13 \\ 2x_1 + 10x_2 + x_3 = 14 \end{cases}$$

$$\begin{pmatrix} 10 & 1 & 1 \\ 2 & 2 & 10 \\ 2 & 10 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 12 \\ 13 \\ 14 \end{pmatrix}$$

Результат работы программы

```
Введите количество неизвестных
3
Введите коэффициенты матрицы системы
10 1 1
2 2 10
2 10 1
Введите вектор свободных коэффициентов
12 13 14

Метод Зейделя
Кол-во итераций: 4
x1 = 1,0010349632
x2 = 1,11205094656
x3 = 0,877382818048
```