算法基础（二十二）： 数学基础 - 高斯消元

基本思想

高斯消元是一种在`O(n^3)`的时间复杂度内求解包含`n`个方程和`n`个未知数的多元线性方程组的方法，以下方的这个方程组为例：

$$\left.\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m, \end{aligned}\right\}$$

我们将这个方程组的系数以及等号后面的结果抽取出来可以得到一个`n*(n + 1)`的增广矩阵，我们对增广矩阵进行以下三种**初等行列变换**，转化为最简阶梯型矩阵

- 某一行乘上一个非`0`的数，等价于将方程的两边乘上一个非零的数
- 交换某两行，等价于交换方程组的某两行
- 把某行的若干倍加到某一行上去，等价于将方程组的某一个方程的若干倍加到另外一个方程上去

这三种行列变换不会影响方程组的解的情况，跟原来的解是一样的，高斯消元就是将一个普通的方程，通过这三种等价的变换变成一个上三角的形式：

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ & a_{2i} & a_{2(i+1)} & a_{2n} & b_2 \\ & & & \vdots & \vdots \\ & & & a_{nn} & b_n \end{pmatrix}$$

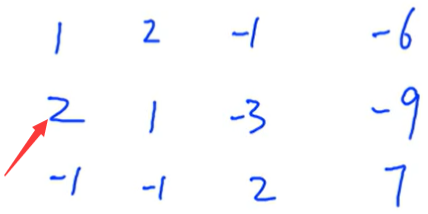
它的解有三种情况：

- 有唯一的解，完美的阶梯型
- 有无穷的解，出现了`0 = 0`的方程
- 无解，出现`0 = 非零`的情况

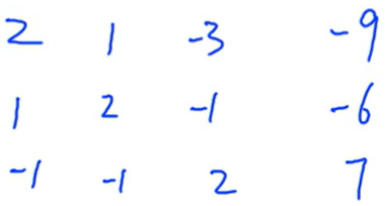
基本过程

枚举每一列，从第一列开始：

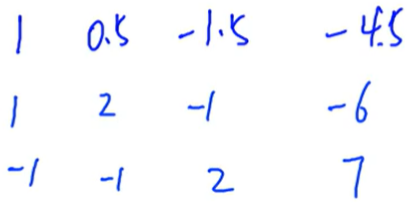
- 找到绝对值最大的那一行



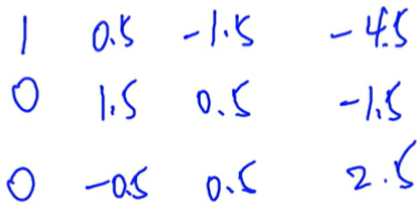
- 将这一行换到最上面



- 将改行的第一个数变成`1`

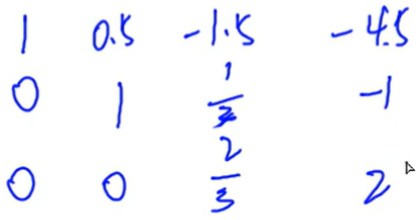


- 用这一行，将第`1`列除了第一行以外的别的行加上或是减去这一行的`k`倍后将第一个数变成`0`

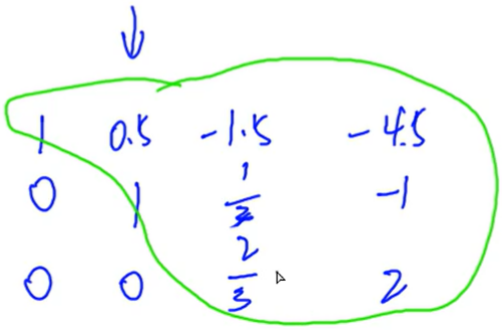


- 从而第一行的方程处理完毕

枚举第二列，但是第一行处理完毕，从第二行开始，得到如下结果

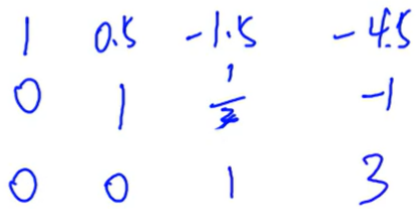


从而变成了一个上三角的形式：



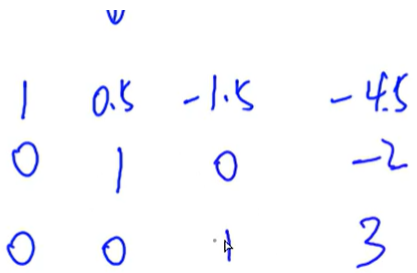
这个方程是一个完美的阶梯型，具有唯一的解

将最后一行的系数变成`1`:



得`x3 = 3`

将第二行的最后一个系数利用初等行变换化为`0`:



得`x2 = -2`

将第一行得后两个系数利用初等行变换变成`0`

得`x1 = 1`

总结一下:

枚举每一列`c`:

1. 找到绝对值最大的那一行
2. 将该行换到最上面
3. 将该行第`1`个数变成`1`
4. 将下面的所有行的第`c`列变成`0`

也就是以下的这个方程组：

代码实现

输入一个包含 n 个方程 n 个未知数的线性方程组。

方程组中的系数为实数。

求解这个方程组。

下图为一个包含 m 个方程 n 个未知数的线性方程组示例：

$$\left.\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m, \end{aligned}\right\}$$

输入格式

第一行包含整数 n。

接下来 n 行，每行包含 $n+1$ 个实数，表示一个方程的 n 个系数以及等号右侧的常数。

输出格式

如果给定线性方程组存在唯一解，则输出共 n 行，其中第 i 行输出第 i 个未知数的解，结果保留两位小数。

如果给定线性方程组存在无数解，则输出 `Infinite group solutions`。

如果给定线性方程组无解，则输出 `No solution`。

数据范围

$1 \leq n \leq 100$,
所有输入系数以及常数均保留两位小数，绝对值均不超过 100。

输入样例：

```
3
1.00 2.00 -1.00 -6.00
2.00 1.00 -3.00 -9.00
-1.00 -1.00 2.00 7.00
```

输出样例：

```
1.00
-2.00
3.00
```

```cpp
#include<iostream>
#include<algorithm>
#include<cmath>

using namespace std;

const int N = 110;
//c++可能存储0的时候存入的不是0而是一个很小的浮点数，为了防止出现精度问题，我们认为小于1e-6这个数就为0
const double eps = 1e-6;


int n;

double a[N][N];
int gauss()
{
    //c表示枚举的是哪一列，r表示枚举的是哪一行
    int c, r;
    //从第0列第0行开始枚举
    for(c = 0, r = 0; c < n; c ++ )
    {
        int t = r;
        //找到当前这一列绝对值最大的那一行
        for(int i = r; i < n; i ++ )
        {
            //如果当前遍历的第i行第c列的绝对值大于备选答案的绝对值
            if(fabs(a[i][c]) > fabs(a[t][c]))
            {
                //记录此时的备选答案
                t = i;
            }
        }
        //这时找到了第c列绝对值最大的那一行t
        //如果第c列的绝对值最大的这一行的值为0
        if(fabs(a[t][c]) < eps) continue;

```

```
        //将绝对值最大的这一行的所有系数换到最上面
        //从哪一行开始枚举就认为这一行是再最上面
        for(int i = c; i <= n; i ++) swap(a[t][i], a[r][i]);

        //将这一行的第一个数变成1
        //倒着来，最后更新第一个数
        for(int i = n ; i >= c; i --)
        {
            a[r][i] /= a[r][c];
        }

        //将这一行的下面所有行的第c列消成0
        //此时第一个数为1的哪一行所在的行数是第r行
        for(int i = r + 1; i < n; i ++)
        {
            //只有不是0的时候才可以去操作
            if(fabs(a[i][c]) > eps)
            {
                //消0的过程就是用第i行的每一列减去最上面方程的a[i][c]倍
                for(int j = n; j >= c; j --)
                {
                    a[i][j] -= a[i][c] * a[r][j];
                }
            }
        }

        //这一行处理完了之后行数加一
        r ++;
    }
    //如果r < n说明在上面的循环中在找第c列绝对值最大的时候这一列全为0，假设此时c = r = k
    //c继续增加但是r不变，c = k + 1， 在这一列中继续消0，并且将绝对值最大的那一行换到第r = k行
    //除了这一行以外其余的所有行第c列会在后面的的过程中变成0，从而第k列，第k+1列同时为0
    //c继续增加持续上面的操作，绝对值最大的行换到上面，其余的行第c列消成0在下面
    //r最后小于c，并且一定有行的未知数系数值全为0，出现0 = ...这种情况
    //退出上面循环的时候c = n, r < n, 并且r指向的是第一个未知数系数值全为0的行
    //这时可能无解，可能无穷解
    if(r < n)
    {
        //遍历增广矩阵的第n + 1列的值
        for(int i = r; i < n; i ++)
        {
            //如果出现了 0 = num（不为0）的情况就说明无解
            if(fabs(a[i][n]) > eps) return 2;
        }

        //否则有无穷的解
        return 1;
    }

    //求解值
    for(int i = n - 1; i >= 0; i --)
```

```
        for(int j = i + 1; j < n; j ++)
            a[i][n] -= a[i][j] * a[j][n];


    //有唯一的解
    return 0;


}
int main()
{
    cin >> n;
    for(int i = 0; i < n; i ++)
        for(int j = 0; j < n + 1; j ++)
            cin >> a[i][j];

    int t = gauss();

    if(t == 0)
    {
        //有唯一的解
        for(int i = 0; i < n; i ++){
            if(fabs(a[i][n]) < eps) a[i][n] = 0;
            printf("%.2lf\n", a[i][n]);
        }

    }
    else if(t == 1)
    {
        //有无穷多组解
        puts("Infinite group solutions");
    }
    else
    {
        //无解
        puts("No solution");
    }

    return 0;


}
```

关于这个代码，主要解释两个地方

第一个是判断无解和无穷解的代码：

```
if(r < n)
    {
        //遍历增广矩阵的第n + 1列的值
        for(int i = r; i < n; i ++)
        {
            //如果出现了 0 = num（不为0）的情况就说明无解
            if(fabs(a[i][n]) > eps) return 2;
        }

        //否则有无穷的解
        return 1;
    }
```

这里可以简单证明一下算法的正确性：

当前面的一轮循环完成后我们有三个主要的性质：

- 每次遍历到`c`列，一定会将这个列`r`行往下除了绝对值最大的那一行外的其他行的`c`列消成`0`
- 每次循环完成，`r`一定指向前置`0`最多的行（不论当前`c`列是否绝对值最大为`0`）
- 每次循环时，`r`指向的这一行往下的行第`c`列之前的列（不包括第`c`列）全为`0`

由性质一，若中间存在一列绝对值最大的行为`0`，那么当循环完毕后，一定有一行的系数值全为`0`，我们可以以一个`n * n`的矩阵作为理解：

假设在其中枚举到了`c = r = k`，第`k`行，第`k`列，假设此时`k`行往下绝对值最大为`0`，那么在下一次循环中列继续枚举，但是行不变即：`c = k + 1, r = k`，于是在后面的消零过程中由于列数有限最多再遍历`n - k`列，所以我们最多再挑选出`n - k`行绝对值最大的行来将其他的行消零，这`n - k`行可以保证系数值不全为`0`，但是在后面循环中`r`是从`k`开始的，也就是说总共有`n - k + 1`行，所以必然会存在一行的系数值全为`0`，而且若不能保证后面的列中都存在绝对值不为`0`的行的话，最后系数值全为`0`的行数将大于`1`

由性质二，`r`一定指向这个系数值全为`0`的行，并且是指向从上往下的第一行系数值全为`0`的行

接下来就判断增广矩阵的最后一列即可，如上面的代码所示

第二个是求解最后未知数的解的代码：

```
    //求解值
    for(int i = n - 1; i >= 0; i --)
        for(int j = i + 1; j < n; j ++)
            a[i][n] -= a[i][j] * a[j][n];
```

以题目中为例：

$$\begin{pmatrix} 1 & \frac{1}{2} & -\frac{3}{2} & -\frac{9}{2} \\ 0 & 1 & \frac{1}{3} & -1 \\ 0 & 0 & 1 & 3 \end{pmatrix}$$

`i`指的是行数，从最后一行开始向上遍历

`i = 2`的时候分析略

`i = 1`的时候

`j = 2`，`a[1][3] -= a[1][2] * a[2][3]，-1 -= 1/3 * 3`，从而变成：

$$\begin{pmatrix} 1 & \frac{1}{2} & -\frac{3}{2} & -\frac{9}{2} \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & 3 \end{pmatrix}$$

注意此时系数矩阵中的系数其实没有变化，仍然是`1 / 3`，不过此处为便于理解，置为`0`

`i = 0`的时候

`j = 1`， 第一行的最后一个系数减去第一行的第二个系数乘第二行的最后一个系数，也就是模拟将`1/2`变为`0`的过程：`-9/2 -= 1/2 * -2`

$$\begin{pmatrix} 1 & 0 & -\frac{3}{2} & -\frac{7}{2} \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & 3 \end{pmatrix}$$

`j = 2`，第一行的最后一个系数减去第一行的第三个系数乘第三行的最后一个系数，也就是模拟将`-3/2`变为`0`的过程：`-7/2 -= -3/2 * 3`

所以这里的`j`即当作第`i`行的后面非零系数的下标向后移动作为系数`a[i][j]`，又当作第`i`行后面行行下标用来提取最后一个系数`a[j][n]`，从下往上依次计算结果

另外如果觉得这样的处理比较麻烦的话可以在消`0`的时候直接将这个方程变成对角矩阵：

```
        //将所有行的第c列消成0
        //此时第一个数为1的哪一行所在的行数是第r行
        for(int i = 0; i < n; i ++)
        {
            //只有不是0的时候才可以去操作
            if(fabs(a[i][c]) > eps && i != r)
            {
                //消0的过程就是用第i行的每一列减去最上面方程的a[i][c]倍
                for(int j = n; j >= c; j --)
                {
                    a[i][j] -= a[i][c] * a[r][j];
                }
            }
        }
```

这样增广矩阵的最后一列就直接是结果了