

线性方程组的解法

本章学习 $\mathbf{Ax} = \mathbf{b}$ 形式的线性方程组的求解方法。有两大类方法：**直接法**和**迭代法**。

直接法

直接法可以直接得到方程组的解，这类方法对于小型矩阵可以快速得到精确的解。这里介绍三类方法：**Cramer方法**、**高斯消去法**、**直接三角分解法**。

Cramer方法

Cramer方法是最直接的线性方程组求解方法，但是效率非常低，不常用。

Cramer方法对上述方程的求解又如下公式：

$$x_i = \frac{\Delta_i}{\det \mathbf{A}}$$

其中 Δ_i 是将 \mathbf{A} 矩阵的第 i 列替换为向量 \mathbf{b} 后得到的矩阵的行列式。

高斯Gauss消去法

高斯消元法是最为直接的求解方法，本质上为小学学习方程组时的基本换元方法的系统化体现。学习了**顺序消元法**和**列主元素消元法**两类，后者是在前者的基础上额外添加了一步的优化。两种方法的具体求解方法已经写入了 `Gauss_Elimination.py` 的文件中。B站[现代不抽象](#)的这个视频解释的很好。

本质上两种方法都首先构建了一个增广矩阵：

$$[\mathbf{A}, \mathbf{b}] = \begin{bmatrix} a_{11} & \cdots & a_{1n} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ a_{n1} & \cdots & a_{nn} & b_n \end{bmatrix}$$

然后按照每行的顺序进行初等行变化，使得矩阵变为如下的形式：

$$\begin{bmatrix} 1 & \cdots & a'_{1n} & b'_1 \\ & \ddots & \vdots & \vdots \\ \mathbf{0} & & 1 & b'_n \end{bmatrix}$$

然后将其反过来代回，变化成下面的矩阵，得到的即为向量 \mathbf{x} 的解：

$$[\mathbf{I}_{n \times n} \quad \mathbf{b}'_{n \times 1}]$$

列主元素消元法，是在初等行变化的基础上，加入一个判定和交换，使得当前行需要消解为1的列的元素是剩余所有行中最大的，执行代码后即可明白。

直接三角分解法

直接三角分解法包括**Doolittle**和**Crout**分解法两类，两个方法都将原先的线性方程组分解为以下形式：

$$\mathbf{Ax} = \mathbf{LUx} = \mathbf{Ly} = \mathbf{b}$$

然后求解的时候，可以首先求解 \mathbf{y} ，然后求解 \mathbf{x} 。

在**Doolittle**分解时， \mathbf{L} 为单位下三角阵， \mathbf{U} 为上三角阵

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ l_{21} & 1 & 0 & \dots & 0 \\ l_{31} & l_{32} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \dots & 1 \end{bmatrix} \quad \mathbf{U} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & u_{22} & u_{23} & \dots & u_{2n} \\ 0 & 0 & u_{33} & \dots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & u_{nn} \end{bmatrix}$$

而在**Crout**分解时，则反过来， \mathbf{L} 为下三角阵， \mathbf{U} 为单位上三角阵

$$\mathbf{L} = \begin{bmatrix} l_{11} & 0 & 0 & \dots & 0 \\ l_{21} & l_{22} & 0 & \dots & 0 \\ l_{31} & l_{32} & l_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \dots & l_{nn} \end{bmatrix} \quad \mathbf{U} = \begin{bmatrix} 1 & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & 1 & u_{23} & \dots & u_{2n} \\ 0 & 0 & 1 & \dots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

Doolittle 和 **Crout** 分解的充分必要条件都为 $\mathbf{A}_{n \times n}$ 的前 $n - 1$ 个顺序主子式行列式不为零。

对于Doolittle和Crout分解法，其元素的计算公式可以根据矩阵的形式反推得到，若有草稿纸的情况下可以手动对较小规模的矩阵手动分解。对于 \mathbf{L} 和 \mathbf{U} 的每个元素的计算，都是交替进行的。对于Doolittle方法，首先计算 \mathbf{U} 的第 n 行，然后计算 \mathbf{L} 的第 n 列，之后计算 \mathbf{U} 的第 $n + 1$ 行，以此类推；Crout方法则反过来。具体的代码求解已经写在LU_Decomposition.py的文件中，可以尝试运行试一试。

迭代法

前面介绍了求解线性方程组的直接方法，对于大型稀疏矩阵的运算，上述方法可能效率较低，因此引入迭代方法，通过多次计算，逐渐逼近方程组的解。同时现代计算GPU等支持并行计算，适配迭代方法的矩阵运算。

这里介绍了四种迭代方法：**简单迭代法**、**Jacobi迭代法**、**Gauss-Seidel迭代法**、以及**逐次超松弛SOR方法**。这几个方法的具体代码已经写在了Iteration_Method.py文件中。

简单迭代法

对于简单迭代法，可以将原方程组写成这样的形式：

$$\mathbf{Ax} = (\mathbf{N} - \mathbf{P})\mathbf{x} = \mathbf{b}$$

$$\mathbf{Nx} = \mathbf{Px} + \mathbf{b}$$

$$\mathbf{x} = \mathbf{N}^{-1}\mathbf{Px} + \mathbf{N}^{-1}\mathbf{b} = \mathbf{Gx} + \mathbf{d}$$

这样可以写成迭代式，这即为简单迭代法。

$$\mathbf{x}^{(k+1)} = \mathbf{Gx}^{(k)} + \mathbf{d}$$

对于上述的简单迭代法的收敛条件，有两个判断定理：

1. 定义矩阵的谱半径 $\rho(\mathbf{G}) = \max |\lambda_i|$ ，即矩阵所有特征值的绝对值的最大，若 $\rho(\mathbf{G}) < 1$ ，那么简单迭代法收敛。这是简单迭代法收敛的充分必要条件。
2. 若矩阵的某种范数 $\|\mathbf{G}\| < 1$ ，那么简单迭代法也必定收敛且唯一，且有两个误差估计式：

$$\|\mathbf{x}^{(k)} - \mathbf{x}^*\| \leq \frac{\|\mathbf{G}\|^k}{1 - \|\mathbf{G}\|} \|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\|$$

$$\|\mathbf{x}^{(k)} - \mathbf{x}^*\| \leq \frac{\|\mathbf{G}\|}{1 - \|\mathbf{G}\|} \|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|$$

个人理解是第二个收敛条件从数学上证实了只要两次求解的解几乎没怎么发生变化，即 $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|$ ，那么就可以认为求解出来的解已经非常接近于真实的精确解 \mathbf{x}^* ，但是若 $\|\mathbf{G}\|$ 很接近1，即使两次解非常接近也无法进行判定。

Jacobi迭代法

在这个迭代法中，原矩阵拆分为了三部分，分别为对角阵和上下三角阵：

$$\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U}$$

若 4×4 的情况，有这样的表示：

$$\mathbf{D} = \begin{bmatrix} a_{11} & & & \\ & a_{22} & & \\ & & a_{33} & \\ & & & a_{44} \end{bmatrix}$$

$$\mathbf{L} = \begin{bmatrix} 0 & & & \\ a_{21} & 0 & & \\ a_{31} & a_{32} & 0 & \\ a_{41} & a_{42} & a_{43} & 0 \end{bmatrix} \quad \mathbf{U} = \begin{bmatrix} 0 & a_{12} & a_{13} & a_{14} \\ & 0 & a_{23} & a_{24} \\ & & 0 & a_{34} \\ & & & 0 \end{bmatrix}$$

这样原来的迭代方法有：

$$\mathbf{x}^{(k+1)} = \mathbf{G}_J \mathbf{x}^{(k)} + \mathbf{b}_J = [-\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})]\mathbf{x}^{(k)} + [-\mathbf{D}^{-1}\mathbf{b}]$$

对于Jacobi迭代法，形态也满足简单迭代法，但是这里有额外的一个收敛定理：

若矩阵 \mathbf{A} 是按行/按列严格占优阵（即其主对角线上的各个元素的绝对值都大于这一行/列上其他元素绝对值之和，即 $|a_{ii}| > \sum_{i \neq j} |a_{ij}|$ ），那么Jacobi迭代法必然收敛。

Gauss-Seidel迭代方法(GS方法)

Gauss-Seidel方法(GS方法)在原先Jacobi方法上做了一些调整，保持矩阵 \mathbf{D} , \mathbf{L} , \mathbf{U} 不变，调整迭代方法为：

$$\mathbf{x}^{(k+1)} = \mathbf{G}_G \mathbf{x}^{(k)} + \mathbf{b}_G = [-(\mathbf{D} + \mathbf{L})^{-1}\mathbf{U}]\mathbf{x}^{(k)} + [(\mathbf{D} + \mathbf{L})^{-1}\mathbf{b}]$$

此时增加一个判定收敛定理，若系数矩阵 \mathbf{A} 为正定矩阵，那么GS方法必然收敛。

逐次超松弛迭代法（SOR方法）

SOR方法在GS方法上引入了一个参数变量 ω ，此时迭代方法调整为：

$$\mathbf{x}^{(k+1)} = \mathbf{G}_S \mathbf{x}^{(k)} + \mathbf{b}_S = -\left(\frac{1}{\omega}\mathbf{D} + \mathbf{L}\right)^{-1} \left[(1 - \frac{1}{\omega})\mathbf{D} - \mathbf{U}\right]\mathbf{x}^{(k)} + \left[\left(\frac{1}{\omega}\mathbf{D} + \mathbf{L}\right)^{-1} \mathbf{b}\right]$$

此时发现，若 $\omega = 1$ 时，SOR方法即为GS方法。

此时的判定收敛的定理中有，若系数矩阵 \mathbf{A} 为正定矩阵， $0 < \omega < 2$ ，那么GS方法必然收敛。