数值计算实验报告

16340217

王晶

第 1 题：

主要算法是高斯消去法和列主元消去法，其中列主元消去法是在高斯消去法的基础上进行了改进，并且数值稳定性更好，避免了极小元素或零元素做分母。

第（1）小题：通过高斯消去法求解线性方程组 Ax=b，且 A 与 b 中元素满足独立同分布的正态分布。

通过如下语句可以得到矩阵 A 和向量 b：

```
b = normrnd(0, 1, N, 1);
A2 = normrnd(0, 1, N, N);
```

算法内容：主要是将 Ax=b，A 矩阵按照从上到下，从左到右的顺序化为上三角矩阵，最后进行从下到上的回代，可以得到最终的结果。
详细步骤:

设$a_{kk}^{(k)} \neq 0$，令$l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} (i = k+1 \sim n)$

公式为：第i行 $+ (-l_{ik}) \times$ 第k行$(i = k+1 \sim n)$

$$B^{(k)} = \begin{pmatrix} a_{11}^{(k)} & \cdots & a_{1n}^{k} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & a_{nn}^{k} \end{pmatrix}$$

最终可得到一个上三角矩阵

然后从$a_{nn}^{(n)} = b_n^{(n)}$开始向上回代，最终得到 x，公式如下：

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij} b_j}{a_{ii}}$$

数值试验将和列主元消去法一起完成，并对比。

第（2）小题：通过列主元消去法求解线性方程组 Ax=b，且 A 与 b 中元素满足独立同分布的正态分布。

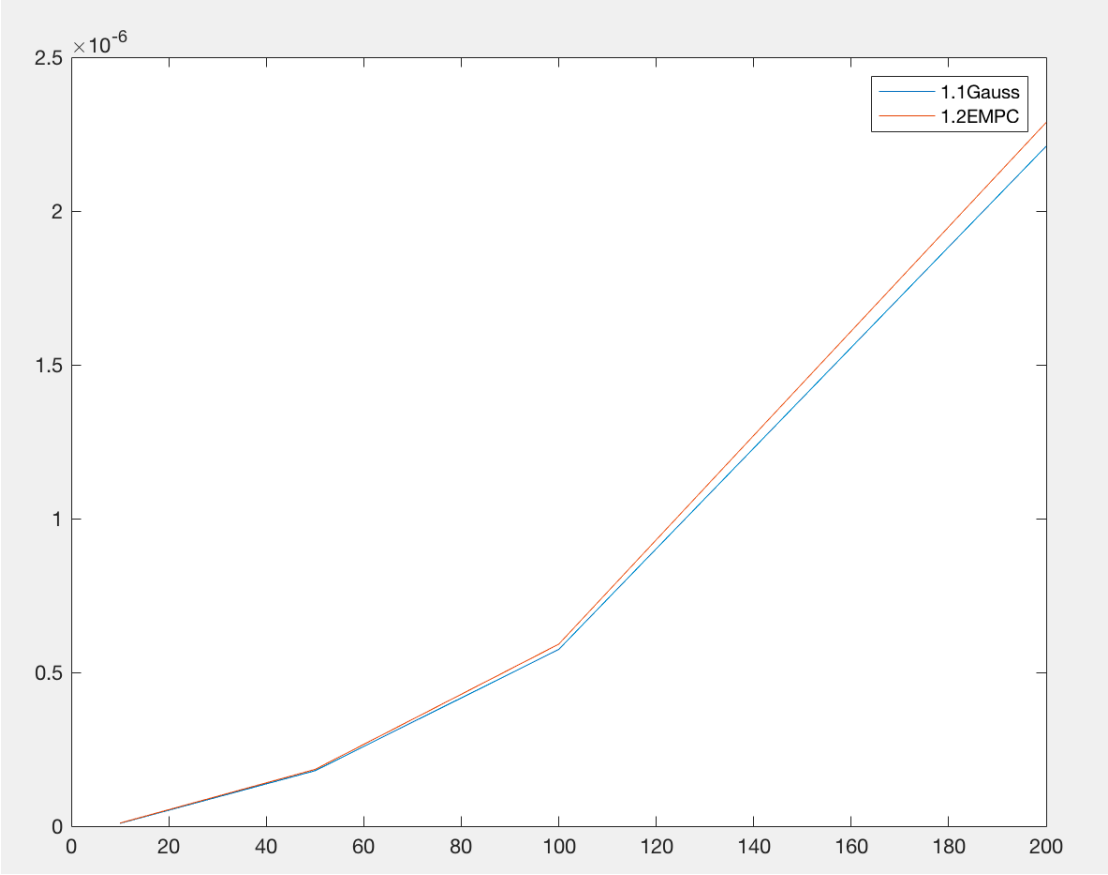
算法内容：行之间的运算近似于高斯消去法，但在第 k 步消元的过程中，要先选取第 k 列第 k 行元素及其以下的各元素中绝对值最大的一个元素，然后通过行变换将它所在的行换到第 k 行，再进行消元。然后 n 步之后，得到一个上三角矩阵，进行回代即可。相比之下，列主元消去法避免了出现极小元素或零元素做分母的情况。

详细步骤：

设$a_{kk}^{(k)} \neq 0$，$\left|a_{kk}^{(k)}\right| = max\left|a_{ik}^{(k)}\right|(k \leq i \leq n)$确定列主元，若是$i \neq k$，就进行行变换，若是相等，则直接进入消元。其余步骤，包括消元，回代，都和高斯消元法相同，即不写出了。

数值试验：

分别记录当 n=10，50，100，200 时所花费的时间，每组 20 次，计算平均值，并将两种方法所花时间放入一张图中显示。



分析：

可以从图中看出，无论是 n=10 还是其他的维数，列主元消去法所花费的时间都比高斯消去法多。大致分析，应该是寻找列主元和进行行变换的时候花费了更多的时间。

第 2 题：

第（1）小题：通过 Jacobi 迭代法求解线性方程组 Ax=b，其中 A 为对称正定矩阵，其特征值服从独立同分布的[0,1]间的均匀分布；b 中元素服从独立同分布的正态分布。那么在 Matlab 中通过如下代码可以求得该矩阵：

```
V = diag(rand(N, 1));
M = orth(rand(N));
A = M*V*M';
```

算法内容：简单理解，就是在一个线性方程组中，通过一个给定的初值，带入其中，得到一组新的值，然后不断的重复这个步骤，最终收敛时得到一个相对精确的根。
详细步骤：

$A=\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}$，$x=\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$，$b=\begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$

设$a_{ii} \neq 0$，令$A=D-L-U$

$D=\begin{pmatrix} a_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & a_{nn} \end{pmatrix}$，$L=\begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ -a_{n1} & \cdots & 0 \end{pmatrix}$，$U=\begin{pmatrix} 0 & \cdots & -a_{1n} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix}$

$(D-L-U)x=b$ 得到 $x=D^{-1}(L+U)x+D^{-1}b$
那么就可以通过这个式子不断地迭代得出根了。

```
D = diag(diag(A));
L = -tril(A,-1);%求
U = -triu(A,1);%求
B = D\(L+U);
f = D\b;
x = B*x0+f;
```

这几句可以看作是算法最主要的内容。

第（2）小题：通过 Gauss-Seidel 迭代法求解线性方程组 Ax=b
相关内容和 Jacobi 迭代相似，但为了加快收敛速度，在迭代代入的过程中进行了调整。

算法内容：在 Jacobi 迭代中，计算完$x_1^{(k+1)}$后，要计算$x_2^{(k+1)}$ ，在这个过程中，并没有用到之前算出的$x_1^{(k+1)}$，而是仍然用着$x_1^{(k)}, \dots, x_n^{(k)}$，所以该算法为了提高收敛速度，进行了改进。

详细步骤：迭代法为：

$$\begin{cases} x^{(0)}, & \text{初始向量} \\ x^{(k+1)} = Bx^{(k)} + f, & k = 0,1,\dots \end{cases}$$

其他步骤依然相似
设$a_{ii} \neq 0$ ， 令$A = D - L - U$
主要区别在于：

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{1}^{i-1} a_{ij}(x_j^{(k+1)}) - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} \right), (i = 1,2,\dots,n, k = 1,2,\dots)$$

在上面，$B = I - (D-L)^{-1}A = (D-L)^{-1}U \equiv G$, $f = (D-L)^{-1}b$
或者：

$X^{(k+1)} = D^{-1}(LX^{(k+1)} + UX^{(k)} + b)$ 求得 $(D-L)X^{(k+1)} = UX^{(k)} + b$

得到 $X^{(k+1)} = (D-L)^{-1}UX^{(k)} + (D-L)^{-1}b$

主要实现语句如下：

```
D = diag(diag(A));
L = -tril(A,-1);%
U = -triu(A,1);%求
G = (D-L)\U;
f = (D-L)\b;
x = G*x0+f;
```

第（3）小题：通过逐次超松弛迭代法求解线性方程组 Ax=b
算法内容：选取分裂矩阵 M 为带参数的下三角矩阵：

$$M = \frac{1}{\omega}(D - \omega L)$$

其中$\omega > 0$是可选择的松弛因子
则迭代矩阵为

$$L_\omega \equiv I - \omega(D - \omega L)^{-1}A = (D - \omega L)^{-1}((1-\omega)D + \omega U)$$

从而解得$Ax = b$
详细步骤：
$Ax = b$的 SOR 方法为：

$$\begin{cases} x^{(0)}, & \text{初始向量} \\ x^{(k+1)} = L_\omega x^{(k)} + f, & k = 0,1,\dots \end{cases}$$

其中$L_\omega \equiv I - \omega(D - \omega L)^{-1}A = (D - \omega L)^{-1}((1-\omega)D + \omega U)$

且$f=\omega(D-\omega L)^{-1}b$

计算公式为：

$$x_i^{(k+1)}=\frac{\omega}{a_{ii}}\left(b_i-\sum_{j=1}^{i-1}a_{ij}x_j^{(k+1)}-\sum_{j=i}^{n}a_{ij}x_j^{(k)}\right),i=1,2\ldots,n,k=0,1,\ldots$$

其中ω为松弛因子，计算过程中一般取（0，2）。

详细语句如下：

```
D=diag(diag(A));      %求A的
L=-tril(A,-1);        %求A的
U=-triu(A,1);         %求A的
B=(D-w*L)\((1-w)*D+w*U);
f=w*((D-w*L)\b);
x=B*x0+f;
```

第（4）小题：通过共轭梯度法法求解线性方程组 Ax=b

算法内容：将共轭性与最速下降法结合，利用已知点处的梯度构造一组共轭方向，并沿这组方向进行搜索，求出目标函数的极小点。

$$\varphi(x)=\frac{1}{2}(Ax,x)-(b,x)$$

令：

$$x^{(k+1)}=x^{(k)}+a_kp^{(k)}$$

使得$\varphi\left(x^{(k+1)}\right)=\min_{a\in R}\varphi\left(x^{(k)}+ap^{(k)}\right)$

且$r^{(0)}=b-Ax^{(0)},p^{(0)}=r^{(0)}$

则$r^{(k+1)}=b-Ax^{(k+1)}=r^{(k)}-a_kAp^{(k)}$

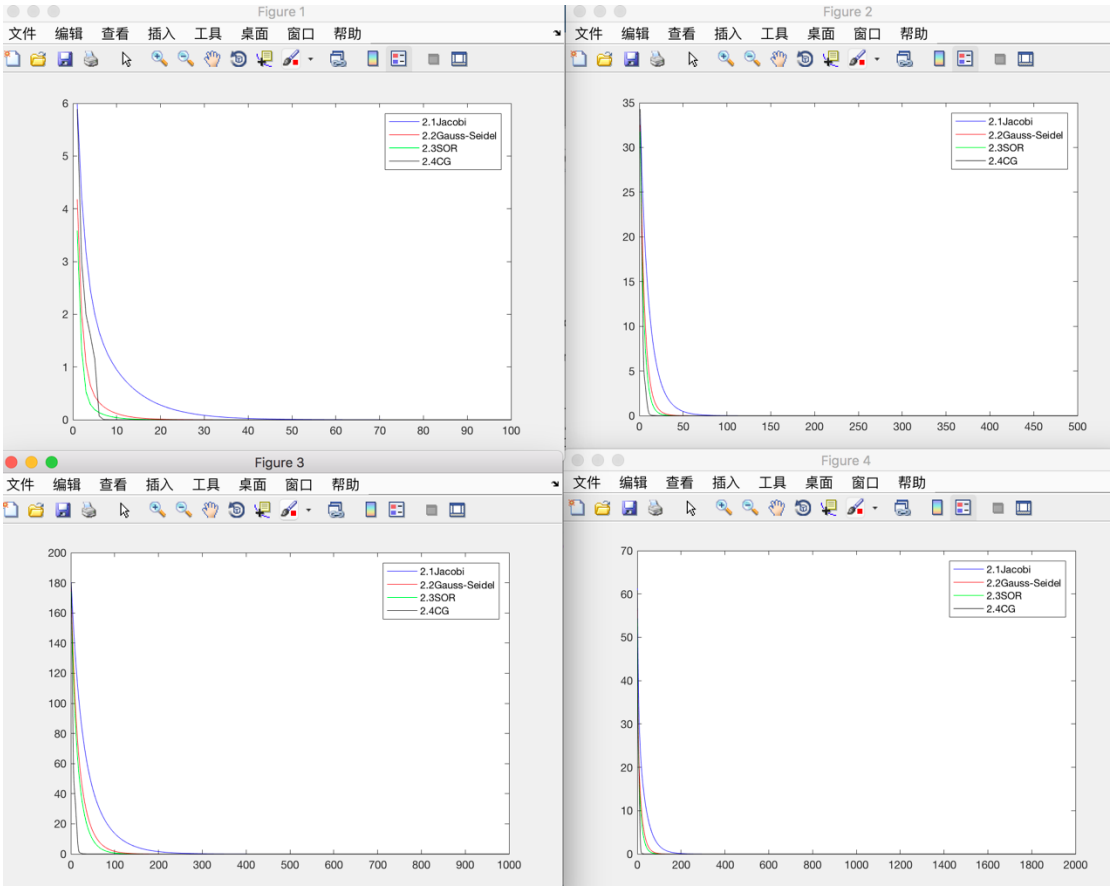
还有$\beta_k=-\frac{(r^{(k)},Ap^{(k-1)})}{(p^{(k-1)},Ap^{(k-1)})}$，以及$a_k=\frac{(r^{(k)},r^{(k)})}{(p^{(k)},Ap^{(k)})}$

然后是$p^{(k+1)}=r^{(k+1)}+\beta_kp^{(k)}$

通过迭代最终得出结果，并在$r^{(k)}=0$,或$\left(p^{(k)},Ap^{(k)}\right)=0$时，停止计算。

数值试验：

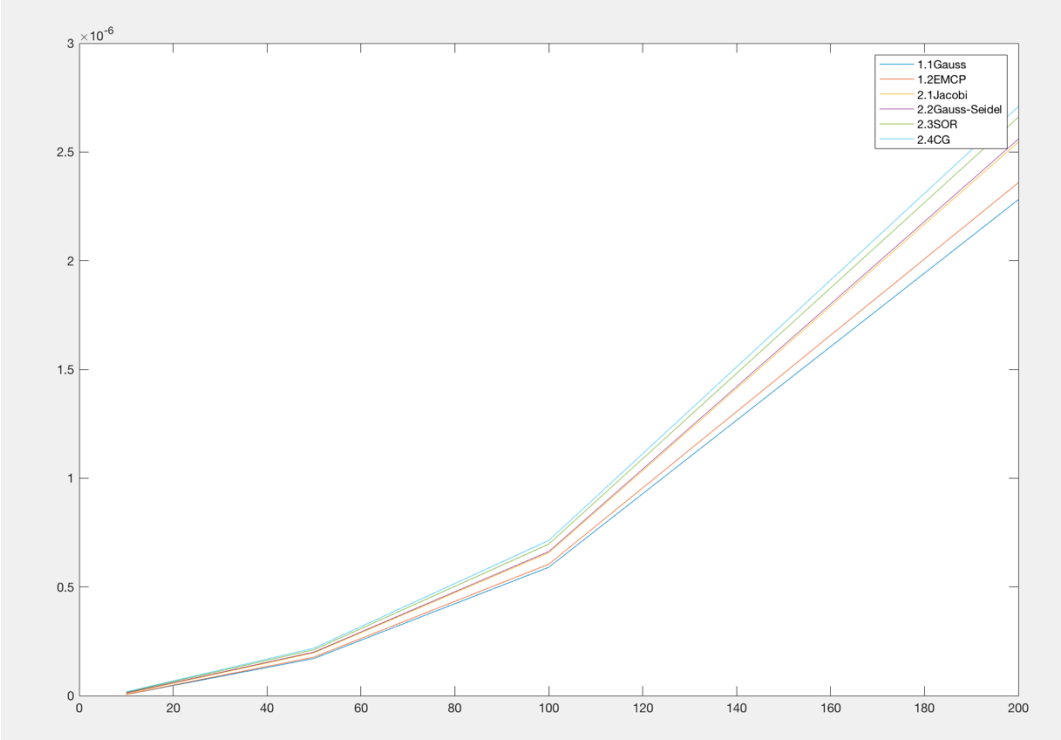
（1）首先是绘制收敛曲线，从左到右，从上到下，figure1-4，分别是 n=10, 50, 100, 200 时的收敛情况。实验中出现过未收敛的情况，未列出。其中，蓝线为 Jacobi 迭代法，红线为 Gauss-Seidel 迭代法，绿线为逐次超松弛迭代法，黑线为共轭梯度法。



分析：

可以明显的发现，在各个维数的情况下，共轭梯度法的收敛速度都是最快的。逐次超松弛迭代法次之，然后是 Gauss-Seidel 迭代法，最后是 Jacobi 迭代法。可以理解为共轭梯度法是在寻找最优的迭代方向，而 Gauss-Seidel 迭代法是逐次超松弛迭代法的一种特殊情况，而 Gauss-Seidel 迭代法又是 Jacobi 迭代法的一种改进，因此这样的结果可以说是合理的。

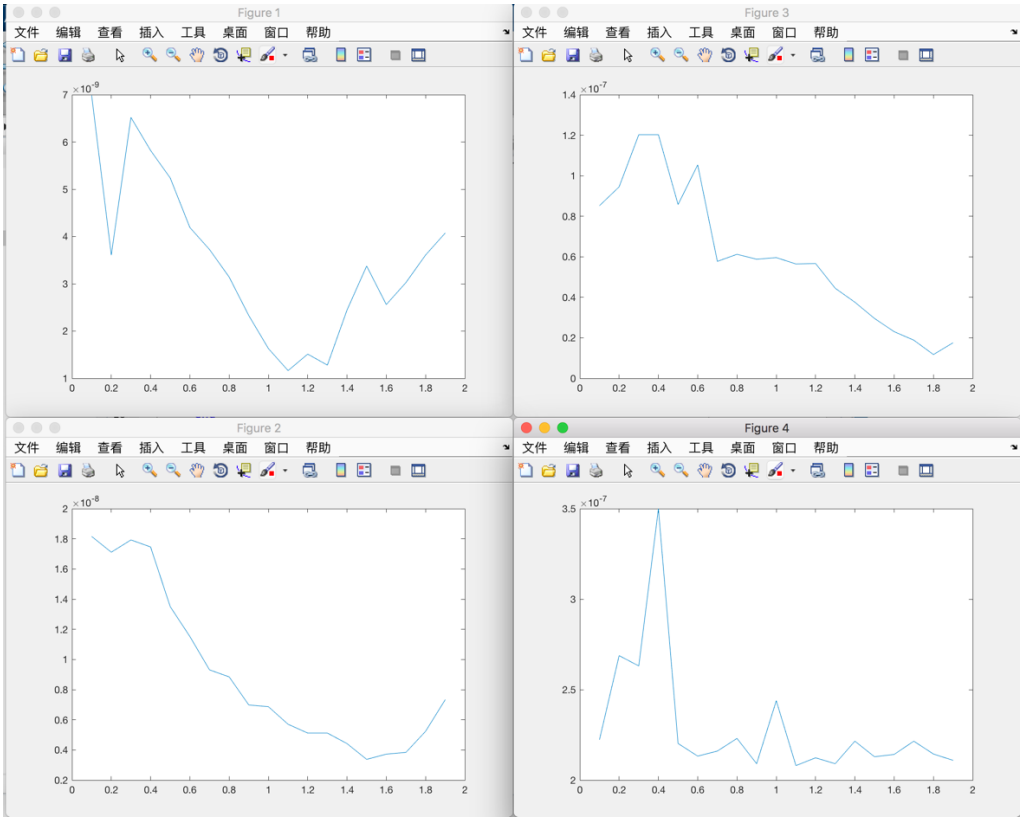
（2）对比六种算法的计算时间。同样是分四种情况，每种循环 20 次计算时间，取平均值。



分析：

可以发现，速度最快的高斯消去法，其次是列主元消去法，因为计算相对简单，而且没有用到矩阵求逆等计算。而随着算法复杂度增加，计算量也增大，时间也增加了。结合之前所测试的，我们可以发现这个顺序正好是收敛速度的倒序，可以理解为牺牲了时间，以求收敛速度，即追求更低的迭代步数。

（3）对比不同松弛因子的情况下，逐次超松弛迭代法的收敛速度。



其中 figure1-4 分别是 n=10， 50， 100， 200 的情况。

分析：

可以发现对于不同的松弛因子，计算所花费的时间是不同的，由于测试样例比较少，可能不适应于所有情况，而在已测试的多个情况中，可以发现当松弛因子ω位于 1.1-1.4 时效果会比较好。

第 3 题：

通过 PageRank 算法来实现对网络节点的受信程度评分。

在 PageRank 算法的每一步中，每个网页的得分都会根据以下公式更新：

$$r = \frac{1-P}{n} + P \times (A^T \times (r./d)) + \frac{s}{n}$$

r 是 PageRank 得分的向量。

P 是标量阻尼因子（通常为 0.85），这是随机浏览者点击当前网页上的链接而不是在另一随机网页上继续点击的概率。

A' 是的邻接矩阵的转置。

d 是包含中每个节点的出度的向量。对于没有外向边的节点，d 设置为 1。

n 是节点的标量数量。

s 是无链接的网页的 PageRank 得分的标量总和。

详细步骤：

基本转移矩阵：$M = L^T \times D^{-1}$ ，L 为邻接矩阵，D 为对角矩阵，其中 D(k,k)表示第 k 个结点的出度（出链）。

处理悬挂网页（没有出链的网页）的随机性修正：$S = M + e \times \frac{a^T}{N}$ ，其中 e 为所有分量均为 1 的列向量，N 为网页总数，a为描述“悬挂网页”的行向量，其第 i 个分量的取值由第 i 个网站是否为“悬挂网站”决定，若是则为 1，否则为 0。

最终得到

$$G = q \times S + (1 - q)e \times \frac{e^T}{N}$$

其中 q 为阻尼系数

最终通过幂迭代法得到了各个网页的 PR 值。

（数据量太大，MATLAB 无法运行，只能直接解释算法，和展示伪代码）

伪代码如下：

首先得到表示节点之间关系的矩阵 L;

设置阻尼系数 q=0.85；获取节点总数 N=size(L, 1);

d=sum(L, 2)获取各个节点出度，D=diag(d)化为对角矩阵;

$M = L^T \times D^{-1}$获取初始转移矩阵;

创建向量 e=ones(N, 1)，得到向量 a=(d==0);

处理悬挂网页得到$S = M + e \times \frac{a^T}{N}$;

然后通过算法计算公式$G = q \times S + (1 - q) \times \frac{e \times e^T}{N}$得到迭代矩阵;

理解为求解$\lim_{n \to \infty} G^n X$的值;

设置初始向量X=ones(N, 1)，即每个网页的 PR 值均，一般为 1;

向量XT=zeros(N, 1)用于存放上一次迭代的X;

（然后迭代得到各节点的 PR 值，即最终X，当 norm($X^{(k)} - X^{(k-1)}$)≤eps 时停止迭代）

进行循环：当$norm(X - XT) \leq eps$时结束

{

$XT = X;$

$X = G \times XT;$

}

通过 sort(XT)进行排序;

输出;