求解线性系统

Solving Linear System

马龙 赵怡浩 许晟伟 张铎 许靖 程超然

前沿实践II

2019 年 4 月 2 日

Radiosity

问题：
如何为静态场景渲染光照贴图？
例如下图中的有柱子和窗口的房间

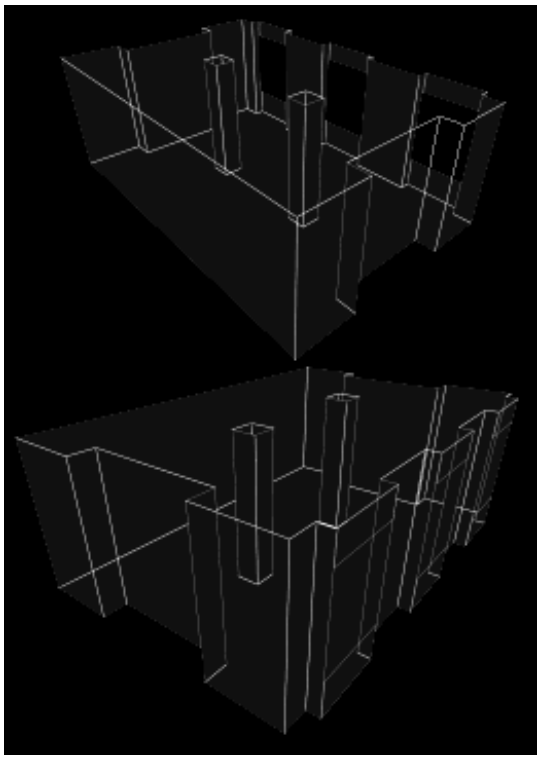


图 1: 房间示意图

Radiosity

直观想法：光线追踪！

图 2: 战地5中实时光线追踪的应用

Radiosity

但是如果你并没有一块最新的RTX2080Ti的话...
我们希望能够找到一些更加“经济”的方法
不妨将房间分割成小贴片，我们来对小贴片进行渲染
最简单的想法是，光照到的小贴片是亮的，其他是暗的

Radiosity

但是如果你并没有一块最新的RTX2080Ti的话...
我们希望能够找到一些更加“经济”的方法
不妨将房间分割成小贴片，我们来对小贴片进行渲染
最简单的想法是，光照到的小贴片是亮的，其他是暗的

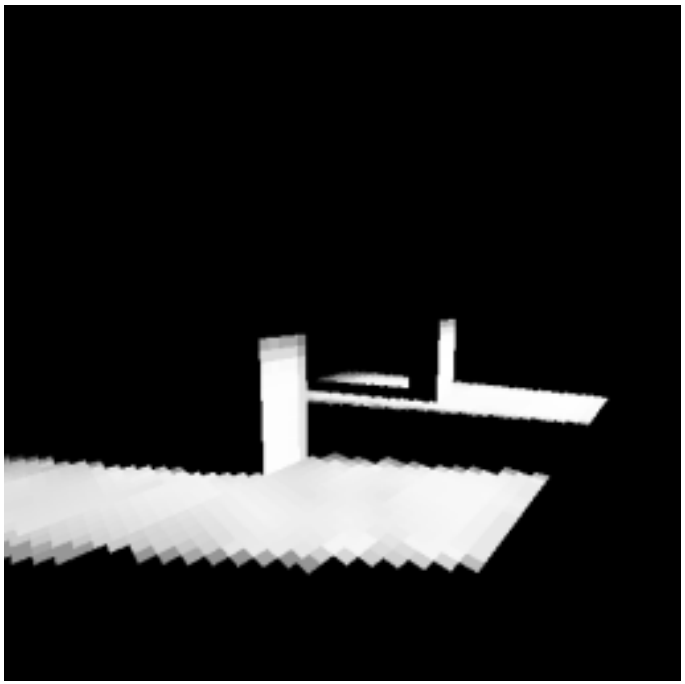
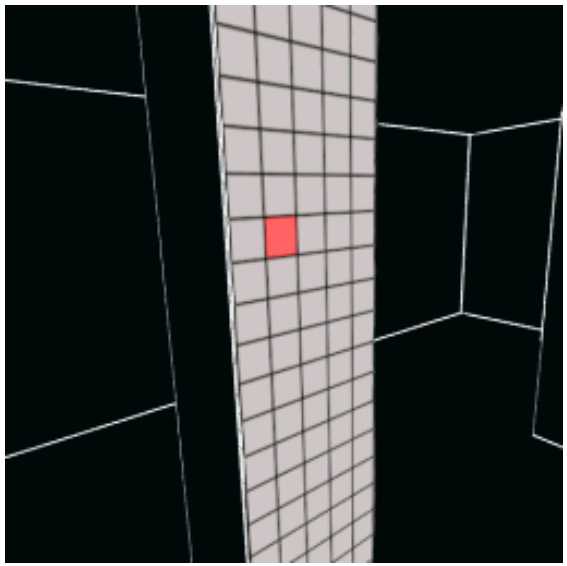


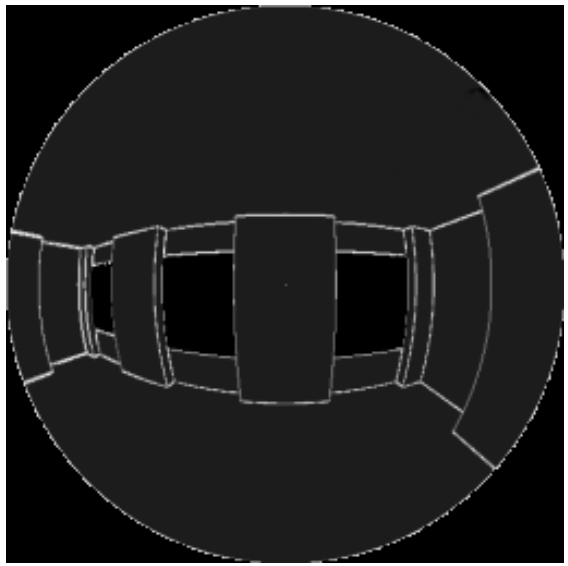
图 3: 效果不太好

Radiosity

问题出在了漫反射上，我们想办法来处理漫反射
不妨先从一个贴片的角度来观察



(a) pic1.



(b) pic2.

图 4: 从贴片的角度

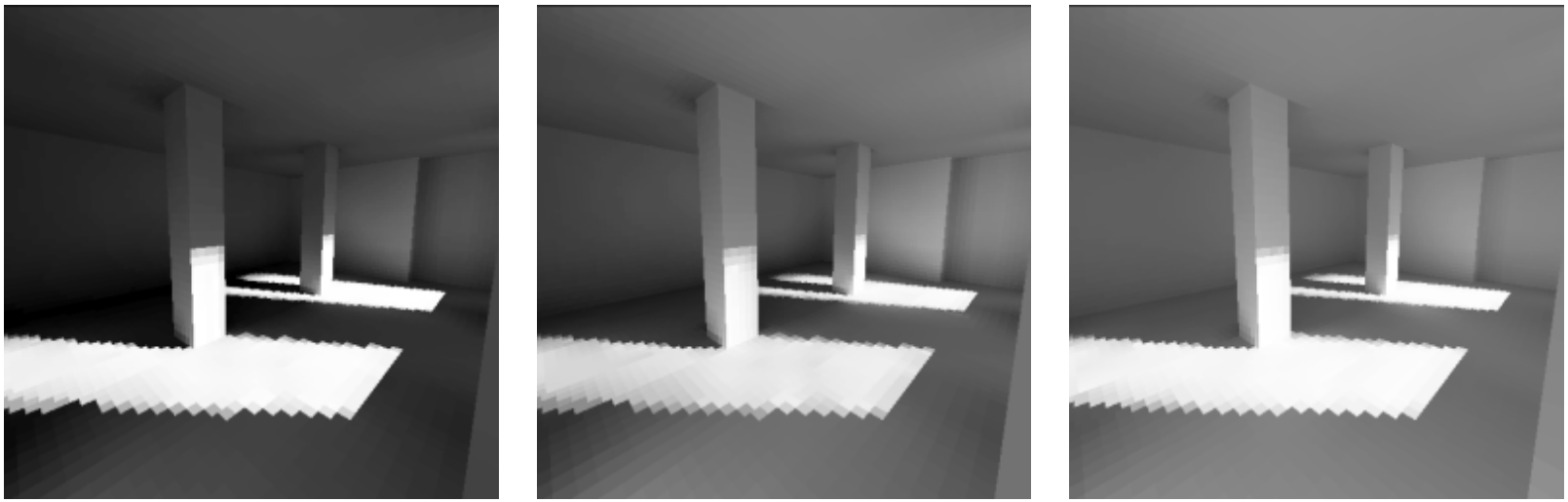
Radiosity

可以简单的认为，我们从贴片的角度看到的其他贴片会和当期贴片直接进行漫反射
因此我们应当提出一种计算漫反射的算法，来解决这个问题
这里我们介绍的是Radiosity（辐射度算法）
一个贴片的辐射度和他的辐射光强（自身发光）、收到的光强和反射率有关
其中收到的光强可以简化为为是其他所有贴片辐射度的线性组合
因此，我们可以用贴片间的一个邻接矩阵来衡量两两贴片间的传递系数

$$B_i = E_i + R_i \sum_{j=1}^{n} B_j F_{ji}$$

Radiosity

可以迭代求解



(a) Itr.2. (b) Itr.4. (c) Itr.16.

也可以求解线性方程组!!!

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，$A \in \mathbb{R}^{n\times n}, \mathbf{x}, \mathbf{b} \in \mathbb{R}^n$. 如果$A$可逆，则这个线性系统对任意b有唯一解：$\mathbf{x} = A^{-1}\mathbf{b}$. 求解线性系统的数值方法通常分为**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). 通常我们从以下方面衡量求解方法的质量：

1. 求解速度；
2. 存储要求；
3. 解的期望准确度等.

矩阵的维数、稀疏性、对称、正定等等特性都是求解中的重要因素.

高斯消元与LU分解

由代数知识可知，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：$U\mathbf{x} = \mathbf{y}$.

高斯消元与LU分解

Algorithm 1 LU 分解，L 存在 A 对角线以下部分（无对角线），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
```

图 5: 高斯消元算法

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

GEPP

若对角线上某个$a_{k,k} = 0$或$\varepsilon \ll 1$，会导致LU分解中$\frac{a_{i,k}}{a_{k,k}}$带来严重误差. 修正LU分解可以采用变换主元(Partial Pivoting)的方法，被称为**GEPP**.
思路是:

- 通过置换将剩下行中当前列数字最大的换到当前行.
- 原问题变为$PA = LU$，P是置换矩阵.

GEPP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.

GEPP

Algorithm 1 LU 分解，L 存在 A 对角线以下部分（无对角线），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
```

图 6: GEPP算法

MATLAB中，运算$x = A\backslash b$时默认使用GEPP，也可以通过[L,U,P]=lu(A)得到A的LU分解.

对称正定矩阵–Cholesky分解

满足$A = A^\top, \forall \mathbf{x} \neq \mathbf{0}, \mathbf{x}^\top A\mathbf{x} > 0$的矩阵$A$称为对称正定矩阵，简称SPD. 对SPD矩阵存在更快(优化常数倍)、更鲁棒，且不要求pivoting的分解Cholesky 分解：

$$A = FF^\top$$

F是下三角矩阵，由对称性可知，对应的上三角矩阵为$F^\top$.
根据代数知识，A能唯一Cholesky分解的充分必要条件是A共轭对称且正定.

Cholesky分解的证明

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$U^\top DL^\top = A^\top = A = LDU$可知$U^\top = L$，同时可以写做$A = LDL^\top = (LD^{1/2})(D^{1/2}L^\top) = FF^\top$，其中$F = LD^{1/2}$是下三角矩阵.
- **必要性**
  $A^\top = (FF^\top)^\top = FF^\top = A$,
  $\forall \mathbf{x} \neq \mathbf{0}, \mathbf{x}^\top A\mathbf{x} = \mathbf{x}^\top FF^\top \mathbf{x} = \|F^\top \mathbf{x}\| > 0.$

□

对称正定矩阵–Cholesky分解

计算可知

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

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
```

图 7: Cholesky算法

稀疏矩阵

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.

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.

高斯消元和稀疏性

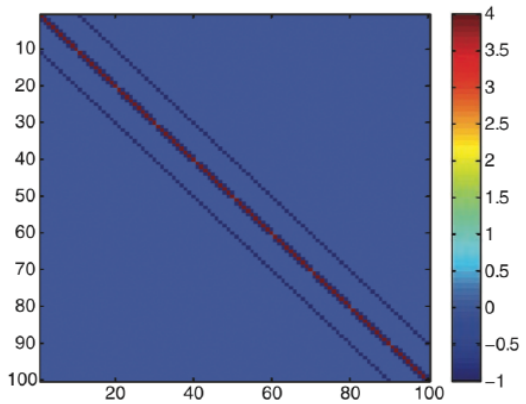


Figure 6.1: Sparsity pattern of a 100 × 100 discrete two-dimensional Laplacian on a uniform mesh.

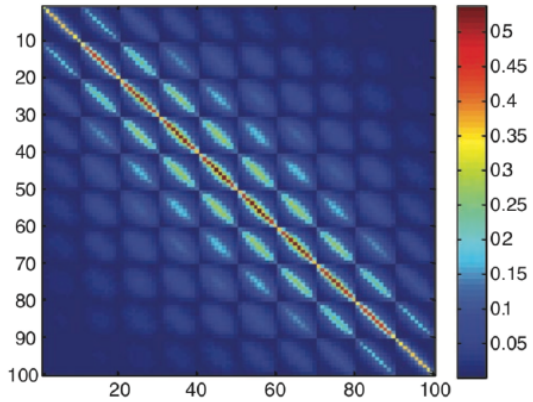


Figure 6.2: An image of the inverse of a 100 × 100 discrete two-dimensional Laplacian on a uniform mesh. The inverse has entries that strongly decay far from the main diagonal, but it is a fully dense matrix.

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)，故称线条带形，且非零元约为$5n$，进行Cholesky分解后有大约$n\sqrt{n}$的非零元，复杂度大约是$O(n^2)$.

高斯消元和稀疏性

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). 因此，分解因子和求逆求解线性方程组的方法是不同的.

迭代法

直接法的局限性：
1)可能引入非零元(如Laplacian)；
2)无法利用初始猜测(如物体追踪时连续时间的物体位置有一定关系)；
3)线性系统不明确可知时(只能通过给出$\mathbf{x}$得到$A\mathbf{x}$，却不知道A).
其中1)可以通过排序策略(ordering strategy)来减少引入的非零元. 对于Cholesky分解，我们考虑寻找稀疏矩阵M，使得$A = M^\top M$且$M^\top M \backslash A$包含极少非零元. 已有的方法大多与A的超图有关.

迭代法

定义

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

矩阵$A(n \times m)$的超图有n个顶点$v_1, ..., v_n$，m条边$\varepsilon_1, ..., \varepsilon_m$，$v_i \in \varepsilon_j$ iff $a_{i,j} \neq 0$

M是A的超图的关联矩阵(incidence graph)时，$M^\top M \backslash A = \emptyset$，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$M^\top M \backslash A = \emptyset$，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$C^\top B \backslash A$的非零元尽量少，也有相应的算法. [2]

Cuthill-McKee算法

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

1: R: n 个节点的新序列
2: 选度数最小的点 x 加入 R
3: **for** i=1,2,...; |R|<n **do**
4: 考虑 R_i 的连通子集 A_i 并排除所有已在 R 中的点 $A_i \leftarrow Adj(R_i) \backslash R$
5: A_i 根据节点度数排升序
6: 把 A_i 接入 R
7: **end for**
8: 返回 R 中点的邻接矩阵 A’

图 8: Cuthill-McKee算法

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. 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. 实现步骤如图.

Cuthill-McKee算法

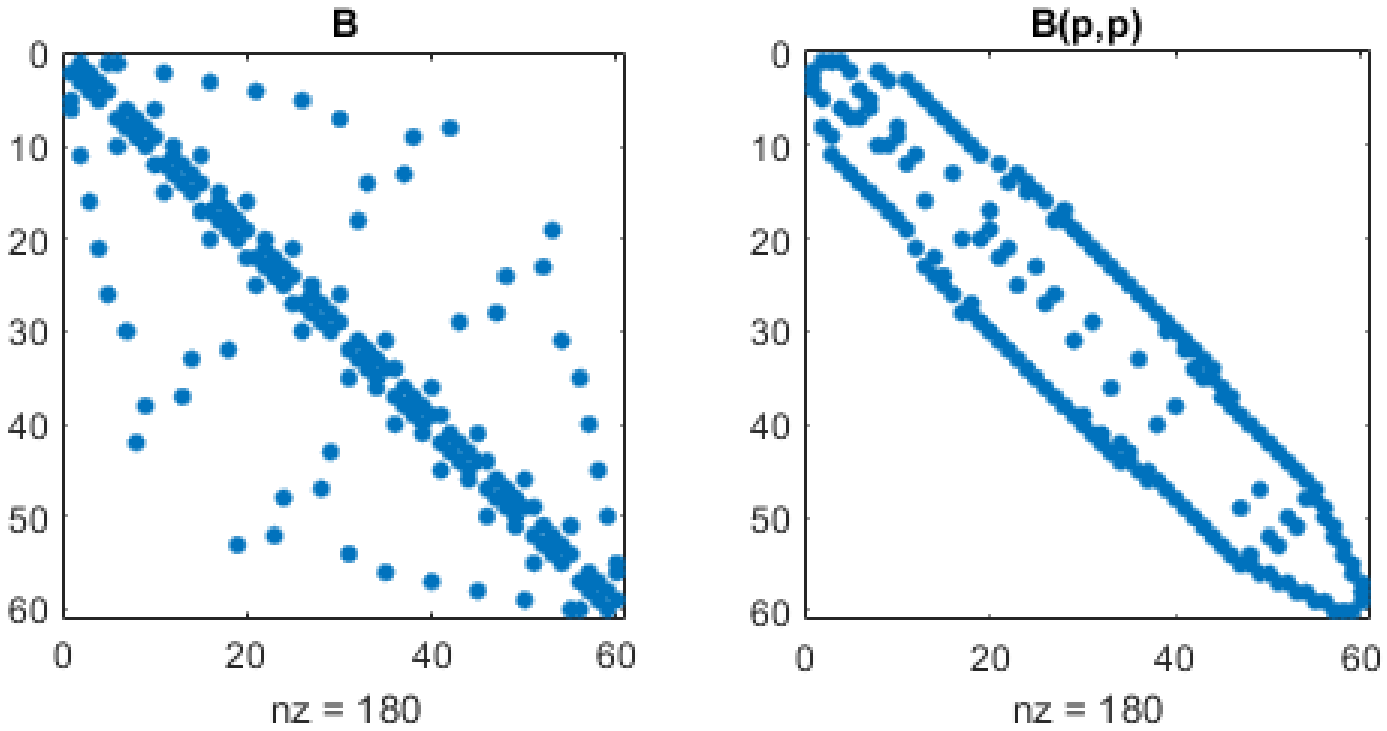


图 9: Cuthill-McKee算法图示

基本静态方法

对于$A\mathbf{x}=\mathbf{b}$，假定$A=M-N$，则有$M\mathbf{x}=N\mathbf{x}+\mathbf{b}$，可得到迭代$M\mathbf{x}_{k+1}=N\mathbf{x}_k+\mathbf{b}$，被称为静态迭代(stationary iteration).
根据M的选取，有两种最基础的迭代方法：

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

Algorithm 4 静态迭代法. Input: M, N, x_0; Output: ans

1: $k \leftarrow 0$
2: **while** $k \leq Iterorerror \geq Error$ **do**
3: $\quad x_{k+1} = -D^{-1}[(E+F)x_k + b]$
4: $\quad k \leftarrow k+1$
5: **end while**

图 10: 静态迭代法

基本静态方法

缺点：收敛速度较慢

优势：1)易于编程；2)对简单问题很高效；

这几种方法的比较：

- Jacobi：可以高度并行化；
- Gauss-Seidel：难以并行化，但常用在多网格方法.
  收敛性与迭代矩阵$T = M^{-1}N = I - M^{-1}A$的特征值有关.

算法的收敛性(1)

引理

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. □

算法的收敛性(2)

推论

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$\lim_{k\to\infty} \left(I + \sum_{j=1}^{k} T^j \right) (I-T) = I$，展开

得$\left(I + \sum_{j=1}^{k} T^j \right) (I-T) = I - T^{k+1} \to I$即可. □

算法的收敛性(3)

定理

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 $\mathbf{x}_0 \in \mathbb{R}^n$，$\lim\limits_{k\to\infty} \mathbf{x}_k$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$\lim\limits_{k\to\infty} \mathbf{x}_k = \mathbf{0} + (I - T)^{-1}\mathbf{c}$收敛，且满足$\mathbf{x} = T\mathbf{x} + \mathbf{c}$. □

所以：
当迭代矩阵$T = M^{-1}N$满足$\rho(T) < 1$时，对于任意的初值，算法都必然收敛.(限原矩阵A为对角优势矩阵)

Krylov子空间

Krylov子空间定义在矩阵A，初值$\mathbf{r}_0$和阶数k上:

$$\mathcal{K}^k(A;\mathbf{r}_0) = \text{span}\left\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \ldots, A^{k-1}\mathbf{r}_0\right\}.$$

思想：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.

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$AV_k = V_{k+1}H_{k+1,k}$.

复杂度$O(k^2 n)$.

Arnoldi迭代

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

1: **for** $j = 1$ to k **do**
2: $\quad \mathbf{z} \leftarrow A\mathbf{v}_j$
3: $\quad$ **for** $i = 1$ to j **do**
4: $\quad\quad h_{i,j} \leftarrow \mathbf{v}_i^T \mathbf{z}$
5: $\quad\quad \mathbf{z} \leftarrow \mathbf{z} - h_{i,j}\mathbf{v}_i$
6: $\quad$ **end for**
7: $\quad h_{j+1,j} \leftarrow ||\mathbf{z}||_2$
8: $\quad$ **if** $h_{j+1,j} = 0$ **then**
9: $\quad\quad$ exit
10: $\quad$ **end if**
11: $\quad \mathbf{v}_{j+1} \leftarrow \mathbf{z}/h_{j+1,j}$
12: **end for**

图 11: Arnoldi迭代

和梯度有关的算法

以下假定$A\mathbf{x}=\mathbf{b}$为待求方程，其中A是SPD实矩阵. 则可将求解原方程转化为**最优化问题**：

$$\phi(\mathbf{x})=\mathbf{x}^{\top}A\mathbf{x}-\mathbf{b}^{\top}\mathbf{x}$$

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.

最速梯度下降(steepest gradient descent)算法

对于当前的解$\mathbf{x}_k$:

沿梯度$\nabla_{\mathbf{x}}\phi(\mathbf{x}_k) = A\mathbf{x}_k - \mathbf{b}$反方向走一步，步长为$\alpha_k$

而步长的选择应当最小化$\phi(\mathbf{x}_k - \alpha_k(A\mathbf{x}_k - \mathbf{b}))$.

最速梯度下降(steepest gradient descent)算法

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

$$\begin{aligned}
\frac{d}{d\alpha_k}\phi(\mathbf{x}_{k+1}) &= \phi'(\mathbf{x}_{k+1})\frac{d}{d\alpha_k}\mathbf{x}_{k+1} \\
&= -\mathbf{r}_{k+1}^\top \mathbf{r}_k \\
&= (A\mathbf{x}_{k+1} - \mathbf{b})^\top \mathbf{r}_k \\
&= [A(\mathbf{x}_k + \alpha_k \mathbf{r}_k) - \mathbf{b}]^\top \mathbf{r}_k \\
&= \alpha_k (A\mathbf{r}_k)^\top \mathbf{r}_k - \mathbf{r}_k^\top \mathbf{r}_k \\
&= 0
\end{aligned}$$

解得$\alpha_k = \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{r}_k^\top A \mathbf{r}_k}$. 此处定义两个向量$\mathbf{u}, \mathbf{v}$在矩阵$A$下的内积是$\langle \mathbf{u}, \mathbf{v} \rangle_A = \mathbf{u}^\top A \mathbf{v}$，上面的式子就可以写成$\alpha_k = \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\langle \mathbf{r}_k, \mathbf{r}_k \rangle_A}$.

最速梯度下降(steepest gradient descent)算法

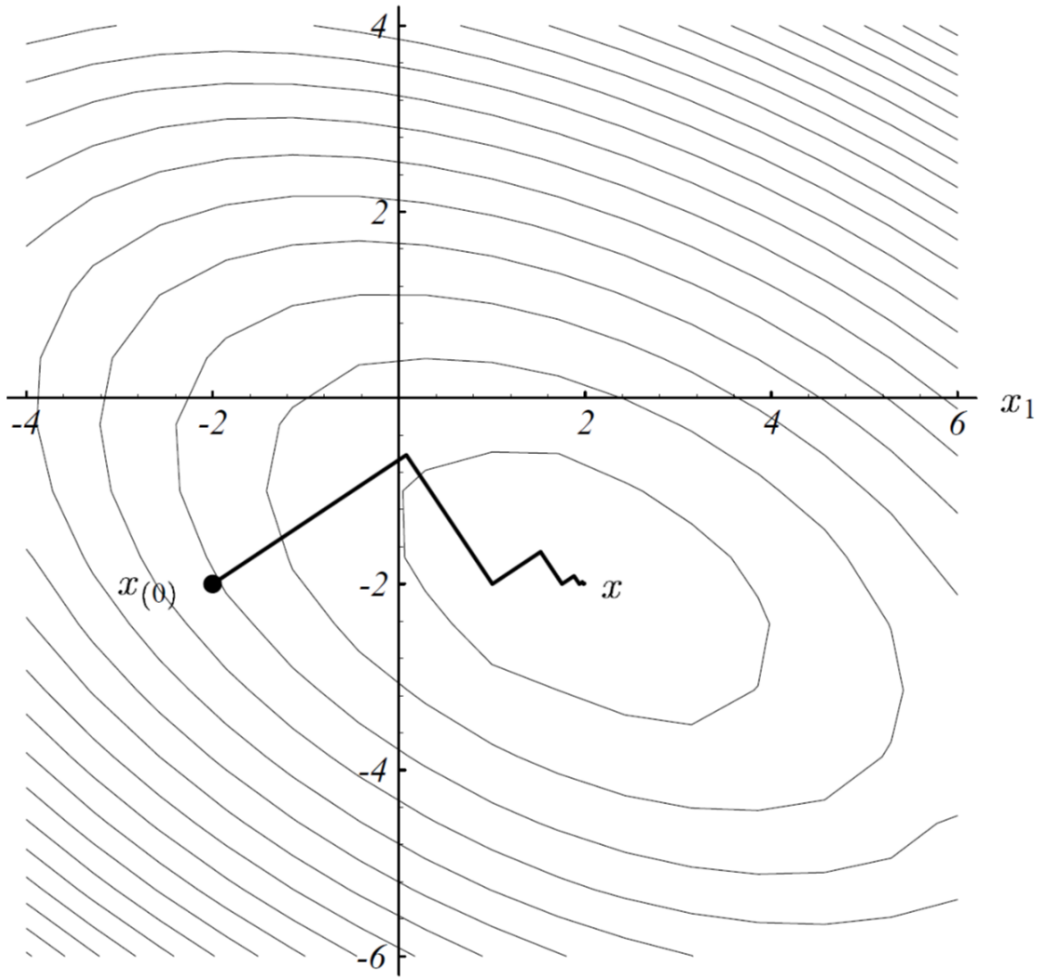


图 12: 梯度下降方向

共轭梯度(conjugate gradient, CG)算法

最速梯度下降中：可能会在不同的时间段向同一个梯度方向走很多步

用共轭方向(Conjugate Directions, CD)的算法来解决这一问题，即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$\langle \mathbf{p}_i, \mathbf{p}_j \rangle_A = 0, \forall i \neq j.$

共轭梯度(conjugate gradient, CG)算法

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$\mathbf{p}_k^\top$,得到

$$\mathbf{p}_k^\top \mathbf{b} = \sum_{i=1}^{n} \alpha_i \mathbf{p}_k^\top A\mathbf{p}_i = \alpha_k \langle \mathbf{p}_k, \mathbf{p}_k \rangle_A, \forall k \in [n],$$

解得$\alpha_k = \frac{\mathbf{p}_k^\top \mathbf{b}}{\langle \mathbf{p}_k, \mathbf{p}_k \rangle_A}$.

即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.

共轭梯度(conjugate gradient, CG)算法

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，每次实际上是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$\|\mathbf{e}_k\|_A = \sqrt{\langle \mathbf{e}_k, \mathbf{e}_k \rangle_A}$

共轭梯度(conjugate gradient, CG)算法

求解完整的基底?? $\Omega(n^2)$

类似于施密特正交化?? $\Theta(n^3)$

因此想要找尽可能“好”的k个向量，在共轭化后表出近似的$\mathbf{x}^*$

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。

共轭梯度(conjugate gradient, CG)算法

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 $\mathbf{x}_k$

1: **for** $j = 1$ to k **do**
2: $\quad \mathbf{s}_{j-1} \leftarrow A\mathbf{p}_{j-1}$
3: $\quad \alpha_{j-1} \leftarrow \frac{\mathbf{r}_{j-1}^T \mathbf{r}_{j-1}}{\mathbf{p}_{j-1}^T \mathbf{s}_{j-1}}$
4: $\quad \mathbf{x}_j \leftarrow \mathbf{x}_{j-1} + \alpha_{j-1}\mathbf{p}_{j-1}$
5: $\quad \mathbf{r}_j \leftarrow \mathbf{r}_{j-1} - \alpha_{j-1}\mathbf{s}_{j-1}$
6: $\quad \beta_{j-1} \leftarrow \frac{-\mathbf{r}_j^T \mathbf{s}_{j-1}}{\mathbf{p}_{j-1}^T \mathbf{s}_{j-1}}$
7: $\quad \mathbf{p}_j \leftarrow \mathbf{r}_j + \beta_{j-1}\mathbf{p}_{j-1}$
8: **end for**

图 13: 共轭梯度算法

非常优秀的$O(km)$复杂度，其中m是矩阵的非零元数量.

收敛性

我们知道

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

收敛性

因此$\mathbf{e}_k = \left(I + \sum_{j=1}^{k} \psi_j A^j\right) \mathbf{e}_0 = P_k(A)\mathbf{e}_0$

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

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

收敛性

即说明：CG的收敛速度被最差的一个特征值的多项式值所限制，故与特征值的分布有关.

对于SPD矩阵的情形，将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.

收敛性

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(left preconditioning). 同时也有right preconditioning:

$$AM^{-1}\mathbf{y} = \mathbf{b}, \mathbf{x} = M^{-1}\mathbf{y}$$

以及split preconditioning:

$$M_1^{-1}AM_2^{-1}\mathbf{y} = M_1^{-1}\mathbf{b}, \mathbf{x} = M_2^{-1}\mathbf{y}.$$

收敛性

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$\hat{L}^\top \hat{L}$；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.

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：$PAP^\top = LBL^\top$，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.

总结

- 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).

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 Uc ar, Iain S. Duff. Fill-in reduction in sparse matrix factorizations using hypergraphs. [Research Report] RR-8448, INRIA. 2014. ¡hal-00932882¿

作业

1. 推导Cholesky分解中矩阵F的具体形式

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

2. 如果实矩阵A的每一列主对角线元素绝对值均大于非主对角线元素绝对值之和，即$|a_{jj}| > \sum_{i=1, i\neq j}^{n} |a_{ij}|, j = 1, 2, \ldots n$，则称之为**严格对角优势矩阵**
证明：A是可逆矩阵

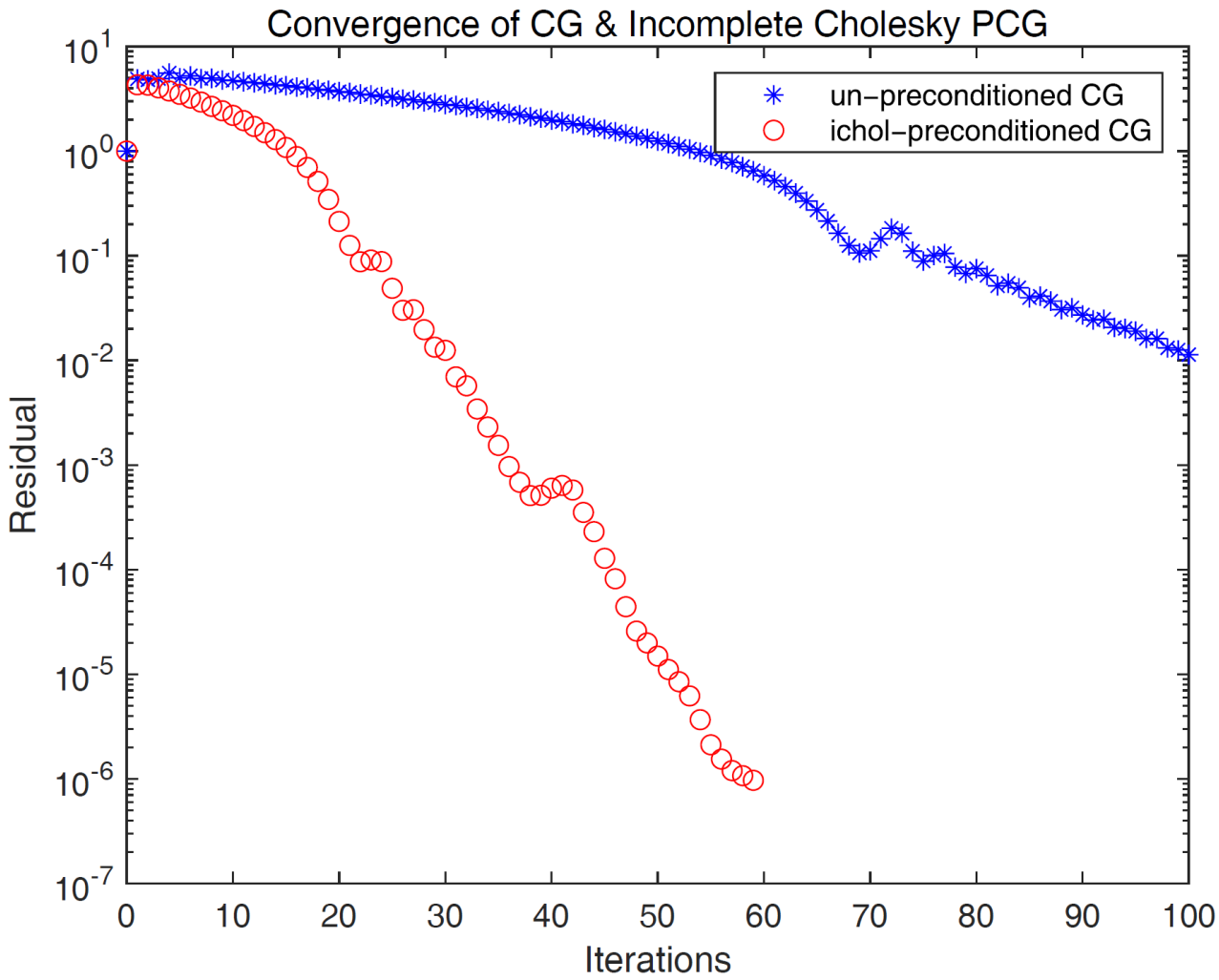
作业

3. 用MATLAB进行稀疏矩阵分解 by ccr

本作业要求实现如教材94页图的效果，即可视化共轭梯度法和非完全Cholesky分解预处理后的共轭梯度法求解线性方程组的过程. 其中稀疏矩阵A数据存储在SparseMatrix.txt文件中(后附)，为一9604×9604的方阵，具有47628个非零元素.
SparseMatrix.txt中的数据排列如下：每组三个数据row, col, v分别代表非零元的行列指标和值，每组由\t分隔. (实际上A是一个五点拉普拉斯微分算子对应的矩阵)
要求解线性方程组$A\mathbf{x}=\mathbf{b}$，其中$\mathbf{b}$是元素全为1的列向量.并绘制两种方法迭代的过程图像. 如下页图是一个示例：

作业

3. 用MATLAB进行稀疏矩阵分解



作业

3. 用MATLAB进行稀疏矩阵分解

提示:

- 代码框架已经在decomposition_prob.mlx中给出(后附).
- 用sparse构建稀疏矩阵(注意之后的pcg, ichol等都只能对稀疏矩阵作用)
- 预处理共轭梯度法pcg用法如下:
  [x, flag, relres, iter, resvec] = pcg(A, b, 1e-6, 100);
  其中输出分别代表线性方程的解，迭代是否收敛(0表示收敛)，相对残差(relative residual)，迭代计数器，历史各项残差的范数；后两个参数分别为收敛界(本例建议设为1e-6)和最大迭代次数(本例建议设为100)

作业

3. 用MATLAB进行稀疏矩阵分解

提示:

- 非完全Cholesky分解预处理ichol用法如下:
  G = ichol(A);
  [x, flag, relres, iter, resvec] = pcg(A, b, 1e-6, 100, G, G');
- 用semilogy绘制残差对于迭代次数的半对数图，注意残差要除以**b**的范数，因为pcg返回值中使用的是相对残差