1) Алгоритм Гаусса

Алгоритм Гаусса используют для решения систем линейных уравнений. Напомним, что линейными уравнениями называются уравнения состоящие только из суммы одночленов первой степени. Например, ниже приведена система линейных уравнений с 4-мя неизвестными.

2x1 + 3x2 − x3 + x4 = 5

x1 − 5x3 − 3x4 = -5

-x2 + 4x3 + 2x4 = 2

x1 − 2x3 = -2

Решением этой системы является:

x1 = 0

x2 = 2

x3 = 1

x4 = 0

Данная система имеет ровно одно решение. В некоторых случаях системы линейных уравнений могут иметь больше одного решения или не иметь решений. Причём заметим, что если система имеет хотя бы 2 решения, то тогда она имеет бесконечно много решений, поскольку если эти 2 решения сложить и поделить пополам, то получится новое решение системы (проверьте это самостоятельно). Поэтому в принципе возможны 3 случая:

-система не имеет решений,
-система имеет ровно одно решение
-система имеет бесконечно много решений.

Причём, если количество уравнений больше количества переменных, то система заведомо либо имеет бесконечно много решений, либо не имеет решений.

Далее, введём некоторые обозначения. Пусть A*X = B – векторное уравнение линейной системы для приведённой выше матрица A, справа расположен столбец свободных членов B, а вектор X состоит из переменных (x1, x2, x3, x4)):

$$\begin{pmatrix} 2 & 3 & -1 & 1 \\ 1 & 0 & -5 & -3 \\ 0 & -1 & 4 & 2 \\ 1 & 0 & -2 & 0 \end{pmatrix} * \begin{pmatrix} x1 \\ x2 \\ x3 \\ x4 \end{pmatrix} = \begin{pmatrix} 5 \\ -5 \\ 2 \\ -2 \end{pmatrix}$$

Чтобы определить количество решений системы (да и, собственно, чтобы решить систему уравнений) необходимо посчитать определитель матрицы системы. Для этого мы приведём матрицу A к диагональному виду. Напомним, что матрица, приведённая к диагональному виду, имеет нули ниже и выше главной диагонали, а на самой диагонали она может иметь нули или отличные от нуля числа. Например, следующая матрица 4x4 (она никак не связана с матрицей A) имеет диагональный вид:

$$\begin{pmatrix} 7 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 31 \end{pmatrix}$$

Затем мы перемножим числа, стоящие на диагонали. Это произведение и будет определителем матрицы. Если определитель матрицы отличен от нуля, то матрица имеет ровно одно решение. Если же определитель матрицы равен нулю, то либо система имеет бесконечно много решений, либо она не имеет решений.

Отметим, что из курса линейной алгебры известно, что некоторые преобразования над матрицами не изменяют их определитель. К этим преобразованиям относится замена строки матрицы на сумму этой строки и некоторой другой строки, умноженной на число, отличное от нуля. С помощью таких преобразований мы будем постепенно изменять нашу матрицу, пока она не примет диагональный вид. Вместе с матрицей мы также будем изменять столбец свободных членов (это позволит нам получить решение системы). Теперь рассмотрим, какие конкретно строки нужно будет заменять и в каком порядке.

Пусть изначально матрица A и столбец свободных членов B имеют вид:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$

Вначале алгоритма мы находимся в первой строке и в первом столбце, по ходу алгоритма мы будем переходить к следующей строке и столбцу. То есть каждый раз будет иметь место переход (i, i) -> (i+1, i+1).

Мы должны выбрать строку, с помощью которой мы будем обнулять первый столбец. Необходимо для этой цели брать строку,

у которой элемент текущего столбца (в данный момент первого столбца) ненулевой. То есть если сейчас текущая строка нам не пододит, то можно выбрать одну из строк, которые ниже текущей и поменять местами с текущей строкой. При таком преобразовании знак определителя матрицы меняется на противоположный. Если подходящей строки не нашлось вовсе, то определитель системы равен нулю и она не имеет решений или имеет бесконечно много решений. Теперь с помощью элемента a11 мы будем обнулять все остальные элементы первого столбца.

Вначале заменим вторую строку матрицы на сумму второй строки и первой, умноженной на коэффициент k = -a21/a11. При этом элемент a21 обнуляется. Коэффициент для каждой ОБНУЛЯЕМОЙ строки свой – его нужно постоянно пересчитывать. Рассмотрим этот процесс на матрице A:

Изначально матрица имеет вид (последний столбец - столбец свободных членов):

$$\begin{pmatrix} 2 & 3 & -1 & 1 \\ 1 & 0 & -5 & -3 \\ 0 & -1 & 4 & 2 \\ 1 & 0 & -2 & 0 \end{pmatrix} \begin{pmatrix} 5 \\ -5 \\ 2 \\ -2 \end{pmatrix}$$

После замены второй строки: (мы домножили первую строку на -1/2 и сложили со второй)

$$\begin{pmatrix} 2 & 3 & -1 & 1 \\ 0 & -1.5 & -4.5 & -3.5 \\ 0 & -1 & 4 & 2 \\ 1 & 0 & -2 & 0 \end{pmatrix} \begin{pmatrix} 5 \\ -7.5 \\ 2 \\ -2 \end{pmatrix}$$

Третью строку заменять не нужно (так как первый элемент третьей строки итак равен нулю). После замены четвёртой строки матрица принимает вид:

$$\begin{pmatrix} 2 & 3 & -1 & 1 \\ 0 & -1.5 & -4.5 & -3.5 \\ 0 & -1 & 4 & 2 \\ 0 & -1.5 & -1.5 & -0.5 \end{pmatrix} \begin{pmatrix} 5 \\ -7.5 \\ 2 \\ -4.5 \end{pmatrix}$$

Первый этап алгоритма закончен. Теперь мы переходим ко второму столбцу и второй строке. Необходимо опять выбрать, с помощью которой строки мы будем обнулять элементы второго столбца. В данном случае вторая строка нам подходит, поэтому её не нужно менять местами ни с какой другой строкой. Если бы элемент a22 был сейчас равен нулю, то нужно было бы взять либо 3-ю, либо 4-ю строку (то есть ту, которая ниже второй) и поменять её местами со второй строкой. При этом преобразовании (поменять местами две строки) определитель матрицы меняет знак.

Теперь обнулим элемент первой строки и второго столбца (мы домножили вторую строку на (-1)*(3/-1.5) и сложили с первой строкой. Матрица принимает вид:

$$\begin{pmatrix} 2 & 0 & -10 & -6 \\ 0 & -1.5 & -4.5 & -3.5 \\ 0 & -1 & 4 & 2 \\ 0 & -1.5 & -1.5 & -0.5 \end{pmatrix} \begin{pmatrix} -10 \\ -7.5 \\ 2 \\ -4.5 \end{pmatrix}$$

Теперь с помощью второй строки мы будем обнулять третью (домножим вторую строку на (-1) * (-1/-1.5) и добавим это к третей строке):

$$\begin{pmatrix} 2 & 0 & -10 & -6 \\ 0 & -1.5 & -4.5 & -3.5 \\ 0 & 0 & 7 & 4.33333 \\ 0 & -1.5 & -1.5 & -0.5 \end{pmatrix} \begin{pmatrix} -10 \\ -7.5 \\ 7 \\ -4.5 \end{pmatrix}$$

После замены 4-ой строки (домножили вторую на (-1)*(-1.5/-1.5) и сложим с четвёртой):

$$\begin{pmatrix} 2 & 0 & -10 & -6 \\ 0 & -1.5 & -4.5 & -3.5 \\ 0 & 0 & 7 & 4.33333 \\ 0 & 0 & 3 & 3 \end{pmatrix} \begin{pmatrix} -10 \\ -7.5 \\ 7 \\ 3 \end{pmatrix}$$

Далее переходим к третей строке и третьему столбцу. Будем обнулять элементы третьего столбца. Третья строка нам подходит, поэтому её не нужно ни с кем менять местами. Обнуляем элемент первой строки, третьего столбца:

$$\begin{pmatrix} 2 & 0 & 0 & 0.190476 \\ 0 & -1.5 & -4.5 & -3.5 \\ 0 & 0 & 7 & 4.33333 \\ 0 & 0 & 3 & 3 \end{pmatrix} \begin{pmatrix} 0 \\ -7.5 \\ 7 \\ 3 \end{pmatrix}$$

Обнуляем элемент второй строки и третьего столбца:

$$\begin{pmatrix} 2 & 0 & 0 & 0.190476 \\ 0 & -1.5 & 0 & -0.714286 \\ 0 & 0 & 7 & 4.33333 \\ 0 & 0 & 3 & 3 \end{pmatrix} \begin{pmatrix} 0 \\ -3 \\ 7 \\ 3 \end{pmatrix}$$

Обнуляем элемент четвёртой строки и третьего столбца:

$$\begin{pmatrix} 2 & 0 & 0 & 0.190476 \\ 0 & -1.5 & 0 & -0.714286 \\ 0 & 0 & 7 & 4.33333 \\ 0 & 0 & 0 & 1.14286 \end{pmatrix} \begin{pmatrix} 0 \\ -3 \\ 7 \\ 0 \end{pmatrix}$$

Далее переходим на четвёртую строку и четвёртый столбец. Обнулим элементы четвёртого столбца. Теперь обнуляем элемент первой строки и четвертого столбца.

$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & -1.5 & 0 & -0.714286 \\ 0 & 0 & 7 & 4.33333 \\ 0 & 0 & 0 & 1.14286 \end{pmatrix} \begin{pmatrix} 0 \\ -3 \\ 7 \\ 0 \end{pmatrix}$$

Далее обнуляем элемент второй строки и четвёртого столбца:

$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & -1.5 & 0 & 0 \\ 0 & 0 & 7 & 4.33333 \\ 0 & 0 & 0 & 1.14286 \end{pmatrix} \begin{pmatrix} 0 \\ -3 \\ 7 \\ 0 \end{pmatrix}$$

Далее обнуляем элемент третьей строки и четвёртого столбца:

$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & -1.5 & 0 & 0 \\ 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 1.14286 \end{pmatrix} \begin{pmatrix} 0 \\ -3 \\ 7 \\ 0 \end{pmatrix}$$

Теперь матрица приняла диагональный вид. Легко найти определитель, перемножив элементы главной диагонали:

2 * -1.5 * 7 * 1.14286 = -24

Если определитель системы отличен от нуля, то можно найти решение системы. Полученную диагональную матрицу можно переписать в виде системы линейных уравнений:

2x1 = 0

-1.5x2 = -3

7x3 = 7

1.14286*x4 = 0

Таким образом, если мы возьмём столбец свободных членов и поделим его на элементы главной диагонали, то мы получаем:

x1 = 0
x2 = 2
x3 = 1
x4 = 0

Теперь приведём код Алгоритма Гаусса:

```
double gauss(vector<vector<double> >& M, int n, vector<double>& X)
{
        double zn = 1.;
    double eps = 1e-8;


    for(int i = 0; i < n; i++)
    { // vibor stroki s pomochu kotoroi mi obnuliaem
                if (fabs(M[i][i]) < eps)
                    for(int j = i+1; j < n; j++)
                                if (M[j][i] > M[i][i])
                                {// uchet znaka opredelitelia
                                zn *= -1.;
                                for(int k = 0; k < n; k++)
                                {
                                        swap(M[i][k], M[j][k]);
                                }
                                }

        if (fabs(M[i][i]) < eps)
            return 0.;

        for(int j = 0; j < n; j++)
        {
            if (fabs(M[j][i]) > eps && j != i)
            { // obnulenie
                double koef = -(M[j][i] / M[i][i]);
                for(int k = 0; k <= n; k++)
                    M[j][k] += M[i][k] * koef;
            }
        }
    }

        double ans = 1;
    for(int i = 0; i < n; i++)
        {
                ans *= M[i][i]; // vichislenie opredelitelia
                X[i] = M[i][n] / M[i][i]; // reshenie sistemi
        }

        return ans * zn;
}
```

2) Алгоритм Гаусса в случае бесконечно многих решений или отсутствия решений системы.

В рассмотренном алгоритме мы прерывали работу, если в некотором очередном столбце все элементы оказались равными нулю, то есть не нашлось строчки, с помощью которой можно обнулить что-либо. При этом определитель матрицы равен нулю. Это значит, что система уравнений либо имеет бесконечно много решений, либо не имеет решений. Однако, в некоторых задачах нам нужно уметь различать эти 2 случая (когда система не имеет решений и когда она имеет бесконечно много решений). Поставим задачу следующим образом: если система имеет 1 или бесконечно много решений, то нужно найти хотя бы одно решение, а если система не имеет решений, нужно вывести "no solution". В этом случае необходимо несколько изменить приведённый выше алгоритм.

Будем игнорировать столбцы матрицы, которые полностью обнулились. В этом случае вместо обычного перехода от строки i и столбца i к следующей строке и следующему столбцу (i, i) -> (i+1, i+1) будем оставаться на той же строке, и только переходить к следующему столбцу, переход (i, j) -> (i, j+1). При таком подходе матрица примет ступенчатый вид. Рассмотрим, например как будет изменяться следующая матрица (в последнем столбце находится столбец свободных членов):

Шаг 1:

$$\begin{pmatrix} 1 & 1.5 & -5 & -3 \\ 2 & 3 & -1 & 1 \\ -2 & -3 & 4 & 2 \\ 4 & 6 & -2 & 0 \end{pmatrix} \begin{pmatrix} -4 \\ 8 \\ -2 \\ 14 \end{pmatrix}$$

Шаг 2:

$$\begin{pmatrix} 1 & 1.5 & -5 & -3 \\ 0 & 0 & 9 & 7 \\ -2 & -3 & 4 & 2 \\ 4 & 6 & -2 & 0 \end{pmatrix} \begin{pmatrix} -4 \\ 16 \\ -2 \\ 14 \end{pmatrix}$$

Шаг 3:

$$\begin{pmatrix} 1 & 1.5 & -5 & -3 \\ 0 & 0 & 9 & 7 \\ 0 & 0 & -6 & -4 \\ 4 & 6 & -2 & 0 \end{pmatrix} \begin{pmatrix} -4 \\ 16 \\ -10 \\ 14 \end{pmatrix}$$

Шаг 4:

$$\begin{pmatrix} 1 & 1.5 & -5 & -3 \\ 0 & 0 & 9 & 7 \\ 0 & 0 & -6 & -4 \\ 0 & 0 & 18 & 12 \end{pmatrix} \begin{pmatrix} -4 \\ 16 \\ -10 \\ 30 \end{pmatrix}$$

Шаг 5:

$$\begin{pmatrix} 1 & 1.5 & 0 & 0.888889 \\ 0 & 0 & 9 & 7 \\ 0 & 0 & -6 & -4 \\ 0 & 0 & 18 & 12 \end{pmatrix} \begin{pmatrix} 4.88889 \\ 16 \\ -10 \\ 30 \end{pmatrix}$$

Шаг 6:

$$\begin{pmatrix} 1 & 1.5 & 0 & 0.888889 \\ 0 & 0 & 9 & 7 \\ 0 & 0 & 0 & 0.666667 \\ 0 & 0 & 18 & 12 \end{pmatrix} \begin{pmatrix} 4.88889 \\ 16 \\ 0.666667 \\ 30 \end{pmatrix}$$

Шаг 7:

$$\begin{pmatrix} 1 & 1.5 & 0 & 0.888889 \\ 0 & 0 & 9 & 7 \\ 0 & 0 & 0 & 0.666667 \\ 0 & 0 & 0 & -2 \end{pmatrix} \begin{pmatrix} 4.88889 \\ 16 \\ 0.666667 \\ -2 \end{pmatrix}$$

Шаг 8:

$$\begin{pmatrix} 1 & 1.5 & 0 & 0 \\ 0 & 0 & 9 & 7 \\ 0 & 0 & 0 & 0.666667 \\ 0 & 0 & 0 & -2 \end{pmatrix} \begin{pmatrix} 4 \\ 16 \\ 0.666667 \\ -2 \end{pmatrix}$$

Шаг 9:

$$\begin{pmatrix} 1 & 1.5 & 0 & 0 \\ 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0.666667 \\ 0 & 0 & 0 & -2 \end{pmatrix} \begin{pmatrix} 4 \\ 9 \\ 0.666667 \\ -2 \end{pmatrix}$$

Шаг 10:

$$\begin{pmatrix} 1 & 1.5 & 0 & 0 \\ 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0.666667 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 4 \\ 9 \\ 0.666667 \\ 0 \end{pmatrix}$$

Далее возможны 2 ситуации:

1) В последней ненулевой строке в столбце свободных членов находится ненулевое число, а в матрице вся строка состоит из нулей. В этом случае решения нет.

2) В последней ненулевой строке в матрице есть ненулевые числа (при этом число, стоящее в столбце свободных членов, может быть или не быть нулевым). В этом случае решений бесконечно много.

Отметим, что нулевой строкой мы называем строку, в которой все числа стоящие в матрице и число, стоящее в столбце свободных членов равно нулю. В нашем случае имеет место быть 2-ая ситуация. Можно перезаписать матрицу в виде системы уравнений.

x1 + 1.5x2 = 4

9x3 = 9

0.666667 * x4 = 0.666667

Мы модем найти x4 и x3 сразу же (x3 = 1, x4 = 1). Для того, чтобы найти x1 и x2 можно в качестве x1 подставить любое число и выразить x2 из первого уравнения (например x1 = 1, x2 = 3). Таким образом одно из решений – (1, 3, 1, 1).

Приведём код Алгоритма Гаусса в случае бесконечно многих решений или отсутствия решений системы (определитель при этом = 0).

```
double gaussMod(vector<vector<double> >& M, int n)
{
        double zn = 1.;
      double eps = 1e-8;
        int J = 0;

    for(int i = 0; i < n; )
    {
                if (fabs(M[i][J]) < eps)
                        for(int j = i+1; j < n; j++)
                                if (fabs(M[j][J]) > eps)
                                {
                                zn *= -1.;
                                for(int k = 0; k <= n; k++)
                                {
                                        swap(M[i][k], M[j][k]);
                                }
                                }

        if (fabs(M[i][J]) < eps)
                        {
```

```
                J++;
                continue;
            }

        for(int j = 0; j < n; j++)
        {
            if (fabs(M[j][J]) > eps && j != i)
            {
                double koef = -(M[j][J] / M[i][J]);
                for(int k = 0; k <= n; k++)
                    M[j][k] += M[i][k] * koef;
            }
        }
                i++;
                J++;
    }

        double ans = 1;
    for(int i = 0; i < n; i++)
        {
                ans *= M[i][i];
        }

        return ans * zn; }
```

3) Алгоритм Гаусса для решения системы по некоторому модулю.

В реальных задачах редко приходится применять Алгоритм Гаусса в чистом виде. Чаще алгоритм необходимо модифицировать для того, чтобы решить систему линейных уравнений по некоторому модулю (чаще всего модуль является простым числом и это очень важно). В таком случам все арифметические действия производятся в кольце остатков на модуль и матрицу A нужно привести к диагональному виду тоже по модулю. Разберум какие изменения нужно внести в алгоритм Гаусса.

Во-первых, в процессе приведения матрицы к диагональному виду после любого арифметического действия (* , + , -) необходимо брать остаток от деления на mod.

Во-вторых, так как деление в кольце - это сложная операция (она эквивалентна домножению на обратный элемент в кольце), то нужно избавиться от всех делений в алгоритме. Вместо домножения строки на коэффициент $k = -a_{ji}/a_{ii}$ мы будем домножать обе строки на некоторое число и потом заменим одну из них их суммой. Например, пусть дана матрица A (мы решаем систему по

модулю 3, поэтому в матрице могут быть только числа 0, 1, 2).

$$\begin{pmatrix} 2 & 2 & 0 & 1 \\ 2 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 2 \\ 1 \end{pmatrix}$$

Для того, чтобы обнулить элемент второй строки и первого столбца домножим первую строку на 2, вторую строку на 1, сложим их (во всех действиях берём остатки по модулю 3) и заменим вторую строку на полученную сумму. Получившеяся матрица имеет вид:

$$\begin{pmatrix} 2 & 2 & 0 & 1 \\ 0 & 2 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 2 \\ 1 \end{pmatrix}$$

Таким образом мы будем обнулять все остальные элементы. Заметим, что при таком преобразовании решение СЛАУ не меняется, но! меняется значение определителя. Поэтому данный алгоритм нельзя использовать, если вам нужно посчитать определитель СЛАУ.

!!! Важно помнить, что данный алгоритм не будет работать, если модуль не является простым числом. В этом случае решить систему сложно, но обычно в задачах модуль является простым числом.

В задачах чаще всего систему линейных уравнений нужно решать по модулю 2. В этом случае матрица СЛАУ состоит из нулей и единиц.

Для решения СЛАУ с модулем 2 удобно использовать класс bitset, который можно использовать, если подключить модуль include<bitset>. Bitset представляет собой вектор, элементами которого являются bool. При этом, побитовые действия над такими векторами (И, ИЛИ, XOR) выполняются намного быстрее, по сравнению с использованием типа vector<bool>. Понятно, что вместо операции + можно использовать операцию XOR, которая представляет собой побитовое сложение по модулю 2.

4) Алгоритм Гаусса для вычисления обратной матрицы к матрице A.

Обратной матрице к матрице A называют матрицу, при умножении на которую получается единичная матрица, то есть A * (A$^{(}$ − 1)) = E.

Разберем алгоритм Жордана-Гуасса для нахождения обратной матрицы к матрице A.

1) Запишем единичную матрицу E справа от матрицы A. Пример:

$$\begin{pmatrix} 3 & 4 & 2 \\ 2 & -1 & -3 \\ 1 & 5 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2) Приведём матрицу A к диагональному виду. При этом будем выполнять преобразования над обеими матрицами. В результате получается (здесь пропущено много промежуточных действий):

$$\begin{pmatrix} 44 & 0 & 0 \\ 0 & 44 & 0 \\ 0 & 0 & 44 \end{pmatrix} \begin{pmatrix} 14 & 6 & -10 \\ -5 & 1 & 13 \\ 11 & -11 & -11 \end{pmatrix}$$

3) Поделим обе матрицы на 44. В результате слева получаем единичную матрицу, а справа - обратную матрицу к матрице A.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 14/44 & 6/44 & -10/44 \\ -5/44 & 1/44 & 13/44 \\ 11/44 & -11/44 & -11/44 \end{pmatrix}$$