数值分析实验报告

（实验一）

姓名：陈明亮

学号：16340023

班级号：06

上课时间： 周一 9-10 节，

周四 9-10 节

【一】问题重述

作业总体条件： 实现下述算法，求解线性方程组 Ax=b，其中我们要求 A 为 n x n 维的已知矩阵，b 是 n 维的已知向量，x 为 n 维的未知向量。

问题 1：编写程序实现高斯消元法，以及列主元消去法，其中此问题要求 A 与 b 中的元素服从独立同分布的正态分布。令 n=10、50、100、200，测试计算时间并绘制曲线。

问题 2：编写程序实现 Jacobi 迭代法，Gauss-Seidel 迭代法，逐次超松弛 SOR 迭代法以及共轭梯度 CG 法。此问题要求 A 为对称正定矩阵，并且其特征值服从独立同分布的[0,1]间的均匀分布；b 中的元素服从独立同分布的正态分布。令 n=10、50、100、200，分别绘制出算法的收敛曲线，横坐标为迭代步数，纵坐标为相对误差。同时，比较 Jacobi 迭代法，Gauss-Seidel 迭代法，逐次超松弛迭代法，共轭梯度法与上问题的高斯消去法，列主元消去法的计算时间。改变逐次超松弛迭代法的松弛因子，分析其对收敛速度的影响。

问题 3：编写程序实现 PageRank 算法，对 Epinions 社交数据集的网络节点进行信任评分，然后对网络节点的信任程度进行评分，最后给出 PageRank 算法的伪代码。

【二】算法设计

一、高斯消元法与列主元消去法设计

1. 高斯消元法的 Matlab 程序实现

首先，高斯消元法解线性方程组 Ax=b 的过程分为两大步骤：消元计算和回代计算。

第一步消元计算的核心在于逐步计算 $A^{(k)}$矩阵，并利用系数矩阵 M 中元素 $m_{ik} = A_{ik}^{k} / A_{kk}^{k}$，逐步消去矩阵 A 中各行的主元项之前的非主元项，同时也将 b 矩阵更新，完成消元计算。下面贴出消元计算的 Matlab 代码。

```
%% 消元计算 %%
for k=1:n-1
    for i=k+1:n
        m(i, k) = AN(i, k, k)/AN(k, k, k);
    end
    for t=1:k
        AN(t,:,k+1) = AN(t,:,k);
        bN(t, k+1) = bN(t, k);
    end
    for i=k+1:n
        for j=k+1:n
            AN(i, j, k+1) = AN(i, j, k) - m(i, k)*AN(k, j, k);
        end
        bN(i, k+1) = bN(i, k) - m(i, k)*bN(k, k);
    end
end
```

代码解释：上述代码中，AN 为三维矩阵，代表着问题中的 A 矩阵，并且该 AN 矩阵的最后一个维度代表着迭代次数。bN 为二维矩阵，代表着问题中的 b 矩阵，n 为矩阵们的维度。代码中利用生成的 m_{ik} 逐渐消去，并辅助更新 bN 矩阵元素。

第二步的回代计算，通过 $x_n = b_n^n / A_{nn}^n$ ，计算出结果 x 的第 n 个元素，然后通过迭代循环逐步的得到 x 中的每一项，巧妙地利用了

$x_i = (b_{ii} - \sum_{j=i+1}^{n} x_j A_{ij}^i) / A_{ii}^i$ 计算公式。

```
%% 回代计算 %%
x(n) = bN(n, n)/AN(n ,n, n);
for i=n-1:-1:1
    sum = 0;
    for j=i+1:n
     sum = sum + AN(i, j, i)*x(j);
    end
x(i) = (bN(i, i) - sum) / AN(i, i, i);
end
```

2. 列主元消去法的 Matlab 程序实现

列主元消去法的算法实现与高斯消去法的核心很相似，但是实际上相对于前面的算法多了一个步骤，具体可以分为以下三个部分：交换矩阵行，消元计算以及回代计算。

交换矩阵各行其实相当于进行一次初等的行变换，其中我们逐行检测矩阵 A 的主元元素，当检测到第 i 行的主元元素的绝对值大于第 k 次迭代的第 k 行的主元时，我们交换此两行，同时也对 b 矩阵中的这两行进行此操作。

```
if ik ~= k
    for j=k:n
     tt = A(k, j);
     A(k, j) = A(ik, j);
     A(ik, j) = tt;
    end
    btemp = b(ik);
    b(ik) = b(k);
    b(k) = btemp;
end
```

其中 ik 是检测到的主元绝对值大于当前行的行序号，可以看到对 b 矩阵也

进行了交换。

消元计算和回代计算实际上是完全参考高斯消去法而生成的，此处结合列主元消去法的特点，于是将矩阵简化，加上对 M 矩阵的启用，我们有了较为精简的消元计算以及回代计算的代码。

```
%% 消元计算 %%
for i=k+1:n
A(i, k) = A(i, k) / A(k, k);
b(i) = b(i) - A(i, k)*b(k);
for j=k+1:n
    A(i, j) = A(i, j) - A(i, k)*A(k, j);
end
end
end
%% 回代计算 %%
b(n) = b(n)/A(n, n);
for i=n-1:-1:1
sum = 0;
for j=i+1:n
    sum = sum + A(i, j)*b(j);
end
b(i) = (b(i) - sum) / A(i, i);
end
```

3. 元素独立同分布服从正态分布的矩阵 A，b 生成

由于使用 Matlab 语言书写实现程序 此些矩阵的生成均运用了 randn 函数，输入对应的维度，生成服从正态分布的矩阵。

二、Jacobi 迭代法，Gauss-Seidel 迭代法，SOR 迭代法以及 CG 迭代法设计

1. Jacobi 迭代法的 Matlab 程序实现

迭代法的优劣判断标准为迭代到精确解的总步数和计算时间。雅克比迭代法是比较基础的迭代法，实现只需要逐行的迭代更新结果矩阵 x 的元素数值，利用

$x_{i,k+1} = (b_i - \sum_{j!=i}^{n} x_{j,k} A_{i,j}) / A_{i,i}$ 公式，逐步接近近似解。同时，为确定迭代步数的大致范围，我们设置最大迭代步数为 1000 步，检测每一步迭代完之后是否接近精确解的可接受范围之内，是则退出，返回迭代步数，反之继续迭代。

```
for k=1:1000
for i=1:n
    sum = 0;
    for j=1:n
        if j~=i
            sum = sum + A(i, j)*x(j, k);
        end
    end
    x(i, k+1) = (b(i) - sum) / A(i, i);
end
end
```

2. Gauss-Seidel 迭代法的 Matlab 程序实现

高斯赛德尔迭代法与雅克比迭代法在本质上十分相似，唯一不同的地方是，前者使用了每次迭代后更新的 x 矩阵的各项值，而后者则一直在使用初始值。这就十分直接地导致了高斯赛德尔迭代法的收敛速度要比雅克比迭代法快不少，更加快速地达到精确解。

```
for k=1:1000
for i=1:n
    sum1 = 0;
    sum2 = 0;
    for j=1:i-1
        sum1 = sum1 + A(i, j)*x(j, k+1);
    end
    for j=i:n
        sum2 = sum2 + A(i, j)*x(j, k);
    end
    x(i, k+1) = x(i, k) + (b(i)-sum1-sum2) / A(i, i);
end
end
```

上述代码中的 sum1 为迭代更新后的 x 各项数值与 A 矩阵乘积之和，这部分

直接地决定了新生成的 x 元素更加与精确解接近。

3. SOR 迭代法的 Matlab 程序实现

SOR 超松弛迭代法是高斯赛德尔迭代法的一个引申，该迭代法与后者的差别就在于一个松弛因子 w，而通过改变该松弛因子的数值，我们可以逐渐地拿到符合最快收敛速度的最佳因子，从而超越高斯赛德尔迭代法的速度。w 用于与目标矩阵 b 与前几项 A 与 x 元素乘积的总和 sum 的乘积中，逐渐改变与精确解的接近程度。

```
for k=1:1000
    for i=1:n
     sum1 = 0;
     sum2 = 0;
     for j=1:i-1
         sum1 = sum1 + A(i, j)*x(j ,k+1);
     end
     for j=i:n
         sum2 = sum2 + A(i, j)*x(j ,k);
     end
     x(i, k+1) = x(i, k) + w*(b(i)-sum1-sum2) / A(i ,i);
    end
end
```

4. CG 迭代法

CG 共轭梯度迭代法的核心就在于，确定一个最快速接近精确解的收敛方向，并逐渐在这个梯度上进行一个收敛的逼近，最终较优的解。其中，我们使用 r 和 p 向量，其中 r 向量记录 b 与 Ax 之间的差值矩阵，并随着迭代逐步更新，p 辅助确定最速逼近方向，逐步利用转置乘法更新结果矩阵 x。

```
for k=1:1000
    if isequal(r0, zeros(n, n)) || isequal(p0'*A*p0, zeros(n, n))
     break;
```

```
    end
    ak = (r0'*r0)/(p0'*A*p0);
    x1 = x0 + ak*p0;
    r1 = r0 - ak*A*p0;
    bk = (r1'*r1)/(r0'*r0);
    p1 = r1 + bk*p0;

    %% 将各项 k+1 的计算结果存回，避免存储多余的项 %%
    x0 = x1;
    r0 = r1;
    p0 = p1;
end
```

5. 生成符合条件矩阵的方法

首先，为了让雅克比矩阵成功收敛，我们需要将 A 以及 2D-A 矩阵皆生成为对称正定矩阵，并且 A 特征值符合[0,1]的均匀分布，b 矩阵符合正态分布。

```
for i=1:20
    b = normrnd(0, 1, n, 1);
    V = diag(unifrnd(0, 1, 1, n));
    U = orth(normrnd(0, 1, n, n));
    A = U' * V * U;
    D = tril(A) + triu(A) - A;

    c = eig(2*D - A);
    min = c(1);
    for j=2:n
        if min > c(j)
            min = c(j);
        end
    end
    if min > 0
        break;
    end
end
```

此处，我们将生成过程定义在一个长度为 20 的循环内，首先生成正态矩阵 b，然后利用 diag 函数生成正态对角阵 V，正交正态矩阵 U，利用对称正定矩阵的定义，我们使用$U^{'}*V*U$ 生成 A，然后拿到对角矩阵 D，检测 2D-A 的正

定性（对称性与 A 相同），符合则输出，不符合重新生成。

三、PageRank 算法的 Matlab 程序实现

1. PageRank 的算法设计

PageRank 算法是实现评价网页中各节点的重要度的有效算法，也是由谷歌创始人设计并提出的。首先，我们使用每个节点-边矩阵，将每个节点对应每个节点的链接矩阵记录下来，其次则是生成概率矩阵 P，表示从网页到另外一个网页的跳转概率。最终，我们使用幂法迭代求解谷歌矩阵的最终结果，拿到对应的网页重要度排序。其中，我们确定阻尼系数为 a=0.85，初始谷歌矩阵的生成算式为 $G=a*H^{'}+J*((1-a)/size)$，设置初始重要度向量各项元素的初始值为 1，进行 R=G*x 的逐次迭代，直到重要度向量与上次迭代的结果值之间的误差不超过误差范围 0.0001，最终得到结果。

```
for i=1:m
    disp([num2str(i), 'Sp']);
    link(point(i, 1)+1) = link(point(i, 1)+1) + 1;
end

for i=1:m
    disp([num2str(i), 'H']);
    if link(point(i, 1)+1) ~= 0
        H(point(i, 1)+1, point(i, 2)+1) = 1 / link(point(i, 1)+1);
    end
end
%%幂法迭代过程%%
G = a*H' + J*((1-a)/size0);
x = ones(size0, 1)/size0;
R = zeros(size0, 1);
while norm(x - R) > 0.001
    R = x;
    x = G*x;
end
```

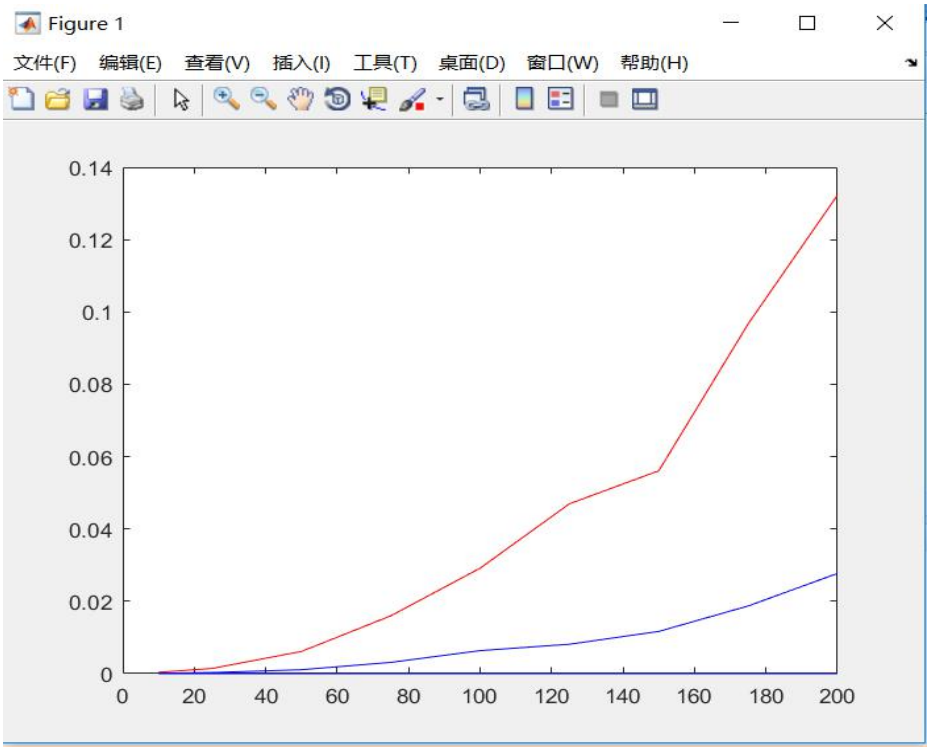
2. PageRank 算法的伪代码

```
输入边矩阵 line, 大小为 size X 2
输入全一矩阵 J,  大小为 size X size

for i=1->m
    link(point(i ,1))+1
endfor
for i=1->m
    if link(point(i, 1)) 不等于 0
        H(point(i,1), point(i, 2)) <= 1 / link(point(i, 1))
    endif
endfor
G = a*H 的转置 + J*((1-a)/size)
R = ones(size);
while R 不接近于前一次迭代的 R
    R = G*x;

得出结果 R
```

【三】数值实验与结果分析

一、对问题一的数值实验与分析

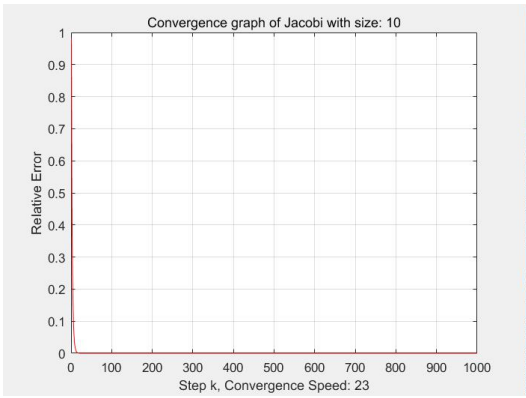
首先，我们对于高斯消元法以及列主元消去法，进行题目要求的数值实验：令 n 为不同的数字，测试计算时间并绘制曲线。

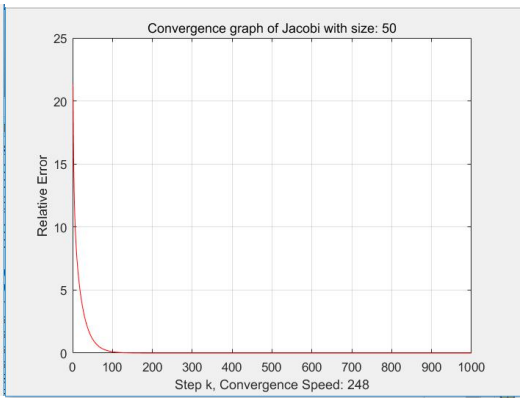


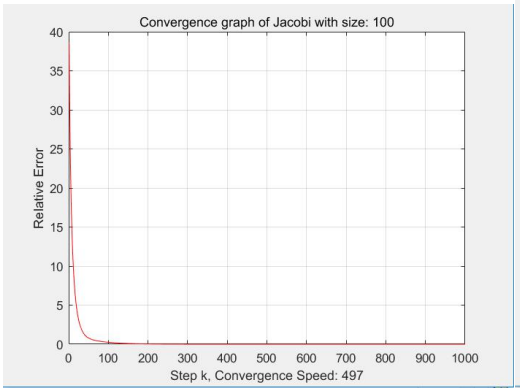
从结果可以看出，高斯消元法（红色线条）比列主元消去法（蓝色线条）的执行时间和速度都要慢，测试的矩阵维度为 20 到 200，并且随着矩阵维度的增大，高斯消元法与列主元消去法之间的差距在逐渐增大。

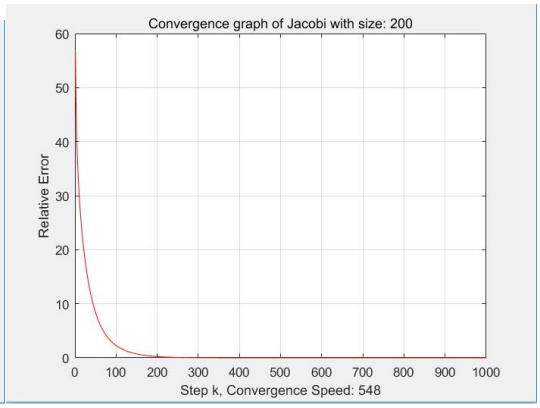
二、问题二各迭代法的数值实验和分析

首先，我们对 Jacobi 迭代法进行数值实验，输入矩阵维度 n，绘制迭代步数的变化曲线。



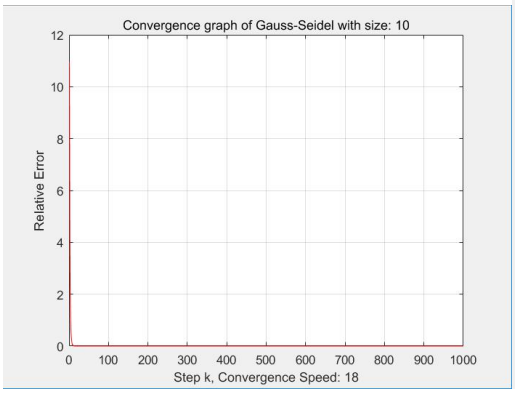


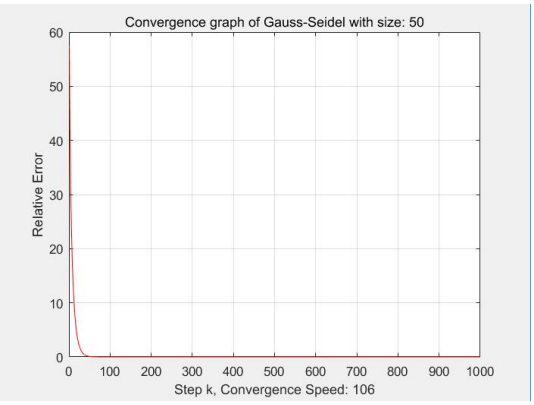


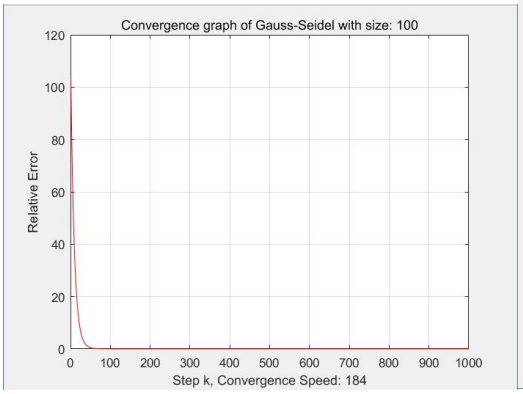


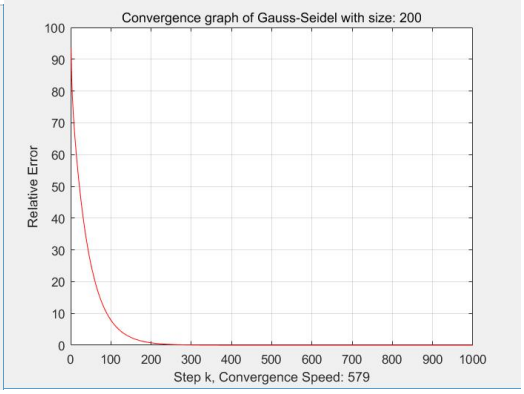
图中横坐标为迭代步数，最下面一行署名了最终收敛的最小步数，纵坐标为相对误差，曲线到达横轴意味着误差与 0 十分相近。Title 栏声明了矩阵的维度。

接下来是 Gauss-Seidel 迭代法的各个 size 迭代步数曲线图。



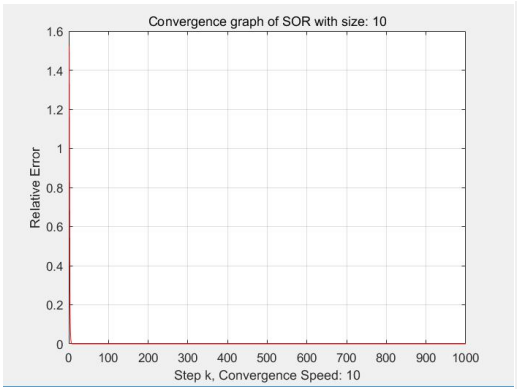


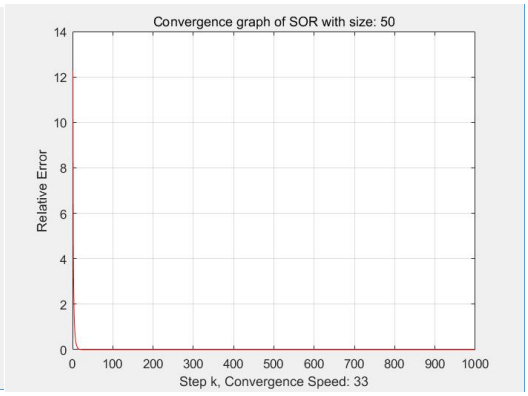


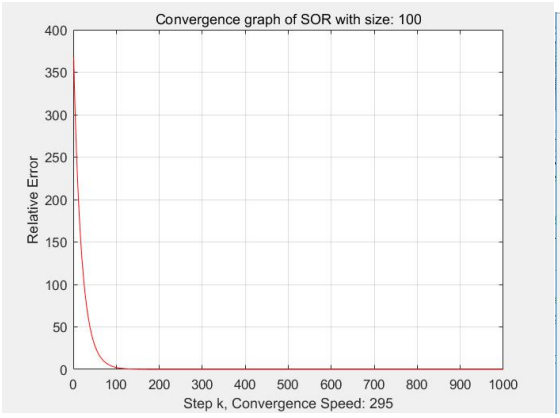


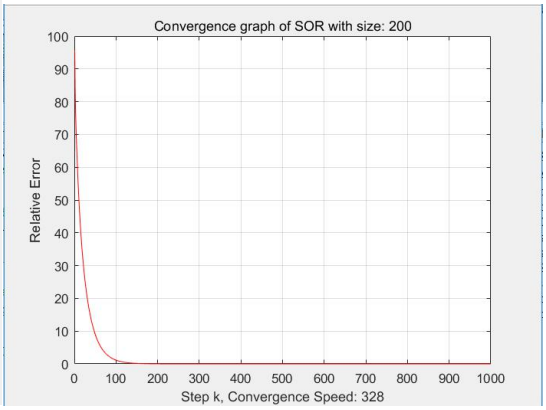
可以明显地看到，雅克比迭代法在总体上比高斯赛德尔迭代法收敛速度慢。

接下来是 SOR 迭代法，此处取松弛因子 w=1.3.



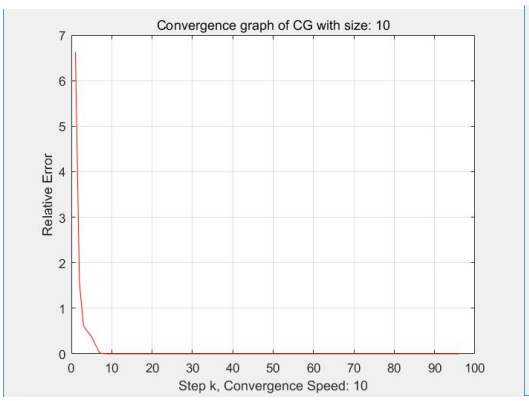


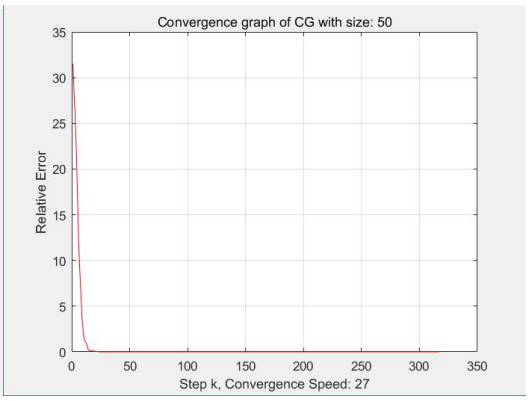


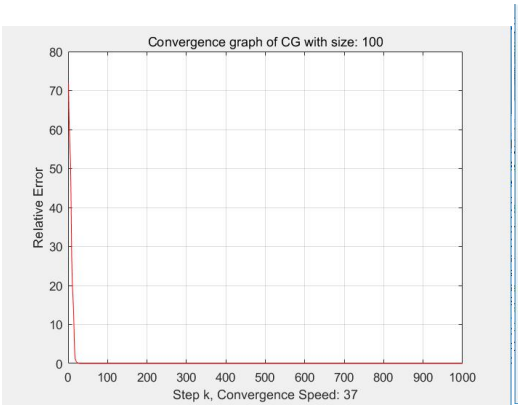


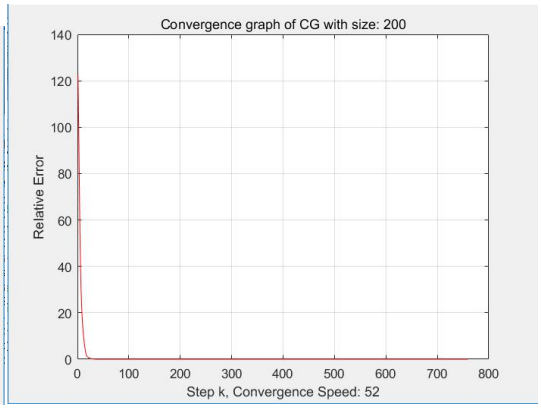
SOR 迭代法的收敛步数与高斯赛德尔迭代法总体上相似。

接下来是 CG 迭代法。



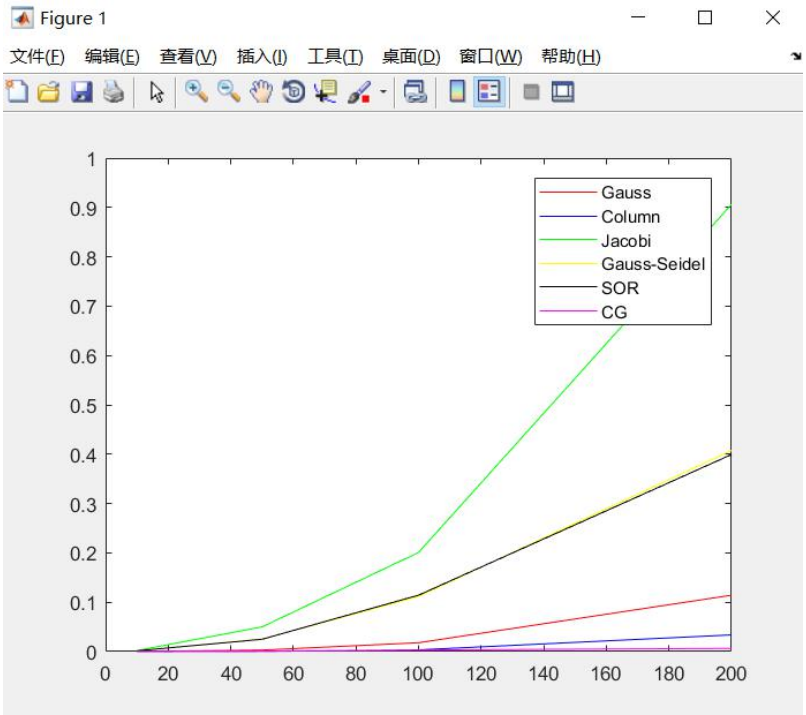






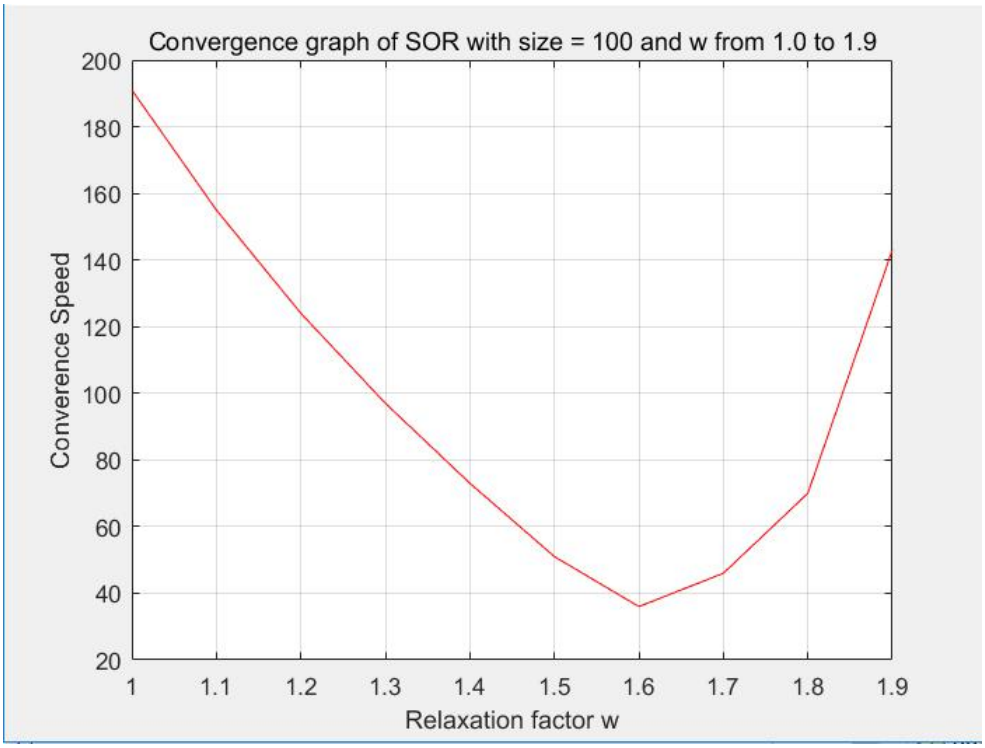
总体来看，CG 迭代法是四种迭代法中最快收敛到精确解的迭代算法，不论是矩阵维度多大，总体上的迭代步数都不会超过 100 步。

三、四种迭代法与高斯，列主元消去法的计算时间比较



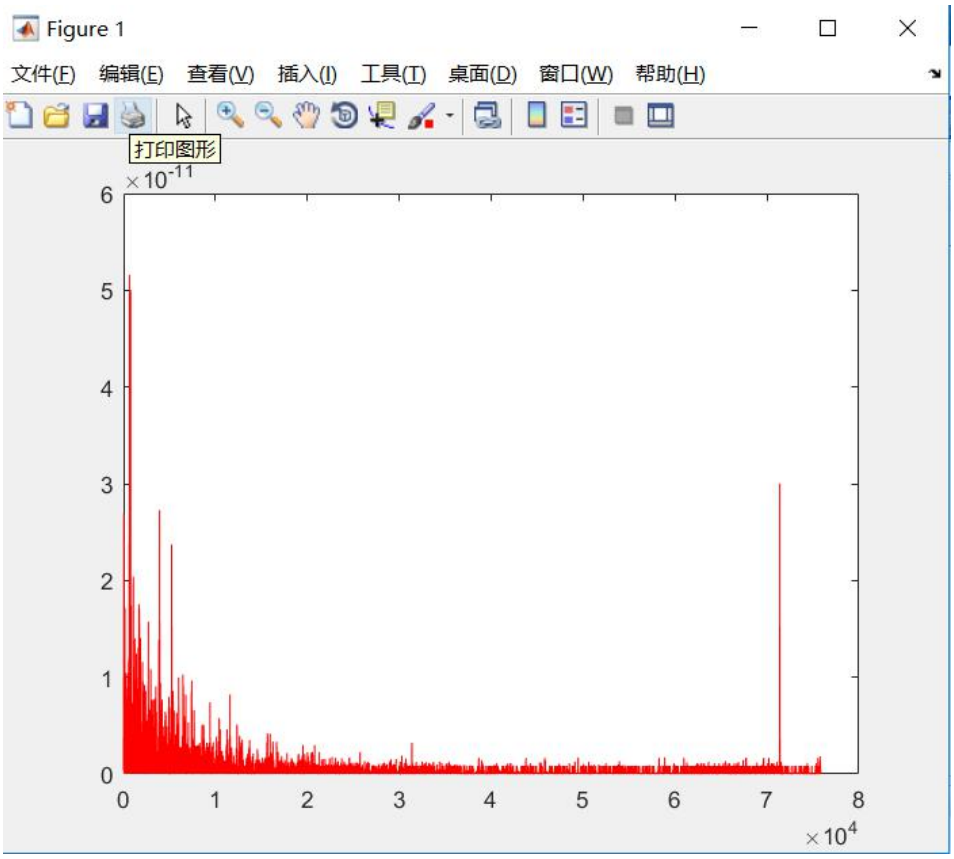
由图可以看到各项算法的运行时间，其中雅克比迭代法最慢，高斯赛德尔与 SOR 迭代法相近，在雅克比之前，接下来则是高斯消元，列主元，以及 CG 法，逐次计算时间减少。

四、改变松弛因子 w，查看 SOR 迭代法的迭代步数变化



上图可以看出，确定松弛因子的范围为 1 到 2，纵坐标为收敛时的迭代步数，那么我们可以看到，对于此时的情况，有 w=1.6 时，矩阵使用 SOR 迭代法求解的速度比使用其他 w 要快。

五、PageRank 的求解重要度排序分析



上图横坐标为网页节点的序号，从 0 到 75887，纵坐标为 PageRank 算法求解之后的重要度相对值，我们可以看到，在 0 到 10000 的区间内，网络节点的重要度成下滑趋势，之后逐渐趋于平稳。并且最重要的网络节点为第 18 个节点，依据为最终的重要度向量 R，具体数值参考下面截图。

13	3.0122e-04
14	1.4076e-04
15	1.5090e-04
16	4.1412e-05
17	3.6795e-05
18	3.9676e-05
19	0.0013
20	3.9397e-04
21	1.4801e-04
22	2.1681e-04
23	1.6946e-04

R 矩阵部分截图

```
>> PageRank
>> max(R)

ans =

    0.0013

>>
```

求最大重要度

由于下标从 1 开始，所以真正意义上 19 号下标，代表着从 0 节点开始的第 18 号节点。