线性代数—高斯消元

引入

消元法

消元法是将方程组中的一方程的未知数用含有另一未知数的代数式表示，并将其带入到另一方程中，这就消去了一未知数，得到一解；或将方程组中的一方程倍乘某个常数加到另外一方程中去，也可达到消去一未知数的目的。消元法主要用于二元一次方程组的求解。

矩阵表示线性方程组

例如，将线性方程组：

$$\begin{cases} 7x_1 + 8x_2 + 9x_3 = 13 \\ 4x_1 + 5x_2 + 6x_3 = 12 \\ x_1 + 2x_2 + 3x_3 = 11 \end{cases}$$

写成矩阵乘法的形式（将系数抽出来）：

$$\begin{pmatrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 13 \\ 12 \\ 11 \end{pmatrix}$$

简记为：$Ax = b$，其中系数矩阵 $A = \begin{pmatrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{pmatrix}$；未知量 $x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$；常数项 $b = \begin{pmatrix} 13 \\ 12 \\ 11 \end{pmatrix}$.

其本质是：矩阵 A（系数矩阵）左乘一个列向量 x（未知量）等与一个列向量 b（常数项）。

增广矩阵

对于一个线性方程组，未知数前的系数构成系数矩阵，如果在系数矩阵右端补上线性方程组的常数项则构成增广矩阵。即为方程组系数矩阵 A 与常数列 b 的并生成的新矩阵，表示为 $(A|b)$。

应用初等行变换，可以将线性方程组对应的增广矩阵先转化为行阶梯形矩阵，再转化为行最简形矩阵，进而完成线性方程组的求解。这个方法叫做消元法解线性方程组，而高斯消元法，是按照一定的顺序进行的消元算法。

增广矩阵行初等变换化为行最简形，即是利用了高斯消元法的思想理念，省略了变量而用变量的系数位置表示变量，增广矩阵中用竖线隔开了系数矩阵和常数列，代表了等于符号。

例如方程 $\begin{cases} 4x+y &= 100 \\ x-y &= 100 \end{cases}$ 可以表示为：$\left(\begin{array}{cc|c} 4 & 1 & 100 \\ 1 & -1 & 100 \end{array}\right)$

行最简形矩阵

在阶梯形矩阵中，若其各行的第一个非零元素均为 1，且所在列的其他元素都为 0，就称该矩阵为行最简形矩阵。

消元法理论的核心

考虑将消元法的思想引入 n 元一次方程组中，我们可以得到如下结论：

- 两方程互换，解不变；
- 一方程乘以非零数 k，解不变；
- 一方程加上另一方程的 k 倍，解不变。

初等变换

对于矩阵 A，可以进行［初等行变换］和［初等列变换］，统称为初等变换（初等行列变换）。初等行变换与初等列变换一样，都有 3 种：对换（switching）、倍乘（multiplication）、倍加（addition）。

初等行变换（列同行）	对应到方程组里的描述
对换：第 i，j 行互换	两方程互换，解不变
倍乘：第 i 行乘非零数 k	一方程乘以非零数 k，解不变
倍加：第 j 行乘 k 加到第 i 行	一方程加上另一方程的 k 倍，解不变

在初等变换中，对换可以通过倍乘和倍加实现，而倍加不能通过倍乘和对换实现。因此，相较对换而言，倍乘和倍加是更为本质的操作。对换操作是为了在消元法中，保证消元的有序，而引入的辅助操作。

高斯消元法（人工）

理论基础

德国数学家高斯对消元法进行了思考分析，得出了如下结论：

- 在消元法中，参与计算和发生改变的是方程中各变量的系数；
- 各变量并未参与计算，且没有发生改变；
- 可以利用系数的位置表示变量，从而省略变量；
- 在计算中将变量简化省略，方程的解不变。

高斯在这些结论的基础上，提出了高斯消元法，首先将方程的增广矩阵利用行初等变换化为行最简形，然后以线性无关为准则对自由未知量赋值，最后列出表达方程组通解。

标准形式

已知 n 元线性一次方程组。

$$
\begin{cases}
a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1 \\
a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2 \\
\cdots \\
a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n = b_n
\end{cases}
$$

求方程组的解的情况。

方法及步骤

高斯消元法在将增广矩阵化为最简形后对于自由未知量的赋值，我们可以将高斯消元法划分为五步骤，从而提出五步骤法，内容如下：

1．增广矩阵行初等行变换为行最简形；
2．还原线性方程组；
3．求解第一个变量；
4．补充自由未知量；
5．列表示方程组通解。

下面我们举例说明。

过程

例：利用高斯消元法五步骤法求解线性方程组：

$$\begin{cases} 2x_1+5x_3+6x_4 &= 9 \\ x_3+x_4 &= -4 \\ 2x_3+2x_4 &= -8 \end{cases}$$

1．化为行最简形矩阵：

具体方法：后面讲（建议先看完这些内容再往下进行）。

原始矩阵

$$\left(\begin{array}{cccc|c} 2 & 0 & 5 & 6 & 9 \\ 0 & 0 & 1 & 1 & -4 \\ 0 & 0 & 2 & 2 & -8 \end{array}\right)$$

将第二行的 2 倍加到第三行

$$\left(\begin{array}{cccc|c} 2 & 0 & 5 & 6 & 9 \\ 0 & 0 & 1 & 1 & -4 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

将第一行乘以 $\dfrac{1}{2}$

$$\left(\begin{array}{cccc|c} 1 & 0 & 2.5 & 3 & 4.5 \\ 0 & 0 & 1 & 1 & -4 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

将第二行的 2.5 倍加到第一行

$$\left(\begin{array}{cccc|c} 1 & 0 & 0 & 0.5 & 14.5 \\ 0 & 0 & 1 & 1 & -4 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

2．还原线性方程组：

在行最简形矩阵的基础上，将之重新书写为线性方程组的形式，即将行最简形中各位置的系数重新赋予变量，中间的竖线还原为等号。

$$\begin{cases} x_1 + 0.5x_4 &= 14.5 \\ x_3 + x_4 &= -4 \end{cases}$$

3．求解第一个变量：

对于所还原的线性方程组而言，将方程组中每个方程的第一个变量，用其他量表达出来；即对于每个方程，仅将第一个变量放在方程左边，其他所有量都放在方程右边。

$$\begin{cases} x_1 = -0.5x_4 + 14.5 \\ x_3 = -x_4 - 4 \end{cases}$$

※因为是行最简形矩阵，所以每个方程的第一个变量的系数一定是 1。

4．补充自由未知量：

第 3 步中，求解出变量 x_1 和 x_3，从而说明了方程剩余的变量 x_2 和 x_4 不受方程组的约束，是自由未知量，可以取任意值，所以需要在第 3 步骤解得基础上进行解得补充，即 $x_2 = x_2$，$x_4 = x_4$。

$$\begin{cases} x_1 = -0.5x_4 + 14.5 \\ x_2 = x_2 \\ x_3 = -x_4 - 4 \\ x_4 = x_4 \end{cases}$$

5．表示方程组的通解：

在第 4 步的基础上，将解表达为列向量组合的表示形式，同时由于 x_2 和 x_4 是自由未知量，可以取任意值，所以在解得右边，令二者分别为任意常数 C_1 和 C_2，即实现了对方程组的求解。

$$\begin{pmatrix}x_1\\x_2\\x_3\\x_4\end{pmatrix}=\begin{pmatrix}0\\1\\0\\0\end{pmatrix}x_2+\begin{pmatrix}-0.5\\0\\-1\\1\end{pmatrix}x_4+\begin{pmatrix}14.5\\0\\-4\\0\end{pmatrix}$$
$$=\begin{pmatrix}0\\1\\0\\0\end{pmatrix}C_1+\begin{pmatrix}-0.5\\0\\-1\\1\end{pmatrix}C_2+\begin{pmatrix}14.5\\0\\-4\\0\end{pmatrix}$$

※用 C_1、C_2、C_3、$\dots$ 表示任意常数。

高斯消元法（计算机）

可以看出来，最难的一步其实是第 1 步：化为行最简形矩阵。如果是手工计算的话，依次考虑每一列，随便找一个这一列非零的且前面均为 0 的行，将这一行全部除以这个数就可以了。

但是程序就需要考虑效率以及除法的精度了，具体思想是将前面已经有非零的数的行固定在最前面，即 $1\sim r$ 的位置固定、后面不再考虑。

方法

初始时 $r=0$，依次枚举每一列：

1. 找到这一列绝对值最大的行（下面做除法的时候，分子大，可以减小精度误差）；$\in[r+1,n]$
2. 把这一行换到第 $r+1$ 行
3. 将这一行的每个数同时除以这一个非零的数；如果没有非零的数则表示没有唯一解（取决于行最简形矩阵的右侧是否非零，所以如果要确定是无穷解还是无解，就需要继续算，在求出行最简形矩阵后再判断）
4. 把这一列，除了第一个数，全都消成零；即按比例减去
5. 固定这个方程，以后不再考虑；即 $r=r+1$

解的情况判断

1. 完美阶梯型：唯一解；
   即第 i 行，第一个非零在第 i 个。
2. 左边没有未知数，右边系数非零：无解；
   即 $r+1\sim n$ 行中，有右侧非零的方程。

3．否则：无穷组解。

高斯-约旦消元法

高斯-约旦消元法，是高斯消元法的另一个版本，其方法与高斯消去法相同；唯一相异之处就是这算法产生出来的矩阵是一个简化行梯阵式，而不是高斯消元法中的行梯阵式。相比起高斯消元法，此算法的效率比较低，却可把方程组的解用矩阵一次过表示出来。

高斯-约旦消元法就是将下面的方程的倍数也减到上面已经考虑过的方程中（有回代，而普通高斯消元法无回代），使上面的方程的该列也变为 0；具体来说就是用当前选中的方程去消掉其他方程的时候，是否消去其上面已经算过的方程。

用高斯消元法和高斯-约旦消元法算出来的行最简形矩阵有如下区别：

高斯消元法

$$\left(\begin{array}{cccc|c} 1 & * & * & * & * \\ 0 & 1 & * & * & * \\ 0 & 0 & 1 & * & * \\ 0 & 0 & 0 & 1 & * \end{array}\right)$$

高斯-约旦消元法

$$\left(\begin{array}{cccc|c} 1 & 0 & 0 & 0 & * \\ 0 & 1 & 0 & 0 & * \\ 0 & 0 & 1 & 0 & * \\ 0 & 0 & 0 & 1 & * \end{array}\right)$$

具体的内容可以见：https://www.cnblogs.com/mk-oi/p/14290455.html；
这篇文章中说高斯-约旦消元法不可判无解和无穷解，但我实测也是可以的。

代码实现

为了代码简便，下文的代码使用的都是高斯-约旦消元法，而不是普通的高斯消元法。

[P3389 高斯消元法]

- $n \times n$ 的矩阵，仅需输出唯一解或无唯一解。

```
const int N = 110;

double a[N][N], b[N];

int main()
{
    int n = rr;
```

```
    for (int i = 1; i <= n; ++i)
    {
        for (int j = 1; j <= n; ++j)
            a[i][j] = rr;
        b[i] = rr;
    }

    for (int i = 1; i <= n; ++i)
    {
        int r = 0;
        for (int j = i; j <= n; ++j)
            if (fabs(a[j][i]) > fabs(a[r][i]))
                r = j;

        if (a[r][i] == 0)
            printf("No Solution\n"), exit(0);

        const double t = a[r][i];
        for (int j = 1; j <= n; ++j)
            swap(a[i][j], a[r][j]), a[i][j] /= t;
        swap(b[i], b[r]), b[i] /= t;

        for (int j = 1; j <= n; ++j)
        {
            if (i == j || a[j][i] == 0)
                continue;
            const double c = a[j][i];
            for (int k = i; k <= n; ++k)
                a[j][k] -= c * a[i][k];
            b[j] -= c * b[i];
        }
    }

    for (int i = 1; i <= n; ++i)
        printf("%.2lf\n", b[i]);
    return 0;
}
```

[P2455 线性方程组]

- $n \times n$ 的矩阵，需要判断解的三种情况。

```cpp
const int N = 55;

double a[N][N], b[N];

int main()
{
    int n = rr;

    for (int i = 1; i <= n; ++i)
    {
        for (int j = 1; j <= n; ++j)
            a[i][j] = rr;
        b[i] = rr;
    }

    int row = 1;
    for (int col = 1; col <= n; ++col)
    {
        int mi = 0;
        double cm = -1;

        for (int i = row; i <= n; ++i)
            if (fabs(a[i][col]) > cm)
                cm = fabs(a[i][col]), mi = i;

        if (a[mi][col] == 0)
            continue;

        const double t = a[mi][col];
        for (int i = 1; i <= n; ++i)
            swap(a[row][i], a[mi][i]), a[row][i] /= t;
        swap(b[row], b[mi]), b[row] /= t;

        for (int i = 1; i <= n; ++i)
        {
            if (i == row || a[i][col] == 0)
                continue;
            const double c = a[i][col];
            for (int j = 1; j <= n; ++j)
                a[i][j] -= c * a[row][j];
            b[i] -= c * b[row];
        }

```

```
        ++row;
    }

    if (row == n + 1)
    {
        for (int i = 1; i <= n; ++i)
            printf("x%d=%.2lf\n", i, b[i]);
        exit(0);
    }

    for (int i = row; i <= n; ++i)
        if (b[i] != 0)
            printf("-1\n"), exit(0);

    printf("0\n");
    return 0;
}
```

Reference

[1] https://oi-wiki.org/math/linear-algebra/matrix/

[2] https://oi-wiki.org/math/linear-algebra/elementary-operations/

[3] https://oi-wiki.org/math/numerical/gauss/

[4] https://www.cnblogs.com/mk-oi/p/14290455.html

合集：学习笔记

标签：算法 , 学习笔记