题目一

问题描述

请实现下述算法，求解线性方程组 Ax=b，其中 A 为 n x n 维的已知矩阵，b 为 n 维的已 知向量，x 为 n 维的未知向量。

（1）高斯消去法。

（2）列主元消去法。

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。

算法设计

高斯消元法

消元过程：(1) $\mathrm{m_{ik}} = \frac{a_{ik}^k}{a_{kk}^k}$ (2) $\begin{cases} a_{ij}^{k+1} = a_{ij}^k - m_{ik}a_{kj}^k \\ b_i^{k+1} = b_i^k - m_{ik}b_k^k \end{cases}$

回代过程：$\begin{cases} x_n = \frac{b^n}{a^n} \\ x_k = (b_k^k - \sum_{j=k+1}^n a_{kj}^k x_j)/a_{kk}^k \end{cases}$

列主元消去法

1. 选取主元素，$\left|a_{j_k,k}\right| = max_{k\le i\le n}|a_{ik}|! = 0$,
2. 交换$(\mathrm{A^k}|\ \mathrm{b^k})$第 k 行与$\mathrm{i_k}$ 行的元素，再进行消元计算。
3. 回代求解：$\begin{cases} x_n = \frac{b^n}{a_{nn}} \\ x_k = (\ b_i - \sum_{j=i+1}^n a_{ij}x_j)/a_{ii}\ , i = n-1, \dots, 2, 1 \end{cases}$

数值实验：

使用 matlab 实现高斯消元法和列主元消去法，分别存储在 gauss.m 和 gauss_col.m 中。

高斯消元法

```
function [x] = gauss(A,b);
n = size(A,1);
x = zeros(n,1);
%高斯消元法
for k=1:n-1
    %消元计算
    for i = k+1:n %行
      mik = A(i,k)/A(k,k);
      for j=k+1:n %列
        A(i,j)=A(i,j)-mik*A(k,j);
      end
      b(i)=b(i)-mik*b(k);
    end
      %回代
    x(n) = b(n)/A(n,n);
    for i = n-1:-1:1
      x(i)=(b(i)-sum(A(i,i+1:n)*x(i+1:n)))/A(i,i);
    end
end
```

列主元消去法

```
function [x] = Gauss_col(A,b);
n = size(A,1);
x = zeros(n,1);
for k=1:n-1
  %选取当前列绝对值最大的数 index 表示其所在的行
    pivot = abs(A(k,k));
    index = k;
    for j =k+1:n
        if(abs(A(j,k))>pivot)
            pivot = abs(A(j,k));
            index = j;
        end
    end
    if(pivot==0)
      error('this matrix can''t be  solve by Gauss.');
    end
    if(index~=k)%说明交换
      tempRow = A(index,:);
      A(index,:)=A(k,:);
      A(k,:) = tempRow;
      tempB = b(index);
      b(index) = b(k);
      b(k) = tempB;
    end
    %消元计算
    for i = k+1:n %行
      mik = A(i,k)/A(k,k);
      for j=k+1:n %列
```

编写测试代码，测试计算时间并绘制曲线，测试完整代码在 testAndPlot.m 中

测试思路：

1. 使用 matlab 的 random 方法生成素服从独立同分布的正态分布矩阵 A，b

```
A10 = random('Normal',10,10,10,10);
B10 = random('Normal',10,10,10,1);
A50 = random('Normal',10,10,50,50);
```

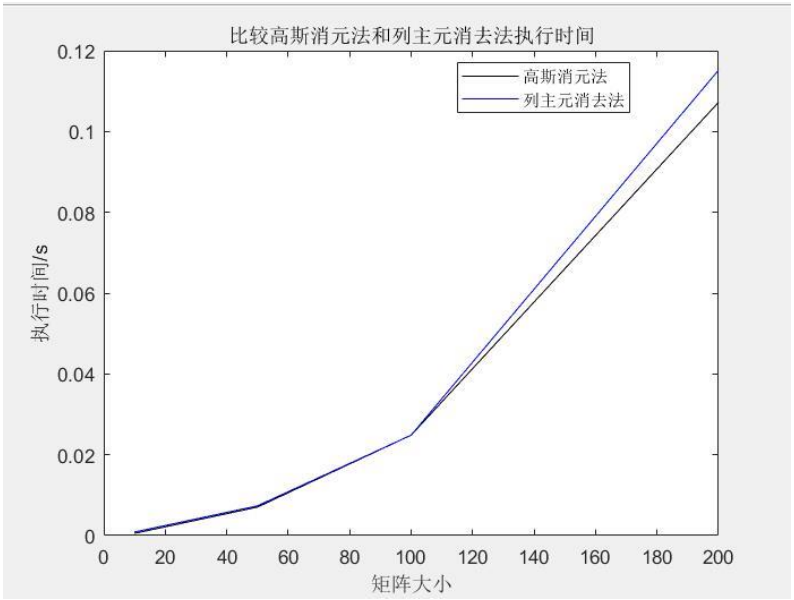
2. 定义两个数组分别存储高斯消去法和列主元消去法的执行时间，使用 tic 和 toc 搭配计算程序运行时间，调用写好的 gauss 和 gauss_col 函数

```
tic
gauss(A10,B10)
guassTime(1)= toc;
tic
gauss_col(A10,B10)
colTime(1) = toc;
```

3. 使用 plot 绘制程序运行时间图像。

```
plot(x,guassTime,'k',x,colTime,'b')
```

结果分析



由图可知，当矩阵较小时，高斯消去法和列主元消去法的执行时间差距不大，但当矩阵大于 100 左右

时，高斯消元法更快，并且随着矩阵大小增大，两者差距变大，思考是因为列主元消去法每次循环时都要选取主元，因此步骤更多，执行时间比高斯消元长。

题目二

问题描述

请实现下述算法，求解线性方程组 Ax=b，其中 A 为 n x n 维的已知矩阵，b 为 n 维的已 知向量，x 为 n 维的未知向量。

（1）Jacobi 迭代法。

（2）Gauss-Seidel 迭代法。

（3）逐次超松弛迭代法。

（4）共轭梯度法。

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。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。

算法设计

（1） Jacobi 迭代法

$A = D - L - U$，D 为对角矩阵，D 为上三角矩阵，U 为上三角矩阵

迭代公式 ： $Dx^{k+1} = (L+U)x^k + b$

（2） Gauss-Seidel 迭代法

迭代公式：$(D-L)x^{k+1} = Ux^k + b$

$$x_i \leftarrow 0.0(i = 1,2,\dots,n)$$

对于 $k = 1,2,\dots N_0$

$$x^{k+1} = (D-L)^{-1}(Ux^k + b)$$

N_0为最大迭代次数

（3） 逐次超松弛迭代法

$$A = D - L - U$$

ω 为松弛因子

$$x_i \leftarrow 0.0(i = 1,2,\dots,n)$$

对于 $k = 1,2,\dots N_0$

$$x^{k+1} = (D-\omega L)^{-1}(((1-\omega)D + \omega U)x^k + \omega b))$$

（4）共轭梯度法

1. 任取$x^0 \in R^n$ 计算$r^0 = b - Ax^0$，取 $p^0 = r^0$.

2. 对于 $k = 1,2,\dots N_0$ 计算

$\alpha_k = \frac{(r^k, r^k)}{(p^k, Ap^k)}$

$x^{k+1} = x^k + a_k p^k$

$r_{k+1} = r^k - \alpha_k A p^k, \beta_k = \frac{(r_{k+1}, r_{k+1})}{(r^k, r^k)}$

$p^{k+1} = r^{k+1} + \beta_k p^k$

3. 若$r^k = 0$,或$(p^k, Ap^k) = 0$ 计算停止，则$x^k = x^*$

数值实验

a. **绘制出算法的收敛曲线**

根据算法编写代码，分别存储在 jacobi.m、gaussSeidel.m、sor.m 和 conjgrad.m 中。

每个函数均返回求解得到的 x，以及迭代的误差数组。

首先编写代码绘制各算法的收敛曲线，完整代码在 test2andPlot.m 文件中。

1. 首先生成符合题目要求的矩阵：

```
x10 = rand(1,10)
V10 = diag(x10)
U10 = orth(rand(10))
A10 = U10*V10*U10'
B10 = random('Normal',10,10,10,1);
```

2. 分别调用不同算法，得到返回的相对误差向量

```
[x1,jacobiError] = jacobi(A10,B10);
[x2,gaussSeidelError] = gaussSeidel(A10,B10);
```

3. 绘制算法的收敛曲线

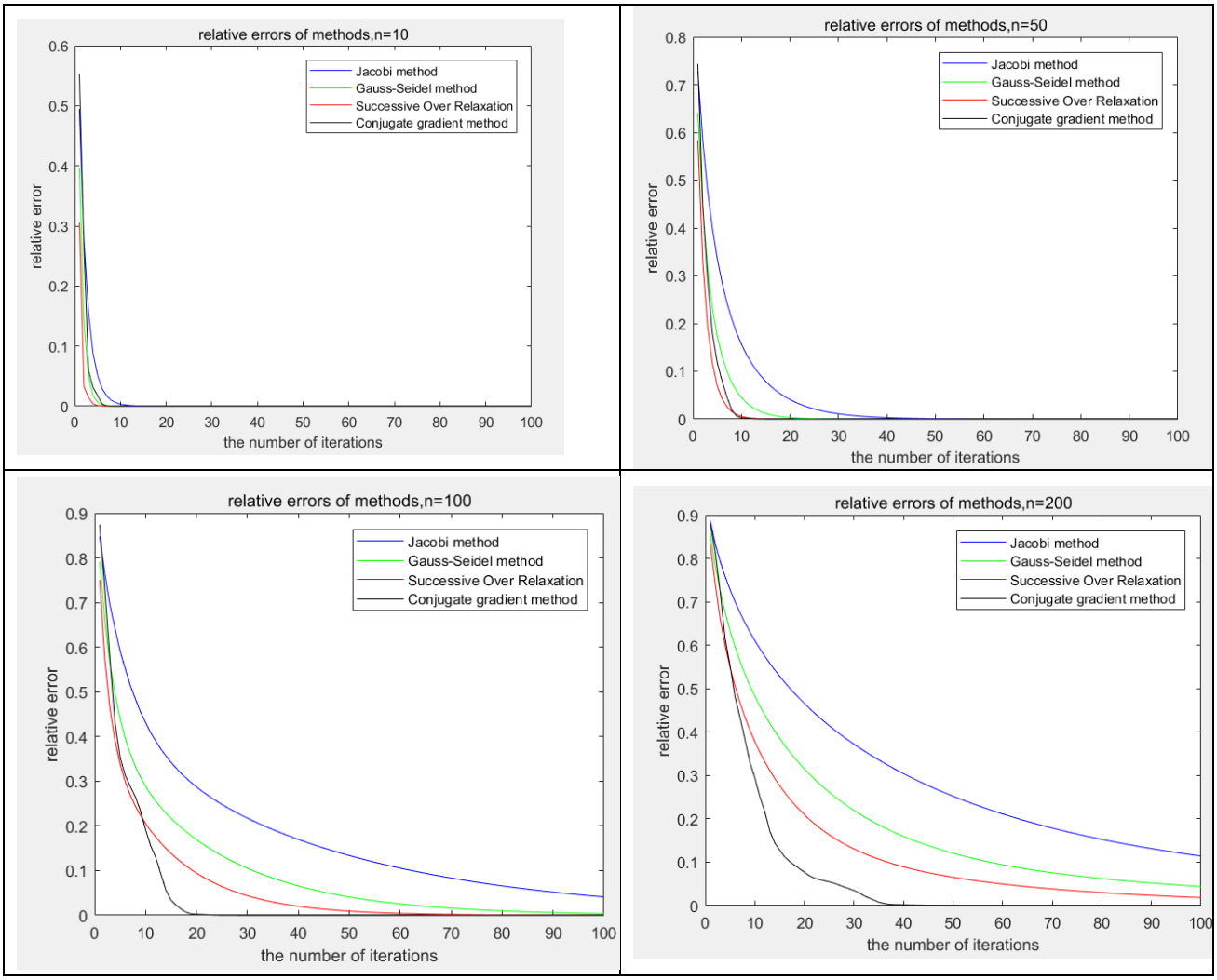
```
plot(x,jacobiError,'b',x,gaussSeidelError,'g',x,sorError,'r',x,conjgradError,'k');
```

计算相对误差方法：

使用 $x = A\backslash b$ 得到方程的精确解

利用公式 $\varepsilon^k = x^k - x^*$，计算每一步的误差。

结果分析

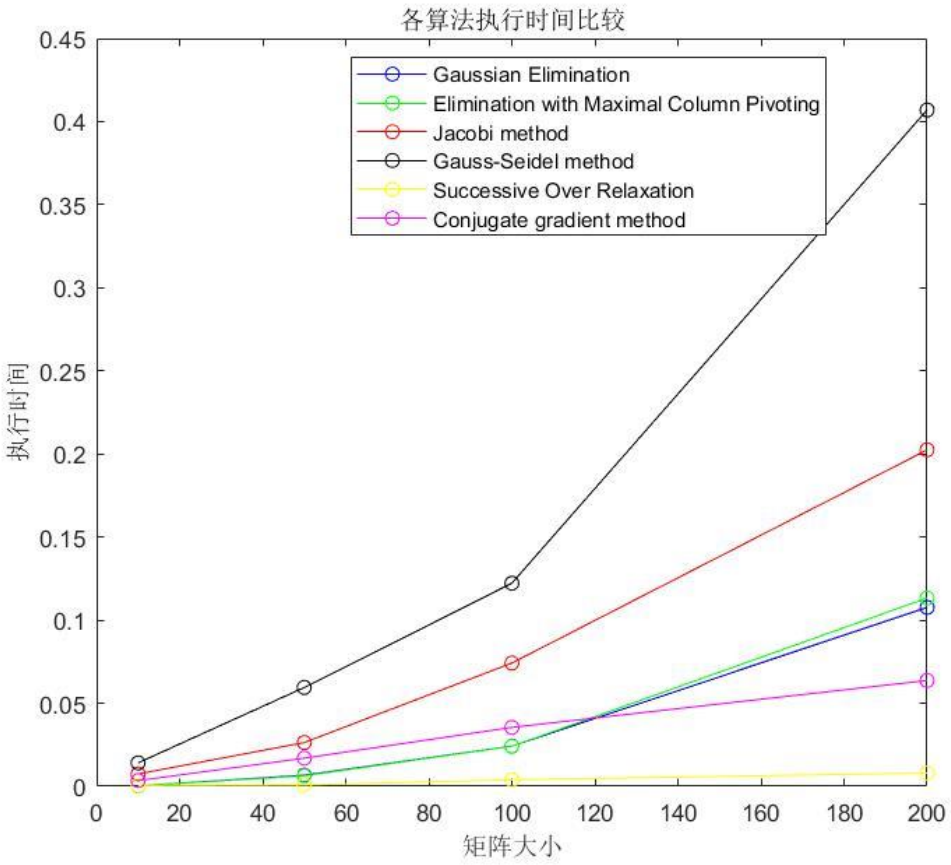


由实验结果观察可知，共轭梯度法，逐次超松弛迭代法，高斯-塞德尔迭代法，雅各比迭代法的收敛速度逐渐降低，并且对于图中的 n=100 的情况，雅各比迭代法不收敛，雅各比迭代法中对应的矩阵 B 的谱半径大于 1。实验过程也发现雅各比迭代法的收敛性确实比其他算法差。因为雅各比迭代法收敛不仅要求 A 本身是对称正定矩阵，2D-A 也应该是正定矩阵。而 A 为对称正定矩阵就可以保证高斯-塞德尔迭代法。

b. **比较 Jacobi 迭代法、Gauss-Seidel 迭代法、逐次超松弛迭代法、 共轭梯度法与高斯消去法、列主元消去法的计算时间**

1. 使用 tic 和 toc 测试程序的执行时间，得到各个算法的计算时间画图，完整测试文件在 compareTime.m 。

结果分析

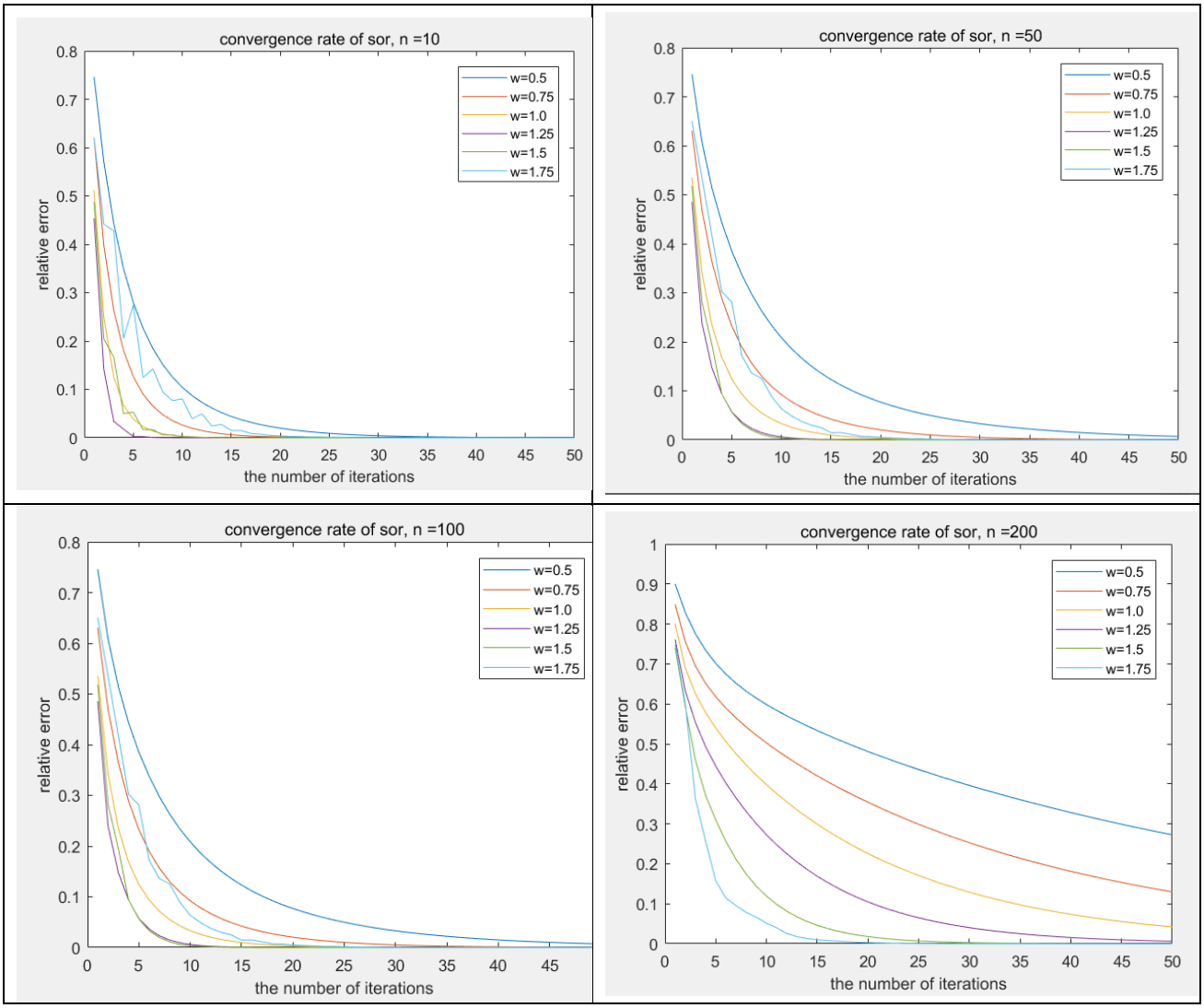


从图中可分析得到，当矩阵大小大于 120 左右时，高斯-塞德尔迭代法，雅各比迭代法，列主元消去法，高斯消去法，共轭梯度法，逐次超松弛迭代法的计算时间依次递减。当矩阵较小时，共轭梯度法的计算时间比列主元消去法和高斯消元法。在我看来，算法的具体实现方式对算法的执行时间也有影响，如一些语句的编写方式对时间也会有影响。所以得到的结果只适合参考。

c. 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。

松弛因子 ω 的取值应在（0,2）算法才会收敛，当ω = 0 时，曲线平行于 x 轴，当ω = 2 时，曲线振荡，这两种情况均不能收敛，因此测试的时候 ω 取值 0.5，0.75，1.0，1.5，1.5，1.75。对于 n=10, 50, 100, 200 进行测试。

结果分析



由结果可以分析得到，当矩阵足够大时，松弛因子 ω 越小，收敛速度越慢。而这也符合我们对超松弛迭代算法的理解，ω 越大，收敛速度越快，ω 越小，算法更稳定。但是当矩阵较小时，收敛速度差距不明显。

问题描述

在 Epinions 社交数据集（https://snap.stanford.edu/data/soc-Epinions1.html）中， 每个网络节点可以选择信任其它节点。借鉴 Pagerank 的思想编写程序，对网络节点的受信 任程度进行评分。在实验报告中，请给出伪代码。

算法设计

由于数据节点多，无法直接创建矩阵，因此需要用稀疏矩阵存储，迭代法计算 pagerank 的值。一开始的 PR 值设为 $\frac{1}{n}$。

借鉴 pagerank 思想。如果一个节点没有信任任何页面，则改为其对信任所有网络节点。避免 0 作分母。主要公式：

$$\mathrm{PageRank}(\mathrm{P_i}) = \frac{1-\mathrm{d}}{\mathrm{N}} + d \sum_{p_j \in M(p_i)} \frac{PageRank(p_j)}{L(p_j)}$$

其中：$\mathrm{p_1}, \mathrm{p_2}, \ldots, \mathrm{p_n}$ 是被研究的页面，$M(p_i)$是信任$\mathrm{p_j}$页面的集合，$\mathrm{L(p_j)}$是$\mathrm{p_j}$信任页面的数量，N是所有网络节点的数量。

考虑存在网络节点只信任自己，那么在多次迭代过程中，其 PR 值只增不减，会使得评分不合理，因此引入阻尼系数 d。使得迭代过程中若遇到该类型的结点，有（$1-\mathrm{d}$）的概率会信任随机的一个网页，信任每个节点的概率是一样的。

对于给定的 ϵ，不断迭代直到，$|\mathrm{P_{n+1}} - \mathrm{P_n}| < \epsilon$.

伪代码

伪代码分为三个部分，分别为 main 主函数，processdata 处理数据，pagerank 计算各结点的 PR 值。N 表示结点个数

其中 matrix 表示存结点信任关系的稀疏矩阵，outdegree 表示网络结点信任结点的个数，damping_factor 表示阻尼系数 d，*delta* **表示** ϵ，用来作为迭代的结束条件。

```
procedure main
    matrix[n][n]
    outdegree[n]
    processdata(matrix,outdegree,n)
    pagerank(n,matrix,outdegree, damping_factor, max_iterations ,delta)
end procedure
```

```
procedure processdata
    for line in file do
        odom ← split line
        matrix[odom[1],odom[0]]←1
        out_degree[odom[0]] ←out_degree[odom[0]]+1
    end for
    for i = 0 to n-1 do
        if outdegree[i] ==0 then
            outdegree[i] = n
        end if
    end for
end procedure
```

```
procedure  pagerank (n,matrix,outdegree,damping_factor, max_iterations)
    damping_value  ←  (1.0 - damping_factor) / n
    page_rank[n]
    for i =0 to max_iterations do
        change ← 0
```

```
        for node=0 to N-1 do
            rank ← 0
            for in_page in matrix.getrow(node).nonzero() do
                rank ←rank+damping_factor*((page_rank[in_page] / out_degree[in_page])
            end for
            rank ← rank+damping_value
            change ← change+abs(page_rank[node] - rank)
            page_rank[node] ← rank
        end for
        if change < min_delta then
            flag = True
            break
        end if
    end for
    return page_rank
end procedure
```

数值实验

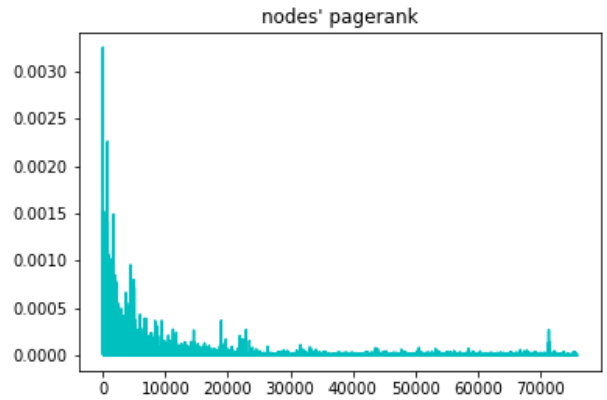
这部分选择 python 实现, 使用 scipy 库的 sparse 部分存储稀疏矩阵,其中稀疏矩阵有不同的存储方式, 由于计算 PR 值过程中，对于每个结点，都要获取对应的指向该结点的所有结点，在矩阵中相当于多次进行 getrow(i)的操作，返回 i 行的所有数据，scipy.sparse 中 lil_matrix 是基于行链接列表的稀疏矩阵，所以执行 getrow 操作比较快。

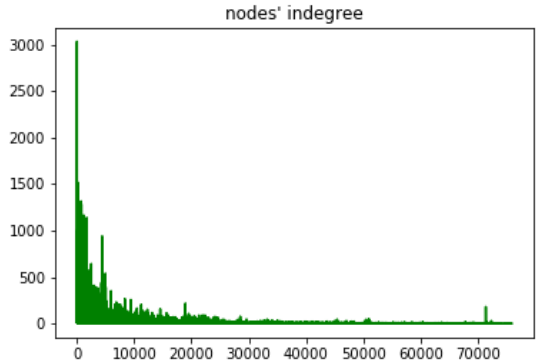
根据设计的伪代码编写代码，完整代码文件在 pagerank.py 中。

设定即阻尼稀疏为 0.85, 最大迭代次数为 100, ϵ 为 0.00001. 一般迭代 30 次左右就能满足 $|P_{n+1} - P_n| < \epsilon$，结束循环。

结果分析

各个节点的 PR 值和入度数如图:





分析 PR 值:

总数	**75888**

最大值	3.250232e-03
最大值位置	18
最小值	1.976597e-06
中位数	2.614664e-06
均值	9.444267e-06
方差	1.806934e-09
标准差	4.250804e-05

根据实验结果，节点 18 最受信任，对应的 PR 值是 3.250232e-03，最小的 PR 值为 1.976597e-06，多个节点都具有最小值。以下给出排名前 10 的节点：

节点位置	PR 值
18	0.003250
737	0.002258
118	0.001521
1719	0.001489
136	0.001424
790	0.001411
143	0.001402
40	0.001308
1619	0.001101
725	0.001072

完整 PR 值数据 sortedreslut.txt 中。