高斯消元

引入

高斯消元法（Gauss–Jordan elimination）是求解线性方程组的经典算法，它在当代数学中有着重要的地位和价值，是线性代数课程教学的重要组成部分。

高斯消元法除了用于线性方程组求解外，还可以用于行列式计算、求矩阵的逆，以及其他计算机和工程方面。

消元法及高斯消元法思想

定义

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。

解释

例一：利用消元法求解二元一次线性方程组：

$$\begin{cases} 4x+y &= 100 \\ x-y &= 100 \end{cases}$$

解：将方程组中两方程相加，消元 y 可得：

$5x = 200$

解得：

$x = 40$

将 $x = 40$ 代入方程组中第二个方程可得：

$y = -60$

消元法理论的核心

消元法理论的核心主要如下：

- 两方程互换，解不变；

- 一方程乘以非零数 k，解不变；
- 一方程乘以数 k 加上另一方程，解不变。

高斯消元法思想概念

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。

高斯消元五步骤法

解释

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，需要掌握线性相关知识，且赋值存在人工经验的因素，使得在学习过程中有一定的困难，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。

利用实例进一步说明该算法的运作情况。

过程

例二：利用高斯消元法五步骤法求解线性方程组：

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

增广矩阵行（初等）变换为行最简形

所谓增广矩阵，即为方程组系数矩阵 A 与常数列 b 的并生成的新矩阵，即 $(A|b)$，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。

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

$$\xrightarrow{r_3-2r_2} \left(\begin{array}{cccc|c} 2 & 0 & 5 & 6 & 9 \\ 0 & 0 & 1 & 1 & -4 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

化为行阶梯形

$$\xrightarrow{\frac{r_1}{2}} \left(\begin{array}{cccc|c} 1 & 0 & 2.5 & 3 & 4.5 \\ 0 & 0 & 1 & 1 & -4 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

$$\xrightarrow{r_1-r_2\times 2.5} \left(\begin{array}{cccc|c} 1 & 0 & 0 & 0.5 & 14.5 \\ 0 & 0 & 1 & 1 & -4 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

化为最简形

还原线性方程组

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

解释

> 所谓的还原线性方程组，即是在行最简形的基础上，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。

求解第一个变量

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

解释

> 即是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。如方程组两方程中的第一个变量 x_1 和 x_3

补充自由未知量

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

解释

> 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，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，补充的方法为 $x_2 = x_2, x_4 = x_4$，这种解得补充方式符合自由未知量定义，并易于理解，因为是自由未知量而不受约束，所以只能自己等于自己。

列表示方程组的通解

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

其中 C_1 和 C_2 为任意常数。

解释

> 即在第 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。

行列式计算

解释

$N \times N$ 方阵行列式（Determinant）可以理解为所有列向量所夹的几何体的有向体积

例如：

$$\begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} = 1$$

$$\begin{vmatrix} 1 & 2 \\ 2 & 1 \end{vmatrix} = -3$$

行列式有公式

$$\det(A) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)}$$

其中 S_n 是指长度为 n 的全排列的集合，σ 就是一个全排列，如果 σ 的逆序对对数为偶数，则 $\operatorname{sgn}(\sigma) = 1$，否则 $\operatorname{sgn}(\sigma) = -1$。

通过体积概念理解行列式不变性是一个非常简单的办法：

- 矩阵转置，行列式不变；
- 矩阵行（列）交换，行列式取反；
- 矩阵行（列）相加或相减，行列式不变；
- 矩阵行（列）所有元素同时乘以数 k，行列式等比例变大。

> 由此，对矩阵应用高斯消元之后，我们可以得到一个对角线矩阵，此矩阵的行列式由对角线元素之积所决定。其符号可由交换行的数量来确定（如果为奇数，则行列式的符号应颠倒）。因此，我们可以在 $O(n^3)$ 的复杂度下使用高斯算法计算矩阵。
>
> 注意，如果在某个时候，我们在当前列中找不到非零单元，则算法应停止并返回 0。

实现

```cpp
constexpr double EPS = 1E-9;
int n;
vector<vector<double>> a(n, vector<double>(n));

double det = 1;
for (int i = 0; i < n; ++i) {
  int k = i;
  for (int j = i + 1; j < n; ++j)
    if (abs(a[j][i]) > abs(a[k][i])) k = j;
  if (abs(a[k][i]) < EPS) {
    det = 0;
    break;
  }
  swap(a[i], a[k]);
  if (i != k) det = -det;
  det *= a[i][i];
  for (int j = i + 1; j < n; ++j) a[i][j] /= a[i][i];
  for (int j = 0; j < n; ++j)
    if (j != i && abs(a[j][i]) > EPS)
      for (int k = i + 1; k < n; ++k) a[j][k] -= a[i][k] * a[j][i];
}

cout << det;
```

矩阵求逆

对于方阵 A，若存在方阵 A^{-1}，使得 $A \times A^{-1} = A^{-1} \times A = I$，则称矩阵 A 可逆，A^{-1} 被称为它的逆矩阵。

给出 n 阶方阵 A，求解其逆矩阵的方法如下：

1. 构造 $n \times 2n$ 的矩阵 (A, I_n)；
2. 用高斯消元法将其化简为最简形 (I_n, A^{-1})，即可得到 A 的逆矩阵 A^{-1}。如果最终最简形的左半部分不是单位矩阵 I_n，则矩阵 A 不可逆。

该方法的正确性证明需要用到较多线性代数的知识，限于篇幅这里不再给出。感兴趣的读者可以自行查阅相关资料。

高斯消元法解异或方程组

异或方程组是指形如

$$\begin{cases} a_{1,1}x_1 \oplus a_{1,2}x_2 \oplus \cdots \oplus a_{1,n}x_n & = b_1 \\ a_{2,1}x_1 \oplus a_{2,2}x_2 \oplus \cdots \oplus a_{2,n}x_n & = b_2 \\ \cdots & \cdots \\ a_{m,1}x_1 \oplus a_{m,2}x_2 \oplus \cdots \oplus a_{m,n}x_n & = b_m \end{cases}$$

的方程组，其中 $\oplus$ 表示「按位异或」（即 `xor` 或 C++ 中的 `^`），且式中所有系数/常数（即 $a_{i,j}$ 与 b_i）均为 0 或 1。

由于「异或」符合交换律与结合律，故可以按照高斯消元法逐步消元求解。值得注意的是，我们在消元的时候应使用「异或消元」而非「加减消元」，且不需要进行乘除改变系数（因为系数均为 0 和 1）。

注意到异或方程组的增广矩阵是 01 矩阵（矩阵中仅含有 0 与 1），所以我们可以使用 C++ 中的 `std::bitset` 进行优化，将时间复杂度降为 $O(\frac{n^2m}{\omega})$，其中 n 为元的个数，m 为方程条数，ω 一般为 32（与机器有关）。

参考实现：

```cpp
std::bitset<1010> matrix[2010];  // matrix[1~n]: 增广矩阵，0 位置为常数

std::vector<bool> GaussElimination(
    int n, int m)  // n 为未知数个数，m 为方程个数，返回方程组的解
                   // （多解 / 无解返回一个空的 vector）
{
  for (int i = 1; i <= n; i++) {
    int cur = i;
    while (cur <= m && !matrix[cur].test(i)) cur++;
    if (cur > m) return std::vector<bool>(0);
    if (cur != i) swap(matrix[cur], matrix[i]);
    for (int j = 1; j <= m; j++)
      if (i != j && matrix[j].test(i)) matrix[j] ^= matrix[i];
  }
  std::vector<bool> ans(n + 1);
```

```
16    for (int i = 1; i <= n; i++) ans[i] = matrix[i].test(0);
17    return ans;
18  }
```

练习题

- Codeforces - 巫师和赌注
- luogu - SDOI2010 外星千足虫
