数值计算方法

-对称正定矩阵的分解及迭代法求解线性方程组

January 8, 2018

本节概要

对称正定矩阵的分解

我们首先接着上一讲的内容探讨矩阵分解，这里我们考虑的是对称正定矩阵。

对称正定矩阵在很多方面都有重要的应用，例如利用有限元方法解杆和板的弯曲问题，最后导出的刚度矩阵（可以简单的理解为系数矩阵 A）就是一个对称正定的矩阵，再比如一个多元函数的二阶导数（即这个函数的 Jacobi 矩阵）也是一个对称矩阵（是否正定要根据这个函数在所考虑的区域附近的性质决定）。

对称正定矩阵因为是对称的，所以实际上它的元素大约只有 $\frac{n^2}{2}$，我们考虑是否能够用一半的计算复杂度得到这个矩阵的 LU 分解。

对称正定矩阵

Definition 1

如果一个 $n \times n$ 的矩阵 A 满足 $A^T = A$，则这个矩阵被称为对称的。如果矩阵 A 满足当 $x \neq 0$ 时，$x^T Ax > 0$，则这个矩阵 A 称为正定的。

例如矩阵

$$A = \begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix}$$

就是一个对称正定矩阵，这是因为对任意的 $x = [x_1, x_2]^T$，我们有

$$\begin{aligned} x^T Ax &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ &= 2x_1^2 + 4x_1x_2 + 5x_2^2 \\ &= 2(x_1 + x_2)^2 + 3x_2^2 \end{aligned}$$

$x \neq 0$ 时结果一定大于 0。

对称正定矩阵

而矩阵

$$A = \begin{bmatrix} 2 & 4 \\ 4 & 5 \end{bmatrix}$$

不是一个对称正定矩阵，这是因为对任意的 $x = [x_1, x_2]^T$，我们有

$$\begin{aligned} x^T Ax &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 2 & 4 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ &= 2x_1^2 + 8x_1x_2 + 5x_2^2 \\ &= 2(x_1^2 + 4x_1x_2) + 5x_2^2 \\ &= 2(x_1 + 2x_2)^2 - 8x_2^2 + 5x_2^2 \\ &= 2(x_1 + 2x_2)^2 - 3x_2^2 \end{aligned}$$

如果令 $x_1 = -2, x_2 = 1$，则 $x^T Ax = -1$。

对称正定矩阵的性质

对称正定矩阵一定是非奇异的，因为如果 A 奇异，则存在 $x \neq 0$ 使得 $Ax = 0$，即 $x^T Ax = 0$，与 A 正定矛盾。除此之外，对称正定矩阵还有以下性质。

Proposition 2

如果 A 是对称矩阵，则 A 是正定的当且仅当 A 的所有特征值是正的。

Corollary 3

一个对称正定矩阵 A 的行列式是大于 0 的。

Proposition 4

如果 A 是一个 $n \times n$ 的对称正定矩阵，而 X 是 $n \times m$ 的满秩的矩阵且 $n \geq m$，则 $X^T AX$ 是一个 $m \times m$ 的对称正定矩阵。

对称正定矩阵的性质

Definition 5

如果一个方阵 A 的方子矩阵的对角线上的元素都是 A 的对角线上的元素，则这个方子矩阵称为 A 的主子矩阵。

Proposition 6

对称正定矩阵的任意一个主子矩阵都是对称正定矩阵。

对称正定矩阵的 Cholesky 分解

Theorem 7

如果 A 是一个对称正定矩阵，则一定存在一个 $n \times n$ 的上三角矩阵 R 使得 $A = R^T R$

Proof.

□

Remark 1. *Cholesky 分解不是唯一的，因为每一步中对子矩阵左上角的元素 a 开根号可以取正也可以取负。但是，如果限定 R 矩阵的对角线上的元素必须为正数的话，那么 Cholesky 分解就是唯一的。*

对称正定矩阵的 Cholesky 分解的算法

对于这个唯一的 Cholesky 分解，我们有以下算法

Algorithm 2 (Cholesky 分解算法).
for $k = 0, 1, 2, \ldots, n$
  if $A_{11}^k < 0$ （这里 $A_{11}^k < 0$ 表示第 k 步时需要进行分解的子矩阵 A^k 的左上角的元素，$A^k \in \mathbb{R}^{n-k+1 \times n-k+1}$ ）
    停机，该矩阵不是正定的
  $R_{kk} = \sqrt{A_1^k}$
  $u^T = \frac{A_{2:n-k+1}^k}{R_{kk}}$ （$A_{2:n-k+1}^k$ 表示第 k 步时需要进行分解的矩阵 A_k 的第一行除第一个元素之外剩下的元素）
  $R_{k,k+1:n} = u^T$ （$R_{k,k+1:n}$ 表示 R 的第 k 行从第 $k+1$ 列到第 n 列的元素）
  $A^{k+1} = A_{2:n-k+1,2:n-k+1}^k - uu^T$ （赋值符号右边的 $A_{2:n-k+1,2:n-k+1}^k$ 矩阵 A^k 除第一行第一列之外剩下的部分构成的子矩阵）
end

Cholesky 分解举例

找到以下矩阵的 Cholesky 分解

$$\begin{bmatrix} 4 & -2 & 2 \\ -2 & 2 & -4 \\ 2 & -4 & 11 \end{bmatrix}.$$

首先有 $R_{11} = \sqrt{A_{11}} = 2$，进而 $R_{1,2:3} = \frac{A^1_{2:3}}{R_{11}} = \frac{[-2,2]}{2} = [-1,1]$，得到

$$R = \left[\begin{array}{c|cc} 2 & -1 & 1 \\ \hline & & \\ & & \end{array}\right].$$

Cholesky 分解举例

把将 A^1 去掉第一行和第一列之后剩下的子矩阵 $A^1_{2:3,2:3}$ 减去 uu^T 得到

$$\left[\begin{array}{c|cc} & 2 & -4 \\ \hline & -4 & 11 \end{array}\right] - \left[\begin{array}{c|cc} & 1 & -1 \\ \hline & -1 & 1 \end{array}\right] = \left[\begin{array}{c|cc} & 1 & -3 \\ \hline & -3 & 10 \end{array}\right]$$

接下来我们利用 2×2 正定对称矩阵的 Cholesky 分解方法对

$$\begin{matrix} 1 & -3 \\ -3 & 10 \end{matrix}$$

进行分解，最终得到

$$R = \begin{bmatrix} 2 & -1 & 1 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{bmatrix}.$$

利用 Cholesky 分解的结果解方程组

用 Cholesky 分解解方程组的方法与 LU 分解一样，我们有

$$Ax = R^T Rx = b \tag{1}$$

等价于求解

$$\begin{aligned} R^T y &= b \\ Rx &= y. \end{aligned} \tag{2}$$

迭代法

我们在解非线性方程时用到了迭代法，基本思想就是把原方程作一个等价变换变成一个求函数不动点，之后不断对迭代函数进行迭代计算，如果得到的是收敛数列则这个收敛数列的极限就是不动点，即原方程的解。

解线性方程组我们也可以利用类似的思想，将方程组化为一个（线性的）多元函数的不动点的问题进行迭代，如果得到的向量序列是收敛的，那么我们就得到了这个（线性的）多元函数的不动点，也就是线性方程组的解。

之前我们所讲的直接法与迭代法最大的区别就在于，直接法在给定的有限步计算之后就一定能得到结果（例如高斯消元是 $O(n^3)$ 的运算），且这个结果是理论上的精确解；而迭代法并不能保证在一定迭代步数之后就得到结果，且得到的结果仍是精确解的一个近似。

虽然迭代法看似不如直接法效果好，但在实际计算中，迭代法对于解线性方程组，尤其是系数矩阵具有特殊结构的线性方程组却是十分有效的方法。

Jacobi 迭代

Jacobi 迭代是最简单直接的迭代方法，我们可以用一下这个例子来说明 Jacobi 迭代法。

Example 8

解方程组

$$\begin{aligned} 3u + v &= 5 \\ u + 2v &= 5. \end{aligned} \tag{3}$$

我们对这个方程组作等价变换，使得第一个方程左边只留下 u，第二个方程左边只留下 v 得到

$$\begin{aligned} u &= \frac{5 - v}{3} \\ v &= \frac{5 - u}{2}. \end{aligned} \tag{4}$$

Jacobi 迭代

很显然，求方程组(9)的解等价于求(4)的不动点。如果我们将 $(u_0, v_0) = (0, 0)$ 作为初始值进行迭代，将会得到如下计算结果

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} \frac{5-v_0}{3} \\ \frac{5-u_0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-0}{3} \\ \frac{5-0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} \\ \frac{5}{2} \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} \frac{5-v_1}{3} \\ \frac{5-u_1}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-5/2}{3} \\ \frac{5-5/3}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{6} \\ \frac{5}{3} \end{bmatrix}$$

$$\begin{bmatrix} u_3 \\ v_3 \end{bmatrix} = \begin{bmatrix} \frac{5-5/3}{3} \\ \frac{5-5/6}{2} \end{bmatrix} = \begin{bmatrix} \frac{10}{9} \\ \frac{25}{12} \end{bmatrix}.$$

后续计算结果见 Matlab 程序 Run Jacobi 1。

Jacobi 迭代

如果我们对这个方程组作另一种等价变换，使得第一个方程左边只留下 v，第二个方程左边只留下 u 得到

$$\begin{aligned} u &= 5 - 2v \\ v &= 5 - 3u. \end{aligned} \tag{5}$$

同样将 $(u_0, v_0) = (0, 0)$ 作为初始值进行迭代，将会得到如下计算结果

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} 5 - 2v_0 \\ 5 - 3u_0 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$
$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} 5 - 2v_1 \\ 5 - 3u_1 \end{bmatrix} = \begin{bmatrix} -5 \\ -10 \end{bmatrix}$$
$$\begin{bmatrix} u_3 \\ v_3 \end{bmatrix} = \begin{bmatrix} 5 - 2(-10) \\ 5 - 3(-5) \end{bmatrix} = \begin{bmatrix} 25 \\ 20 \end{bmatrix}.$$

后续计算结果见 Matlab 程序 Run Jacobi 2（迭代 12 次达到给定精度）。

Jacobi 迭代收敛性的判定

很明显，第一个迭代产生的向量序列是收敛的，而第二种迭代产生的向量序列是发散的。与解非线性方程的不动点迭代类似，我们需要迭代收敛的判定定理。我们首先有以下定义。

Definition 9

一个 $n \times n$ 的矩阵 $A = (a_{ij})$，如果对任意 $1 \leq i \leq n$，对角线上的元素 $|a_{ii}| > \sum_{i \neq j} |a_{ij}|$，则这个矩阵 A 被称为严格对角占优的。换句话说，对角占优的矩阵在第 i 行中对角线上的元素的绝对值大于这一行中其余所有元素的绝对值的和。

我们有以下定理：

Theorem 10 (Jacobi 迭代收敛的充分条件)

如果 $n \times n$ 的矩阵 A 是严格对角占优的，那么

1. *A 是非奇异的（满秩的、可逆的）；*
2. *对任意的 b 和任意的初始猜测，利用 Jacobi 迭代解方程组 $Ax = b$ 得到的结果都是收敛的。*

Jacobi 迭代收敛性的判定

如果我们考虑由(4)和(5)所定义的迭代函数，可以发现这两个函数实际上都是由 $Ax = b$ 等价变换使第 i 个方程左端只剩下第 i 个未知数得到的，只是对于第一个迭代函数

$$A = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix},$$

而对于第二个迭代函数

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix}$$

很明显，对于第一个方程组系数矩阵是严格对角占优的，而对于第二个方程组系数矩阵不是严格对角占优的。

Remark 3. *方程组对角占优是 Jacobi 迭代收敛的充分条件而非必要条件，虽然第二个方程组所导出的迭代函数产生了发散的向量序列，但这不代表所有不是严格对角占优的矩阵所导出的 Jacobi 迭代函数都会产生发散的序列。*

Jacobi 迭代收敛性的判定

考虑一下两个矩阵

$$A = \begin{bmatrix} 3 & 1 & -1 \\ 2 & -5 & 2 \\ 1 & 6 & 8 \end{bmatrix}$$

$$B = \begin{bmatrix} 3 & 2 & 6 \\ 1 & 8 & 1 \\ 9 & 2 & -2 \end{bmatrix}$$

很明显，矩阵 A 是严格对角占优的，而矩阵 B 不是严格对角占优的。但是如果我们将 B 矩阵的第一行和第三行交换位置，则新的 B 矩阵就变成了严格对角占优的矩阵。

Jacobi 迭代的矩阵形式

Jacobi 迭代函数产生的机制就是将 $Ax = b$ 作等价变换，使第 i 个方程左端只剩下第 i 个未知数，这时方程组的右端就是迭代函数。这个过程也可以用矩阵的形式表示出来。

我们令 D 矩阵对角线上的元素与系数矩阵 A 的对角线上的元素相等，其余部分为 0；L 矩阵在对角线以下的元素与矩阵 A 对角线以下的元素相等，其余部分为 0；U 矩阵在对角线以上的元素与矩阵 A 对角线以上的元素相等，其余部分为 0。我们有

$$A = L + D + U, \tag{6}$$

进而原方程组可以写为

$$Ax = Lx + Dx + Ux = b. \tag{7}$$

对这个方程组作等价变换得到

$$\begin{aligned} Lx + Dx + Ux &= b \\ Dx &= b - (L + U)x \\ x &= D^{-1}(b - (L + U)x). \end{aligned} \tag{8}$$

Jacobi 迭代的算法

一个对角矩阵 D 的逆也是一个对角矩阵 D^{-1}，其中 D^{-1} 中对角线上的元素就是 D 的对角矩阵角线上相应位置的元素的倒数，因此求 D^{-1} 的运算量非常小，因此 Jacobi 迭代的算法如下

Algorithm 4 (Jacobi 迭代算法).
设 x^0 为初始猜测
for $k = 0, 1, 2, \ldots$
$\quad x^{k+1} = D^{-1}(b - (L + U)x^k)$
end

Jacobi 迭代的算法

例如对于我们讨论的第一个例子，

$$\begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}=\begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

则 Jacobi 迭代可以如下算法求得

$$\begin{aligned}\begin{bmatrix} u_{k+1} \\ v_{k+1} \end{bmatrix} &= D^{-1}(b-(L+U)x_k) \\ &= \begin{bmatrix} 1/3 & 0 \\ 0 & 1/2 \end{bmatrix}\left(\begin{bmatrix} 5 \\ 5 \end{bmatrix}-\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} u_k \\ v_k \end{bmatrix}\right) \\ &= \begin{bmatrix} (5-v_k)/3 \\ (5-u_k)/2 \end{bmatrix},\end{aligned}$$

与我们直接移项得到的形式一致。

Gauss-Seidel 迭代和超松弛迭代

Jacobi 迭代的一个变型就是 Gauss-Seidel 迭代，仍然考虑解方程组

$$\begin{aligned} 3u + v &= 5 \\ u + 2v &= 5. \end{aligned} \tag{9}$$

对这个方程组使用 Gauss-Seidel 迭代得到

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} \frac{5-v_0}{3} \\ \frac{5-u_1}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-0}{3} \\ \frac{5-5/3}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} \\ \frac{5}{3} \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} \frac{5-v_1}{3} \\ \frac{5-u_2}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-5/3}{3} \\ \frac{5-10/9}{2} \end{bmatrix} = \begin{bmatrix} \frac{10}{9} \\ \frac{35}{18} \end{bmatrix}$$

$$\begin{bmatrix} u_3 \\ v_3 \end{bmatrix} = \begin{bmatrix} \frac{5-v_2}{3} \\ \frac{5-u_3}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-35/18}{3} \\ \frac{5-55/54}{2} \end{bmatrix} = \begin{bmatrix} \frac{55}{54} \\ \frac{215}{108} \end{bmatrix}.$$

后续计算结果可见 Matlab 程序，在这个例子中 Gauss-Seidel 迭代的收敛速度（6 次迭代达到给定精度）比 Jacobi 迭代要快一些，这在大多数情况下是成立的，但并不是绝对的。

Gauss-Seidel 迭代与 Jacobi 迭代

Gauss-Seidel 迭代和 Jacobi 迭代形式上非常类似，其的不同之处在于:

- 在作等价变换使第 i 个方程左端只剩下第 i 个未知数之后，Jacobi 迭代直接将右端的函数作为迭代函数，计算时，右端函数中的未知数的值都代为上一步迭代的计算结果;
- 而 Gauss-Seidel 迭代中，每当左端的某一个未知量的值有了更新就会马上被代入到右端去进行计算。

与 Jacobi 迭代一样，我们有

Theorem 11 (Gauss-Seidel 迭代收敛的充分条件)

如果 $n \times n$ 的矩阵 A 是严格对角占优的，那么

1. *A 是非奇异的（满秩的、可逆的）;*
2. *对任意的 b 和任意的初始猜测，利用 Gauss-Seidel 迭代解方程组 $Ax = b$ 得到的结果都是收敛的。*

Gauss-Seidel 迭代的矩阵形式

与 Jacobi 迭代一样，Gauss-Seidel 迭代也可以写为矩阵形式，令原方程组为

$$Ax = Lx + Dx + Ux = b, \tag{10}$$

则 Gauss-Seidel 迭代可以写为

$$\begin{aligned}(L + D)x^{k+1} &= b - Ux^k \\ x^{k+1} &= (L + D)^{-1}(b - Ux^k).\end{aligned} \tag{11}$$

但是这里要注意一个问题，实际计算中我们是不会用到(11)，这是因为求 $(L + D)^{-1}$ 的计算量很大。一般来说，求任意矩阵逆的运算量是 $O(n^3)$，相当于作一次高斯消元法的计算量（实际上，计算机求矩阵逆的算法基本都要先作一次高斯消元）。

Gauss-Seidel 迭代的算法

为了避免求矩阵的逆，Gauss-Seidel 迭代在实际计算中一般不直接用矩阵相乘，而是一个未知量一个未知量的更新，但形式上我们有

Algorithm 5 (Gauss-Seidel 迭代算法).
设 x^0 为初始猜测
for $k = 0, 1, 2, \ldots$
$\quad x^{k+1} = D^{-1}(b - Ux^k - Lx^{k+1})$
end
我们再次强调这个算法实际上是一个式子一个式子进行的，而非直接利用矩阵与向量的乘积。

Gauss-Seidel 迭代的算法

例如对于线性方程组

$$\begin{bmatrix} 3 & 1 & -1 \\ 2 & 4 & 1 \\ -1 & 2 & 5 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix}$$

利用 Gauss-Seidel 迭代，得到

$$\begin{aligned} u_{k+1} &= \frac{4 - v_k + w_k}{3} \\ v_{k+1} &= \frac{1 - 2u_{k+1} - w_k}{4} \\ w_{k+1} &= \frac{1 + u_{k+1} - 2v_{k+1}}{5}. \end{aligned}$$

计算结果可见 Matlab 程序，由于系数矩阵对角占优，所以最终结果（迭代 11 次）收敛到 $[2, -1, 1]^T$。

超松弛迭代（SOR）

如果我们将 Gauss-Seidel 迭代的右端项与迭代的上一步结果作一个线性组合，就得到了松弛迭代，例如对于上面考虑的方程组，我们得到

$$
\begin{aligned}
u_{k+1} &= (1-\omega)u_k + \omega\frac{4 - v_k + w_k}{3} \\
v_{k+1} &= (1-\omega)v_k + \omega\frac{1 - 2u_{k+1} - w_k}{4} \\
w_{k+1} &= (1-\omega)w_k + \omega\frac{1 + u_{k+1} - 2v_{k+1}}{5}.
\end{aligned}
$$

这里 ω 称为松弛系数，如果这个松弛系数大于 1，则这个迭代称为超松弛迭代。使用超松弛迭代的原因是我们希望在保持迭代为不动点迭代的前提下，利用 Gauss-Seidel 迭代收敛速度快的优势加速迭代的收敛。例如在上面的例子中令 $\omega = 1.25$，我么可以利用 Matlab 来观察计算过程（迭代 7 次就达到要求的精度）。

（超）松弛迭代（SOR）的矩阵形式及算法

对于（超）松弛迭代，我们实际上是在原方程组两边同时乘以了一个系数 ω，得到

$$
\begin{aligned}
(\omega L+\omega D+\omega U)x &= \omega b\\
(\omega L+D)x &= \omega b-\omega Ux+(1-\omega)Dx\\
x &= (\omega L+D)^{-1}[(1-\omega)Dx-\omega Ux]+\omega(D+\omega L)^{-1}b.
\end{aligned}
\tag{12}
$$

我们进而得到 SOR 的算法

Algorithm 6 (SOR 迭代算法).
设 x^0 为初始猜测
for $k=0,1,2,\ldots$
$\quad x^{k+1}=(\omega L+D)^{-1}[(1-\omega)Dx^k-\omega Ux^k]+\omega(D+\omega L)^{-1}b$
end

三种算法的比较

如果我们利用三种迭代解线性方程组

$$\begin{bmatrix} 3 & -1 & 0 & 0 & 0 & \frac{1}{2} \\ -1 & 3 & -1 & 0 & \frac{1}{2} & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & 0 \\ 0 & \frac{1}{2} & 0 & -1 & 3 & -1 \\ \frac{1}{2} & 0 & 0 & 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ \frac{3}{2} \\ 1 \\ 1 \\ \frac{3}{2} \\ \frac{5}{2} \end{bmatrix}.$$

迭代六步之后我们得到的结果见下表

Jacobi	Gauss–Seidel	SOR
0.9879	0.9950	0.9989
0.9846	0.9946	0.9993
0.9674	0.9969	1.0004
0.9674	0.9996	1.0009
0.9846	1.0016	1.0009
0.9879	1.0013	1.0004

而真实解为 $x = [1, 1, 1, 1, 1, 1]^T$

三种算法的比较

在这个例子中，SOR 迭代相对较好。但是我们再次强调，每一种算法都不是一定比其它算法好，甚至对于同一个问题会出现有些算法收敛而其它算法发散的情况，因此要根据实际情况进行判断算法的优劣性。对于 SOR 算法中 ω 的取值也没有一个对任何问题都有效的判定标准，只有一些特殊问题可以理论上得到最优的 ω 取值。

使用迭代法的必要性

高斯消元给出了有限步计算得到精确解的算法，而迭代法的收敛性和迭代步数都是不确定，那么为什么我们还要探讨迭代法呢？

最重要的原因事实上是计算量。牛顿迭代，包括 LU 分解的计算量都是 $O(n^3)$，而迭代法中，每一步迭代（相当于作几次矩阵与向量相乘）的计算量最多是 $O(n^2)$，如果 n 很大，而达到足够精度的迭代步数远小于 n 的话，那么一次迭代法的计算量就比高斯消元要小。特别是在以下两种情况之下迭代法都是好的选择。

A 与 b 的微小变化

如果我们需要解一系列 $Ax = b$ 的问题，而 A 与 b 都在不断变化，但变化量很小，那么这时候迭代法就比消元试用了，因为上一次利用迭代法得到的计算结果就可以作为下一次迭代法的初值。例如解之前的方程组

$$\begin{bmatrix} 3 & -1 & 0 & 0 & 0 & \frac{1}{2} \\ -1 & 3 & -1 & 0 & \frac{1}{2} & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & 0 \\ 0 & \frac{1}{2} & 0 & -1 & 3 & -1 \\ \frac{1}{2} & 0 & 0 & 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ \frac{3}{2} \\ 1 \\ 1 \\ \frac{3}{2} \\ \frac{5}{2} \end{bmatrix}.$$

如果 b 从 $b = [2.5, 1.5, 1, 1, 1.5, 2.5]^T$ 变为 $b = [2.2, 1.6, 0.9, 1.3, 1.4, 2.45]^T$，可以验证这个方程组的解从 $x = [1, 1, 1, 1, 1, 1]^T$ 变为 $x = [0.9, 1, 1, 1, 1, 1]^T$。

A 与 b 的微小变化

如果我们以上一次迭代的第六步结果作为迭代初值，则利用 Gauss-Seidel 迭代再迭代两步后的结果是

x_7	x_8
0.8980	0.8994
0.9980	0.9889
0.9659	0.9927
1.0892	1.0966
0.9971	1.0005
0.9993	1.0003

这种方法称为润色，在实际计算中有很多的应用，尤其是当数据不断更新的情况。

稀疏矩阵的处理

迭代法在矩阵为稀疏矩阵的情况下有更大的作用。如果一个 $n \times n$ 的矩阵非零元素的个数为 $O(n)$ 的话，则这个矩阵称为稀疏矩阵；而如果一个 $n \times n$ 的矩阵非零元素的个数为 $O(n^2)$ 的话，则这个矩阵称为稠密矩阵。

稀疏矩阵虽然非零元素比较少，但是如果对稀疏矩阵作 $PA = LU$ 分解的话，很容易出现填充（fill-in）现象，也就是说一个稀疏矩阵的 $PA = LU$ 分解出来的 L 与 U 往往不再是稀疏的，这样即便得到了 $PA = LU$ 分解也需要 $O(n^2)$ 的运算才能得到方程的解。而如果对于稀疏矩阵作迭代的话每次计算量是 $O(n)$。这样如果迭代收敛速度比较快的话，可以省下很多的计算量。

稀疏矩阵的处理

例如我们解 $100,000$ 阶的类似于上面的例子中的方程组，则稀疏矩阵的形式大致是（例如 $n = 12$）

$$A = \begin{bmatrix}
3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\
-1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\
0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\
0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 3 & -1 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{2} & 0 & -1 & 3 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 \\
0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 \\
0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 \\
\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3
\end{bmatrix}.$$

而右端项为 $b = [2.5, 1.5, \ldots, 1.5, 1.0, 1.0, 1.5, \ldots, 1.5, 2.5]^T$。

稀疏矩阵的处理

如果我们把这个矩阵的所有元素，包括零元素储存下来，则这个矩阵有 $n^2 = 10^{10}$ 个元素，如果没有元素用 IEEE 双精度浮点数储存，每个数的储存需要 8 个字节，则一共需要 8×10^{10} 字节，这大约是 80 个 G 的储存量。一般台式机的内存是无法存下这个矩阵的。

除此之外，对这个矩阵作 $PA = LU$ 分解需要 $n^3 \approx 10^{15}$ 次运算，如果你的 CPU 是 1 GHz（大约 10^9 个周期每秒），则浮点运算速度的上界大约为 10^8 次计算每秒。那么这个 $PA = LU$ 分解需要 10^7 秒，这大概等于 4 个月的时间。

而对于这个方程组作一次迭代运算，其运算量大概是 $800,000 \approx 10^6$ 次，即便我们作 100 步的迭代，我们也只需要 1 秒的时间。而对于上面我们考虑的问题，实际上迭代法只需要 27 步就可以从初始猜测 $x^0 = [0, 0, \ldots, 0, 0]$ 收敛到真实解 $[1, 1, \ldots, 1, 1]$。计算程序和结果见 Matlab 程序，需要注意我们使用了 Matlab 的 spdiags 函数用以生成稀疏矩阵，在 Matlab 中，稀疏矩阵的存储量大约只有稀疏矩阵非零元素的个数，即 $O(n)$。

求解非线性方程组

求解线性方程组有许多重要的应用，其中之一是求解非线性方程组。我们上一讲中讨论了如何解非线性方程，我们现在讨论如何求解非线性方程组。我们以一个由三个方程组成的三元方程组为例来探讨这个问题。例如求解 $F(u,v,w)=(f_1,f_2,f_3)^T=0$，或 $F(x)=0$，$x=[u,v,w]^T$，即

$$\begin{aligned} f_1(u,v,w)&=0\\ f_2(u,v,w)&=0\\ f_3(u,v,w)&=0. \end{aligned} \tag{13}$$

我们首先定义 $F(x)$ 的 Jacobi 矩阵为

$$DF(x)=\begin{bmatrix} \frac{\partial f_1}{\partial u} & \frac{\partial f_1}{\partial v} & \frac{\partial f_1}{\partial w}\\ \frac{\partial f_2}{\partial u} & \frac{\partial f_2}{\partial v} & \frac{\partial f_2}{\partial w}\\ \frac{\partial f_3}{\partial u} & \frac{\partial f_3}{\partial v} & \frac{\partial f_3}{\partial w} \end{bmatrix}.$$

Jacobi 矩阵实际上是一元函数导数在多元方程组中的推广。

求解非线性方程组的牛顿迭代

求解非线性方程组的基本方法就是牛顿迭代，也就是说，我们要把一元方程的牛顿迭代

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^{k+1})} \tag{14}$$

推广到多元方程组中去。我们首先利用多元方程组在 $x_0 = [u_0, v_0, w_0]^T$ 处的 Taylor 展开，得到

$$F(x) = F(x_0) + DF(x_0)(x - x_0) + O(\|x - x_0\|^2). \tag{15}$$

这里 $DF(x_0)(x - x_0)$ 表示矩阵 $DF(x_0)$ 与向量 $x - x_0$ 相乘

求解非线性方程组的牛顿迭代

例如，如果 $F(u,v)=(e^{u+v},\sin u)$，则在 $x_0=(0,0)$ 附近有

$$
\begin{aligned}
F(x) &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} e^0 & e^0 \\ \cos 0 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + O(x^2) \\
&= \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} u+v \\ u \end{bmatrix} + O(x^2).
\end{aligned}
$$

求解非线性方程组的牛顿迭代

非线性方程组的牛顿迭代构造的基本思想与非线性方程的一样，都是忽略高阶项 $O(x^2)$。若令 $x = r$ 是非线性方程组的根，则

$$0 = F(r) \approx F(x_0) + DF(x_0)(r - x_0), \tag{16}$$

移项得

$$x_0 - DF(x_0)^{-1}F(x_0) \approx r. \tag{17}$$

具体的算法是

Algorithm 7 (非线性方程组的牛顿代算法).
设 x^0 为初始猜测
for $k = 0, 1, 2, \ldots$
$\quad x^{k+1} = x^k - DF(x^k)^{-1}F(x^k)$
end

求解非线性方程组的牛顿迭代

由于求矩阵的逆计算量大约是高斯消元的 3 倍，因此在实际计算中我们并不求 $DF(x^k)^{-1}$，而是求解关于 s 的方程组

$$DF(x^k)s = -F(x^k), \tag{18}$$

之后再计算

$$x^{k+1} = x^k + s. \tag{19}$$

Example 12

求解方程组

$$\begin{aligned} v - u^3 &= 0 \\ u^2 + v^2 - 1 &= 0 \end{aligned} \tag{20}$$

求解非线性方程组的牛顿迭代举例

以上方程组的解可以由下图大致表示出来

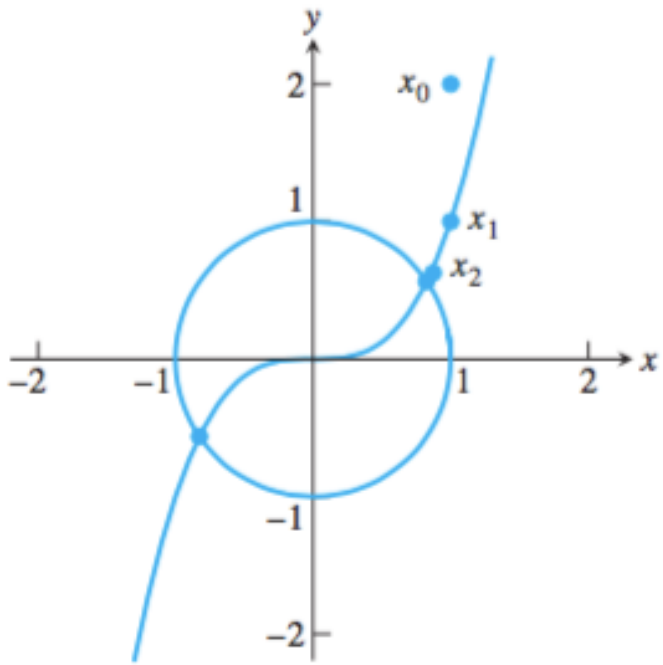

Figure 1: $f_1(u,v) = v - u^3 = 0$ 和 $f_2(u,v) = u^2 + v^2 - 1 = 0$ 的图像

求解非线性方程组的牛顿迭代举例

为了精确求解原线性方程组，我们首先找到它的 Jacobi 矩阵

$$DF(u,v)=\begin{bmatrix} -3u^2 & 1 \\ 2u & 2v \end{bmatrix}.$$

如果我们以 $x_0=[1,2]^T$ 为初值，则第一步中，我们首先要解

$$\begin{bmatrix} -3 & 1 \\ 2 & 4 \end{bmatrix}\begin{bmatrix} s_1 \\ s_2 \end{bmatrix}=-\begin{bmatrix} 1 \\ 4 \end{bmatrix}.$$

得到 $s=[0,-1]^T$，进而 $x^1=x^0+s=[1,1]^T$.

求解非线性方程组的牛顿迭代举例

第二步我们更新 Jacobi 矩阵和 $F(x)$ 的值，得到

$$\begin{bmatrix} -3 & 1 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = -\begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

计算得 $s = [-\frac{1}{8}, -\frac{3}{8}]^T$，进而 $x^2 = x^1 + s = [\frac{7}{8}, \frac{5}{8}]^T$. 后续计算见下表

step	u	v
0	1.00000000000000	2.00000000000000
1	1.00000000000000	1.00000000000000
2	0.87500000000000	0.62500000000000
3	0.82903634826712	0.56434911242604
4	0.82604010817065	0.56361977350284
5	0.82603135773241	0.56362416213163
6	0.82603135765419	0.56362416216126
7	0.82603135765419	0.56362416216126

如果我们从 $[-1, -2]^T$ 开始迭代就会收敛到 $y-$ 轴左边的根，具体计算结果见 Matlab 程序。

求解非线性方程组的牛顿迭代举例

Example 13

求解方程组

$$\begin{aligned} f_1(u,v) &= 6u^3 + uv - 3v^3 - 4 = 0 \\ f_2(u,v) &= u^2 - 18uv^2 + 16v^3 + 1 = 0 \end{aligned} \tag{21}$$

首先求 Jacobi 矩阵

$$DF(u,v) = \begin{bmatrix} 18u^2 + v & u - 9v^2 \\ 2u - 18v^2 & -36uv + 48v^2 \end{bmatrix}.$$

求解非线性方程组的牛顿迭代举例

从初始猜测 $[u_0, v_0] = [2, 2]^T$ 开始得到如下结果

step	u	v
0	2.00000000000000	2.00000000000000
1	1.37258064516129	1.34032258064516
2	1.07838681200443	1.05380123264984
3	1.00534968896520	1.00269261871539
4	1.00003367866506	1.00002243772010
5	1.00000000111957	1.00000000057894
6	1.00000000000000	1.00000000000000
7	1.00000000000000	1.00000000000000

从其它点上迭代得到的根见 Matlab 程序。

课后阅读

[NA] 第 2 章：2.5.1, 2.5.2, 2.5.3;2.6.1,2.6.2.

作业：
[(1)] 2.5:2,4,5; 2.6:8,14.
[(2)] 2.5:1,5; 2.6:2,3.