前沿计算研究实践（II）	**2019 年春季**

专题 5：Solution of Linear Systems

教授：陈宝权　　作者：马龙，赵怡浩

免责声明：该笔记尚未进行正式出版物的通常审查，未经教师允许不能向课程外分发.

5.1 背景

在应用几何学和其他应用领域经常会遇到的计算瓶颈是求解线性系统的解：

$$A\mathbf{x} = \mathbf{b}$$

其中，$A = \{a_{i,j}\} \in \mathbb{R}^{n*n}, \mathbf{x}, \mathbf{b} \in \mathbb{R}^n$. 当 A 可逆时，这个线性系统对任意 $\mathbf{b}$ 有唯一解：$\mathbf{x} = A^{-1}\mathbf{b}$. 求解线性系统的数值方法有直接法（有限步内求得精确解）和迭代法（从初始猜测开始，利用可接受的算法迭代求得近似解）等. 通常我们从多个方面衡量求解方法的质量，主要的有这样几个：

1. 求解速度；
2. 存储要求；
3. 解的期望准确度.

矩阵的维数、稀疏性、对称、正定等性质都是影响解法选择的重要因素.

5.2 直接法

5.2.1 高斯消元和 LU 分解

我们知道，对线性方程组进行基本行变换可以简化方程，同时不改变解. 高斯消元的主要想法是：通过基本行变换化为上三角，再回带得到解，即分解成 $A = LU$，其中 L 是单位下三角矩阵，U 是上三角矩阵. 高斯消元过程即:

1. LU 分解：$A = LU$；
2. 前向替换求解：$L\mathbf{y} = \mathbf{b}$；
3. 回带求解：$U\mathbf{x} = \mathbf{y}$.

Algorithm 1 LU 分解，L 存在 A 对角线以下部分（不含），U 存在 A 对角线及以上部分

```
for k = 1 to n − 1 do
    for i = k + 1 to n do
        a_{i,k} = a_{i,k} / a_{k,k}
        for j = k + 1 to n do
            a_{i,j} = a_{i,j} − a_{i,k} a_{k,j}
        end for
    end for
end for
```

复杂度分析：LU 分解 $O(n^3)$, 前向和回带求解 $O(n^2)$. 实际实现中通常将 $\mathbf{b}$ 看作 A 的第 $n+1$ 列，在得到 U 的同时得到 $\mathbf{y}$.

5.2.2 数值稳定性

若对角线上某个 $a_{k,k}=0$ 或 $\epsilon \ll 1$，会导致 LU 分解中 $\frac{a_{i,k}}{a_{k,k}}$ 带来严重误差. 修正 LU 分解可以采用 Gaussian Elimination with Partial Pivoting（GEPP）. 思路是：通过置换将剩下行中当前列绝对值最大的换到当前行来避免上述情况. 则原问题变为 $PA=LU$，P 是置换矩阵. 通常情况下可以避免大误差，但仍可以构造出适当的矩阵 A 使得 GEPP 失效.[1]
MATLAB 中，运算 $\mathbf{x}=A\backslash\mathbf{b}$ 时默认使用 GEPP，也可以通过 $[L,U,P]=lu(A)$ 得到 A 的 LU 分解.

5.2.3 针对对称正定矩阵的 Cholesky 分解

满足 $A=A^T, \forall \mathbf{x}\neq\mathbf{0}, \mathbf{x}^TA\mathbf{x}>0$ 的矩阵 A 称为对称正定（symmetric positive-definite，SPD）矩阵. 对 SPD 矩阵存在常数更小（优化到 $\frac{1}{3}n^3$）、更鲁棒，且不需要做 pivoting 的分解 Cholesky 分解：

$$A=FF^T$$

F 是下三角矩阵，由对称性可知，对应的上三角矩阵为 F^T. Cholesky 分解可以扩展到复数域上，应用中我们主要在实数域中考虑.

引理 5.1 *实矩阵 A 能唯一 Cholesky 分解的充分必要条件是 A 对称且正定.*

证明：（充分性）由 A 的 LDU 分解可知，$A=LDU$，其中 L 和 U 分别是对角元为 1 的下上三角，D 是对角元素为正的对角矩阵（由 A 正定可知）. 由 $U^TDL^T=A^T=A=LDU$ 可知 $U^T=L$，同时可以写做 $A=LDL^T=(LD^{1/2})(D^{1/2}L^T)=FF^T$，其中 $F=LD^{1/2}$ 是下三角矩阵.

（必要性）$A^T = (FF^T)^T = FF^T = A, \forall \mathbf{x} \neq \mathbf{0}, \mathbf{x}^T A\mathbf{x} = \mathbf{x}^T FF^T \mathbf{x} = ||F^T\mathbf{x}|| > 0.$ ■

计算可知，

$$f_{i,j} = \begin{cases} \sqrt{a_{ij} - \sum_{k=1}^{i-1} f_{ik}^2}, & i = j \\ \frac{a_{ij} - \sum_{k=1}^{j-1} f_{ik} f_{jk}}{fjj}, & i > j \\ 0, & i < j \end{cases} \tag{5.1}$$

Algorithm 2 SPD 矩阵 A 的 Cholesky 分解，F 存在 A 的对角线及以下部分

```
for k = 1 to n do
    a_{k,k} = sqrt(a_{k,k})
    for i = k + 1 to n do
        a_{i,k} = a_{i,k} / a_{k,k}
    end for
    for j = k + 1 to n do
        for i = j to n do
            a_{i,j} = a_{i,j} - a_{i,k} a_{k,j}
        end for
    end for
end for
```

5.2.4 稀疏矩阵的情况

稀疏矩阵通常存储非零元素的位置和元素值即可. 稀疏矩阵在高斯消元/LU 分解中的复杂度也会发生变化.

5.2.4.1 稀疏带形矩阵

三对角矩阵是只有三条对角线上元素（$a_{i,i}, a_{i,i-1}, a_{i-1,i}$）非 0 的矩阵，其高斯消元的复杂度为 $O(n)$，空间复杂度也为 $O(n)$. 类似的，只有几条离主对角线近的对角线上元素非 0 的矩阵的复杂度也被大大降低.

5.2.4.2 高斯消元和稀疏性

当矩阵是带状并且带形区域有一定稀疏性，或者矩阵是稀疏的，但非零元素的位置没有规律时，不同解法会有不同的表现. 考虑网格上的离散二维拉普拉斯矩阵，如图 6.1 所示. 每个点只与其上下左右及自己有关（对格点按第一维坐标第一关键字、第二维坐标第二关键字排序后编号），故成线条带形，非零元约有 $5n$ 个，但进行 Cholesky 分解后有大约 $n\sqrt{n}$ 个非零元，复杂度是 $O(n^2)$.

考虑原矩阵是稀疏的，但其逆矩阵几乎总是稠密矩阵，如 Laplacian 矩阵的 Cholesky 因子是稀疏的，但逆矩阵是稠密的（图 6.2），这个现象被称为非零元填充（fill-in）. 因此，分解因子和求逆求解线性方程组的方法是不同的.

这一问题可以通过合理的重排序策略（reordering strategy）来加以缓解. 对于 Cholesky 分解，我们考虑寻找稀疏矩阵 M，使得 $A = M^T M$ 且 $M^T M \backslash A$ 包含极少非零元. 已有的方法大多与 A 的超图有关.

定义 5.2 超图 $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ 是由顶点集 $\mathcal{V}$ 和超边集 $\mathcal{E}$ 组成的，每条超边是顶点的集合.

定义 5.3 矩阵 A（n*m）的超图有 n 个顶点 $v_1, ..., v_n$，m 条边 $\epsilon_1, ..., \epsilon_m$，$v_i \in epsilon_j$ *iff* $a_{i,j} \neq 0$

M 是 A 的超图的关联矩阵（incidence graph）时，$M^T M \backslash A = \emptyset$，但 M 的非零元会有 A 的两倍. 还有一种方法是最小化 M 的行数，满足 $M^T M \backslash A = \emptyset$，已证明这等价于找到图的最小团覆盖，是 NPC 问题. 但可以通过限制每个团的大小得到有效的算法. 针对一般矩阵的 LU 分解，寻找 C, B 满足 $C^T B \backslash A$ 的非零元尽量少，也有相应的算法. [2]
书中还指出一种著名的 Cuthill-McKee 算法，从更直观的角度，通过将对称矩阵的行列置换后，使得矩阵的带宽尽量小，这是 NPC 问题，但存在一些算法得到较优解. 思路是：把矩阵看作一个图的邻接矩阵，通过重新编号来减小邻接矩阵的带宽. 实现步骤如下. 将最后得到的节点编号序列倒序，就可以得

Algorithm 3 Cuthill-McKee Algorithm. Input：对称矩阵 A. Output：重排后的矩阵 A ‘

```
R: n 个节点的新序列
Q: BFS 的队列
while 有节点未访问 do
    选度数最小的点 x 加入 R
    考虑 R_i 的连通子集 A_i 并排除所有已在 R 中的点 A_i ← Adj(R_i)\R 加入 Q
    while Q 非空 do
        取队首元素 C
        考虑 C 的连通子集 A_i 并排除所有已在 R 中的点 A_i ← Adj(R_i)\R 加入 Q
    end while
end while
返回 R 中点的邻接矩阵 A'
```

到 RCM 算法. 直观来看，我们希望有边的点编号相邻，最后得到的矩阵呈现两头窄的带形. RCM 只是将 CM 算法的结果进行了翻转.

5.3 迭代法

5.3.1 背景

直接法的局限性：

1. 可能引入非零元（如上面的 Laplacian 的例子）；
2. 无法利用已有的近似解的信息（如连续解多个具有相似性的线性方程组时）；
3. 线性系统不明确可知时解法无法使用（比如有时只能通过给出 x 求出 Ax，却不知道 A 的显式表示）.

这一节的迭代法可以在一定程度上解决这些问题.

5.3.2 基本的静态方法

对于 $A\mathbf{x} = \mathbf{b}$，拆分 $A = M - N$，则有 $M\mathbf{x} = N\mathbf{x} + \mathbf{b}$，可得到迭代 $M\mathbf{x}_{k+1} = N\mathbf{x}_k + \mathbf{b}$，原方程组的解是它的不动点，称为静态迭代（stationary iteration）. 其中 M 的选取应当易于求逆，且与 A 尽量接近以便于快速收敛，其中第一条是必要的，因此大多数静态方法更侧重于第一条. 若将 A 表示成 $A = D + E + F$，其中 D, E, F 分别是矩阵 A 的对角线、上三角和下三角，那么根据 M 的选取，有两种最基础的迭代方法：

1. Jacobi: $M = D$;
2. Gauss-Seidel: $M = D + E$.

Algorithm 4 静态迭代法. Input：M, N，$\mathbf{x}_0$；Output：ans

```
k ← 0
while k ≤ Iter or error ≥ Error do
    x_{k+1} = −D^{−1}[(E + F)x_k + b]
    k ← k + 1
end while
```

这些方法的收敛速度较慢，但优势体现在：

1. 易于编程；
2. 对简单问题很高效；

3. 收敛性分析较为简单；

4. 可以给出对某个线性方程组求解难易程度的直观估计.

这两种方法也有各自的特点. Jacobi 可以高度并行化（第 $k+1$ 次迭代仅与第 k 次迭代结果有关，可以分块并行计算），将 A 变换成主对角优势矩阵以帮助收敛；Gauss-Seidel 就不太方便并行化，但常用在多网格方法（粗糙化为低维问题后用 GS 快速收敛，再插值回去）.

5.3.3 算法的收敛性

引理 5.4 *定义复可对角化方阵 T 的**谱半径**（**spectral radius**）$\rho(T)$ 为 T 所有特征值的模长的最大值，那么 $\lim_{k\to\infty} T^k = 0$ iff $\rho(T) < 1$.*

证明：可知 $T = P^{-1}DP$，其中 P 是正交阵，D 是 T 的所有特征值构成的对角阵，那么 $T^k = P^{-1}D^kP$，由 $\lim_{k\to\infty} D^k = \operatorname{diag}\left\{\lambda_1^k, \lambda_2^k, \ldots, \lambda_n^k\right\} = 0$ iff 这些特征值的最大模长小于 1 可得. ■

推论 5.5 *当 $\rho(T) < 1$ 时，有 $(I-T)^{-1}$ 存在，$I + \sum_{k=1}^{\infty} T^k$ 收敛至 $(I-T)^{-1}$.*

证明：收敛性可以由上述引理的指数衰减直接得到，由于 1 不是 T 的特征值，$I-T$ 显然是可逆的. 考虑证明 $\lim_{k\to\infty}(I + \sum_{j=1}^{k} T^j)(I-T) = I$，展开得 $(I + \sum_{j=1}^{k} T^j)(I-T) = I - T^{k+1} \to I$ 即可. ■

定理 5.6 *对于形如 $\mathbf{x}_k = T\mathbf{x}_{k-1} + \mathbf{c}$ 的迭代过程，当矩阵 T 复可对角化且 $\rho(T) < 1$ 时，对于任意的初值 $\mathbf{x}_0 \in \mathbb{R}^n$，$\lim_{k\to\infty} \mathbf{x}_k$ 收敛至相同的解 $\mathbf{x} = T\mathbf{x} + \mathbf{c}$.*

证明：展开 $\mathbf{x}_k$ 可以得到 $T^k\mathbf{x}_0 + (\sum_{j=0}^{k-1} T^j)\mathbf{c}$，于是 $\lim_{k\to\infty} \mathbf{x}_k = \mathbf{0} + (I-T)^{-1}\mathbf{c}$ 收敛，且满足 $\mathbf{x} = T\mathbf{x} + \mathbf{c}$. ■

上述定理说明了，当迭代矩阵 $T = M^{-1}N$ 满足 $\rho(T) < 1$（且复可对角化，这一点几乎总是成立的）时，对于任意的初值，算法都必然收敛，且容易看出收敛速度与特征值的最大模长有关. Jacobi 与 Gauss-Seidel 算法在原矩阵 A 是**对角优势矩阵（diagonally dominant matrix）**时都满足这一条件，对于其它矩阵，收敛性可能得不到保证.

5.3.4 Krylov 子空间

Krylov 子空间定义在矩阵 A，初值 $\mathbf{r}_0$ 和阶数 k 上，$\mathcal{K}^k(A;\mathbf{r}_0) = \operatorname{span}\left\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \ldots, A^{k-1}\mathbf{r}_0\right\}$. 这一类方法的思想是，从初始解 $\mathbf{x}_0$ 出发，在集合 $\mathbf{x}_0 + \mathcal{K}^k(A;\mathbf{r}_0)$ 中按照某种估计选取一个尽可能优的当前解.

我们关注这一类方法的原因是，计算这个子空间只需要执行矩阵-向量乘法，很容易在稀疏矩阵的情形下在 O(非零元个数) 的复杂度内实现，避免了代价更高（或者，在某些场景中，矩阵本身不可知，从而无法计算）的矩阵-矩阵乘法.

5.3.4.1 Arnoldi 迭代

Arnoldi 迭代是基于施密特正交化的一个算法. 给定任意的单位向量初值 $\mathbf{v}_1$ 和阶数 k，算法输出 $\mathcal{K}^k(A;\mathbf{v}_0)$ 的一组标准正交基 $V_k = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k)$ 和一个上 Hessenberg 矩阵 $H_{k+1,k}$（只有上三角和下对角线有非零元素的矩阵），满足 $AV_k = V_{k+1}H_{k+1,k}$.

Algorithm 5 Arnoldi 迭代：输入单位长度向量 $\mathbf{v}_1$，输出 V_{k+1} 和 $H_{k+1,k}$

```
for j = 1 to k do
    z ← A v_j
    for i = 1 to j do
        h_{i,j} ← v_i^T z
        z ← z − h_{i,j} v_i
    end for
    h_{j+1,j} ← ||z||_2
    if h_{j+1,j} = 0 then
        exit
    end if
    v_{j+1} ← z / h_{j+1,j}
end for
```

复杂度显然是 $O(k^2n)$ 的. 当 Krylov 子空间无法继续扩张时，算法退出，此时下面提到的一些算法应当已经收敛，因此不需要考虑这个问题.

5.3.5 共轭梯度（conjugate gradient, CG）算法

以下假定要求解的方程组是 $A\mathbf{x} = \mathbf{b}$，其中 A 是 SPD 实矩阵. 我们将求解原方程组转化为一个最优化问题：令 $\phi(\mathbf{x}) = \mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$，求导容易得到这个函数的唯一极小值点就是 $A\mathbf{x} = \mathbf{b}$ 的解.

5.3.5.1 最速梯度下降（steepest gradient descent）算法

这个算法就是大家都很熟悉的梯度下降算法. 对于当前的解 $\mathbf{x}_k$，我们沿梯度 $\nabla_{\mathbf{x}}\phi(\mathbf{x}_k) = A\mathbf{x}_k - \mathbf{b}$ 的反方向走一步，步长为 α_k，而步长的选择应当最小化 $\phi(\mathbf{x}_k - \alpha_k(A\mathbf{x}_k - \mathbf{b}))$. 我们记 $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k = -\nabla_{\mathbf{x}}\phi(\mathbf{x}_k)$ 为此时的**残差（residual）**. 再次对 α_k 求导得到

$$
\begin{aligned}
\frac{d}{d\alpha_k} f(\mathbf{x}_{k+1}) &= f'(\mathbf{x}_{k+1}) \frac{d}{d\alpha_k} \mathbf{x}_{k+1} \\
&= -\mathbf{r}_{k+1}^T \mathbf{r}_k \\
&= (A\mathbf{x}_{k+1} - \mathbf{b})^T \mathbf{r}_k \\
&= [A(\mathbf{x}_k + \alpha_k \mathbf{r}_k) - \mathbf{b}]^T \mathbf{r}_k \\
&= \alpha_k (A\mathbf{r}_k)^T \mathbf{r}_k - \mathbf{r}_k^T \mathbf{r}_k \\
&= 0
\end{aligned}
$$

解得 $\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{r}_k^T A \mathbf{r}_k}$. 此处定义两个向量 $\mathbf{u}, \mathbf{v}$ 在矩阵 A 下的内积是 $\langle \mathbf{u}, \mathbf{v} \rangle_A = \mathbf{u}^T A \mathbf{v}$，内积 $\langle \cdot, \cdot \rangle_A$ 是一个对称双线性函数，上面的式子就可以写成 $\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\langle \mathbf{r}_k, \mathbf{r}_k \rangle_A}$.

5.3.5.2 共轭梯度算法

在最速梯度下降中，我们可能会在不同的时候向同一个梯度方向走很多步，这是我们不想要的. 一种叫做共轭方向（Conjugate Directions, CD）的算法试图解决这一问题，它强制每次选取的方向都与之前的所有方向**共轭**：定义两个向量 $\mathbf{u}, \mathbf{v}$ **共轭**当且仅当 $\langle \mathbf{u}, \mathbf{v} \rangle_A = 0$，类似于通常内积下的正交，由此定义 $\mathbb{R}^n$ 的一组**共轭基底** $(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n)$ 满足 $\langle \mathbf{p}_i, \mathbf{p}_j \rangle_A = 0, \forall i \neq j$. 设方程组的解为 $\mathbf{x}^* = \sum_{i=1}^n \alpha_i \mathbf{p}_i$，我们有 $A\mathbf{x}^* = \sum_{i=1}^n \alpha_i A \mathbf{p}_i$，对任意 k 左乘 $\mathbf{p}_k^T$ 得到 $\mathbf{p}_k^T \mathbf{b} = \sum_{i=1}^n \alpha_i \mathbf{p}_k^T A \mathbf{p}_i = \alpha_k \langle \mathbf{p}_k, \mathbf{p}_k \rangle_A, \forall k \in [n]$，解得 $\alpha_k = \frac{\mathbf{p}_k^T \mathbf{b}}{\langle \mathbf{p}_k, \mathbf{p}_k \rangle_A}$. 这意味着，只要求出了一组完整的共轭基底，每个分量的系数就可以被完全确定了.

从另一个角度看，记每一步的**误差向量** $\mathbf{e}_k = \mathbf{x}^* - \mathbf{x}_k$ 为当前解与目标解的差，直观地来看，它应当在当前搜索的整个解空间 $\mathbf{e}_0 + span\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k\}$ 中最小化 $||\mathbf{e}_k||_A = \sqrt{\langle \mathbf{e}_k, \mathbf{e}_k \rangle_A}$；展开后显然也可以得到每步清除 $\mathbf{e}_0$ 在 $\mathbf{p}_k$ 这个方向上的全部分量是最优的. 这与上面给出的结论等价.

当然，求解完整的基底意味着 $\Omega(n^2)$ 的时间复杂度，如果直接使用类似于施密特正交化的方法求解则需要 $\Theta(n^3)$ 的时间，因此我们实际想要的是找尽可能“好”的 k 个向量，在共轭化后表出近似的 $\mathbf{x}^*$，要求这个共轭化过程易于求解，并在 k 小于 n 的前提下做到足够近似. 作为 CD 的一种优化，CG 在这里选取了每步的目标函数梯度作为“好”的基底，它满足很多优秀的性质：

1. 选取梯度是利用局部信息所能做出的最好选择；
2. 梯度与之前的所有搜索方向正交，因此每次我们搜索的子空间都会扩大；
3. 选取梯度会降低共轭化的复杂度。

考虑 (2). 首先注意到 $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k = A\mathbf{x}^* - A\mathbf{x}_k = A\mathbf{e}_k$，而我们已经知道 $\mathbf{e}_k$ 不含有任何 $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k$ 的分量，任取 $\mathbf{p}_j, j \leq k$ 有 $\mathbf{p}_j^T \mathbf{r}_k = \mathbf{p}_j^T A \mathbf{e}_k = \sum_{i=k+1}^n c_i \langle \mathbf{p}_j, \mathbf{p}_k \rangle_A = 0$，于是 $\mathbf{r}_k$ 垂直于 $span\{\mathbf{p}_1, \ldots, \mathbf{p}_k\}$，记为 $\mathcal{D}_k$.

这个结论同时导出了 (3). Arnoldi 迭代告诉我们，$A\mathcal{D}_k$ 实际上是包含在 $\mathcal{D}_{k+1}$ 内的，因此 $\mathbf{r}_k$ 还垂直于

$A\mathcal{D}_{k-1}$，于是 A-共轭于 $\mathcal{D}_{k-1}$. 这意味着我们以 $\mathbf{r}_k$ 为基础共轭化得到下一个搜索方向 $\mathbf{p}_{k+1}$ 时，只需要对 $\mathbf{p}_k$ 一个向量做共轭化就可以了，其他向量的系数都是 0.

Algorithm 6 共轭梯度算法，输入 SPD 矩阵 A，向量 $\mathbf{b}$，迭代次数 k 和初值 $\mathbf{x}_0$，输出 k 次迭代后的近似解 $\mathbf{x}_k$

```
for j = 1 to k do
    s_{j-1} ← A p_{j-1}
    α_{j-1} ← (r_{j-1}^T r_{j-1}) / (p_{j-1}^T s_{j-1})
    x_j ← x_{j-1} + α_{j-1} p_{j-1}
    r_j ← r_{j-1} − α_{j-1} s_{j-1}
    β_{j-1} ← (−r_j^T s_{j-1}) / (p_{j-1}^T s_{j-1})
    p_j ← r_j + β_{j-1} p_{j-1}
end for
```

我们得到了非常优秀的 $O(km)$ 的复杂度，其中 m 是矩阵的非零元数量. 空间复杂度是 $O(n)$ 的（除矩阵 A 外）.

5.3.6 收敛性

我们知道

$$\begin{aligned}\mathbf{e}_k &\in \mathbf{e}_0 + \mathcal{D}_k \\ &= \mathbf{e}_0 + span\{\mathbf{r}_0, \ldots, \mathbf{r}_{k-1}\} \\ &= \mathbf{e}_0 + span\{\mathbf{r}_0, A\mathbf{r}_0, \ldots, A^{k-1}\mathbf{r}_0\} \\ &= \mathbf{e}_0 + span\{A\mathbf{e}_0, A^2\mathbf{e}_0, \ldots, A^k\mathbf{e}_0\}\end{aligned}$$

因此可以写成 $\mathbf{e}_k = \left(I + \sum_{j=1}^k \psi_j A^j\right)\mathbf{e}_0 = P_k(A)\mathbf{e}_0$ 的形式，其中 $P_k(x)$ 是一个 k 次多项式，且这个形式充要. 在 SPD 矩阵 A 的一组正交单位特征向量 $\mathbf{v}_j$ 下展开 $\mathbf{e}_0$，得到系数 η_j，我们得到

$$\begin{aligned}\mathbf{e}_k &= \textstyle\sum_{\lambda_j} \eta_j P_k(\lambda_j)\mathbf{v}_j \\ ||\mathbf{e}_k||_A^2 &= \textstyle\sum_{\lambda_j} \eta_j^2 [P_k(\lambda_j)]^2 \lambda_j \\ \min_{\mathbf{e}_k} ||\mathbf{e}_k||_A^2 &= \min_{P_k} \textstyle\sum_{\lambda_j} \eta_j^2 [P_k(\lambda_j)]^2 \lambda_j \\ &\leq \min_{P_k} \max_{\lambda^*} P_k(\lambda^*) \textstyle\sum_{\lambda_j} \eta_j^2 \lambda_j \\ &= \min_{P_k} \max_{\lambda^*} P_k(\lambda^*) ||\mathbf{e}_0||_A^2\end{aligned}$$

这意味着，CG 的收敛速度被最差的一个特征值的多项式值所限制，因此与特征值的分布有关. 当特征值分布非常密集时，少数几个根就已经足够限制住这个最大值了，CG 就收敛得很快. 对于 SPD 矩阵的情形，我们把一个矩阵的绝对值最大和最小的特征值的比值称为这个矩阵的**条件数**（condition number），条件数很大的矩阵称为 ill-conditioned.

针对这一问题，我们可以对矩阵进行**预处理**，把问题转化为 $M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}$，其中 M 称为 preconditioner，满足可逆且 $M^{-1}A$ 的收敛性较好（condition number 较小，或特征值分布较优秀，etc），然后解新问题即可，这叫做 left preconditioning. 同时也有 right preconditioning $AM^{-1}\mathbf{y} = \mathbf{b}, \mathbf{x} = M^{-1}\mathbf{y}$ 以及 split preconditioning $M_1^{-1}AM_2^{-1}\mathbf{y} = M_1^{-1}\mathbf{b}, \mathbf{x} = M_2^{-1}\mathbf{y}$. 由于我们需要 $M^{-1}A$ 是 SPD 矩阵，仅仅保证 M 是 SPD 可能是不够的. 不过我们可以分解 SPD 的 $M = EE^T$，然后解 $E^{-1}AE^{-T}\mathbf{x} = \mathbf{b}$，可以发现 $E^{-1}AE^{-T}$ 与 $M^{-1}A$ 有着相同的特征值.

一种简单的 preconditioning 技术是部分 Cholesky 分解（incomplete Cholesky factorization）. 我们试图对 A 矩阵执行 Cholesky 分解，但在这个过程中保持非零元集合不扩大，并把落到 A 的非零元集合外的元素都抹掉，得到 A 的一个近似分解 $\hat{L}^T\hat{L}$；或者，更进一步地，动态维护分解过程中每一行的非零元素，只保留其中最大的一部分，丢弃剩余的.

5.4 总结

求解线性方程组是一个很复杂的领域，需要根据矩阵的特点有针对性地使用相应的方法.

- 假如矩阵很小，或者很稠密：直接用 GEPP 一类的方法解方程通常是最好的，其他方法可能不能带来什么额外的收益. 特别地，如果是 SPD 矩阵，Cholesky 分解更好；如果对称但不正定，有一些 GEPP 的变种，比如 Bunch-Kaufman 算法：$PAP^T = LBL^T$，其中 P 是排列矩阵，L 是单位下三角阵，B 是分块对角矩阵，且对角块都是常数大小（不超过 2*2）.
- 如果矩阵很大而且稀疏，或者没有显式给出：可以尝试使用某种重排序策略，如果不 work 的话尝试迭代法.
- 如果矩阵的非零带宽很窄：窄到常数的程度就可以直接 GEPP 了. 如果没那么窄，重排序也不太行，同样可能需要回到迭代法.
- 假如矩阵很大，很稀疏而且是 SPD 的：直接上共轭梯度. 假如只对称不正定，可以用类似于 CG 的其他迭代方法，比如 MINRES（最小化误差项的 L2 范数）. 如果连对称都不是，可以使用 GMRES（空间需求较大）或者 BiCGSTAB（空间需求较小，但收敛性可能就很奇怪了）.

5.5 reference

1. Foster L V . Gaussian Elimination with Partial Pivoting Can Fail in Practice[M]. Society for Industrial and Applied Mathematics, 1994.
2. Oguz Kaya, Enver Kayaaslan, Bora Uçar, Iain S. Duff. Fill-in reduction in sparse matrix factorizations using hypergraphs. [Research Report] RR-8448, INRIA. 2014. <hal-00932882>