# 数值计算实验1

**数据科学与计算机学院**

**梁育诚**

**学号 16340133**

**班级 教务二班**

---

## 内容1

**请实现下述算法，求解线性方程组Ax=b，其中A为n×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）高斯消元法。
（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。

### 问题描述（高斯消元法）

对于一个这样的线性方程组

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases} \tag{1.1}$$

，或写成矩阵形式

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

，简记为$\boldsymbol{Ax = b}$.
我们可以使用高斯消元法来进行求解。

### 算法设计（高斯消元法）

高斯消元法主要分为消元和回代两个部分。消元的目的是把A矩阵变成一个上三角矩阵，回代的目的是利用已知的x去求解每一行中仅有的一个未知的x。

(1) 第一步（*k=1*）
设$a_{11}^{(1)} \neq 0$，首先计算乘数

$$m_{i1} = a_{i1}^{(1)} / a_{11}^{(1)}, i = 2, 3, \ldots, n.$$

用$-m_{i1}$乘方程组（1.1）的第1个方程，加到第$i$个（$i = 2$，$3$，$4$，$\ldots$，$n$）方程上，消去方程组（1.1）的从第2个方程到第$n$个方程中的未知数$x_1$，得到与方程组（1.1）等价的线性方程组：

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{bmatrix}$$

简记为

$$A^{(2)}x = b^{(2)}，$$

其中，$A^{(2)}$，$b^{(2)}$的元素计算公式为

$$\begin{cases} a_{ij}^{(2)} = a_{ij}^{(1)} - m_{i1}a_{1j}^{(1)}, i, j = 2, 3, \ldots, n. \\ b_i^{(2)} = b_i^{(1)} - m_{i1}b_1^{(1)}, i = 2, 3, \ldots, n. \end{cases}$$

(2) 第$k$次消元（$k = 1, 2, \ldots, n-1$）
假设第$k-1$步已经完成，我们有如下线性方程组：

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1k}^{(1)} & \cdots & a_{1n}^{(1)} \\ & a_{22}^{(2)} & \cdots & a_{2k}^{(2)} & \cdots & a_{2n}^{(2)} \\ & & \ddots & \vdots & & \vdots \\ & & & a_{kk}^{(k)} & \cdots & a_{kn}^{(k)} \\ & & & a_{nk}^{(k)} & \cdots & a_{nn}^{(k)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_k^{(k)} \\ \vdots \\ b_n^{(k)} \end{bmatrix},$$

简记为$A^{(k)}x = b^{(k)}$.
设$a_{kk}^{(k)} \neq 0$，计算乘数

$$m_{ik} = a_{ik}^{(k)} / a_{kk}^{(k)}, i = k+1, k+2, \ldots, n.$$

用$-m_{ik}$乘方程组第$k$个方程，并加到第$i$个方程（$i = k+1, \ldots, n$），消去从第$k+1$个方程到第$i$个方程中的未知数$x_k$，得到$A^{(k+1)}x = b^{(k+1)}$.
其中$A^{(k+1)}$，$b^{(k+1)}$元素的计算公式为：

$$\begin{cases} a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik}a_{kj}^{(k)}, i, j = k+1, \ldots, n. \\ b_i^{(k+1)} = b_i^{(k)} - m_{ik}b_k^{(k)}, i = k+1, \ldots, n. \end{cases},$$

显然$A^{(k+1)}$中从第$1$行到第$k$行与$A^{(k)}$相同。

（3）经过$n-1$步消元计算后，我们得到$A^{(n)}x = b^{(n)}$：

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ & & \ddots & \vdots \\ & & & a_{nn}^{(n)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(k)} \end{bmatrix},$$

若A是非奇异矩阵，且$a_{kk}^{(k)} \neq 0$（$k = 1, 2, \ldots, n-1$），求解矩阵的回代公式为：

$$\begin{cases} x_n = b_n / a_{nn}^{(n)}, \\ x_k = b_k^{(k)} - \sum_{j=k+1}^{n} a_{ki}^{(k)} x_j / a_{kk}, k = n-1, n-2, \ldots, 1. \end{cases},$$

这样，高斯的消元与回代就结束了，我们就可以得到x的解。

### 问题描述（列主元消元法）

在上述高斯消元法中不难发现，在每一次的消元步骤中，我们都需要除以当前列的主元。如果这个主元是一个非常小的数，那么这样将会导致精度丢失，影响算法的稳定性。因此我们需要一个改进的方法，就是列主元消元法，其基本的思路和高斯消元法一致，不同之处在于每一步的消元前，都选取该列中**绝对值最大**的元素作为主元，这样可以保证数值较小的数不会丢失精度。

### 算法设计（列主元消去法）

由于算法的大部分与高斯消元法类似，在这里只讲述不同的地方。
对于k=1,2,...,n-1
① 按列选绝对值最大的元素作为主元：

$$|a_{i_k,k}| = \max_{k \leq i \leq n} |a_{ik}|$$

② 如果$a_{i_k,k} = 0$，则停止计算。
③ 如果$i_k = k$，则说明当前元素是主元，无需交换，执行④
交换$A$中的行和对应$b$中的元素：$a_{kj} \leftrightarrow a_{i_k,j}(j = k, k+1, \cdots, n)$

$$b_k \leftrightarrow b_{i_k}$$

将上述步骤放在高斯消元法每次消元步骤之前即可。

### 数值实验

编写好高斯消元法和列主元消元法的matlab代码，随机生成矩阵。以矩阵的维数n为横坐标，以求解每个矩阵的运行时间作为纵坐标，进行作图分析，对比两个算法的数值稳定性。
高斯消元法：

```
function[x]=gauss_elim(A,b)
% A为系数矩阵且必须为方阵
% b为常数项向量
n=size(A,1);
x=zeros(1,n);
%分解
for k=1:n
    %顺序主子式为0，则退出
    w=det(A(1:k,1:k))
    if (w==0)
        return;
    end
    %若主元为0，则换一个

    if(A(k,k)==0)
        t=min(find(A(k+1:n,1)~=0+k));
        if(isempty(t))
            return
        end;
        temp=A(k,:);tb=b(k);
        A(k,:)=A(t,:);b(k)=b(t);
        A(t,:)=temp;b(t)=tb;
    end;

    for i=k+1:n
        m=A(i,k)/A(k,k);
        for j=k+1:n
            A(i,j)=A(i,j)-m*A(k,j);
        end
        b(i) = b(i)-m*b(k);
    end
end

%回代
x(n)=b(n)/A(n,n);
for i=n-1:-1:1
    sum = 0;
    for j=i+1:n
        sum = sum + A(i,j)*x(j);
    end
    x(i) = (b(i)-sum)./A(i,i);
end
```

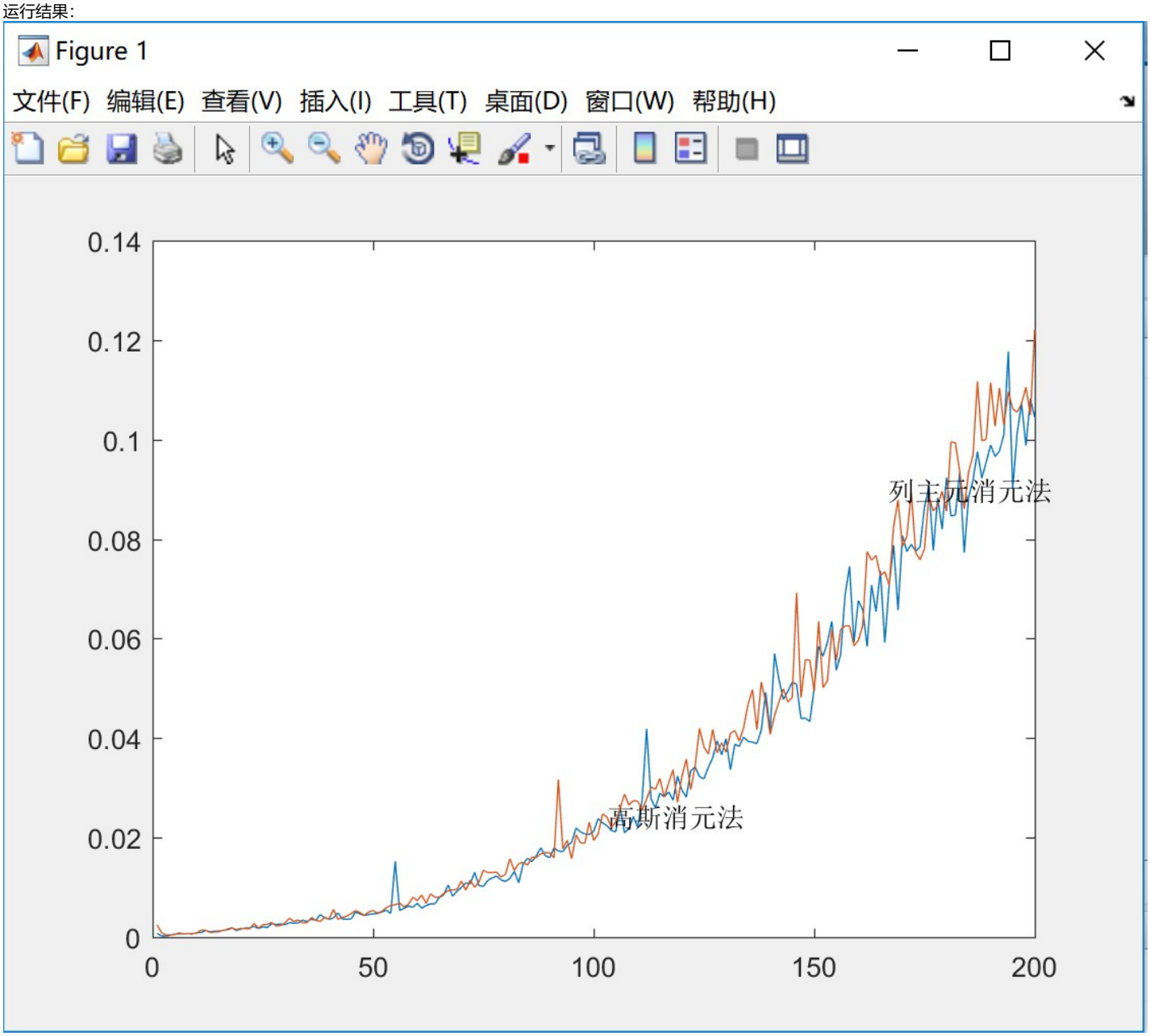
**列主元消元法：**

```
function[x] = gauss_elim_pro(A, b)
% A为系数矩阵且必须为方阵
% b为常数项向量

n=length(b);
x=zeros(n,1);
c=zeros(1,n);

%分解
for i = 1: n
    %顺序主子式为0，则退出
    w=det(A(1:i,1:i))
    if (w==0)
        return;
    end
    max=abs(A(i,i));
    m = i;
    for j = i + i:n
        if max < abs(A(j,i))
            max = abs(A(j,i));
            m = j;
        end
    end
    %如果最大的主元不在本行，需要交换
    if (m ~= i)
        for k = i:n
            c(k) = A(i,k);
            A(i,k)=A(m,k);
            A(m,k)=c(k);
        end
        d1 = b(i);
        b(i)=b(m);
        b(m)=d1;
    end
```

```
35.
36.     for k = i+1:n
37.         for j = i+1:n
38.             A(k,j) = A(k,j)-A(i,j)*A(k,i)./A(i,i);
39.         end
40.         b(k)=b(k)-b(i)*A(k,i)./A(i,i);
41.         A(k,i)=0;
42.     end
43. end
44.
45. % 回代
46. x(n)=b(n)/A(n,n);
47. for i = n-1:-1:1
48.     sum=0;
49.     for j=i+1:n
50.         sum = sum + A(i,j)*x(j);
51.     end
52.     x(i)=(b(i)-sum)./A(i,i);
53. end
```

**结果分析**

运行结果:




其中蓝线是高斯消元法，橙线是

可以看出，两条线几乎是重叠的，这也说明两个算法在时间上是差不多的，但是列主元消元法的线更平滑一些，但是整体比高斯消元法的线高一些，这些说明了列主元消元法牺牲时间以换取更好的数值稳定性，其中的时间就花在了寻找列主元上。

## 内容2

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 与 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。

### 问题描述

这部分我们要实现和分析的求解矩阵的迭代法。考虑现行方程组Ax=b，其中A为非奇异矩阵，当A是**低阶稠密矩阵**时，可以使用内容一中的高斯消元法以及列主元消元法来进行求解。但是，当A是**高阶稀疏矩阵**时，就可以利用迭代法来进行求解。迭代法利用了A中零元素较多的特点。

### 算法设计（雅可比迭代法）

首先我对迭代法的思路进行大致的分析，后续算法中就不再赘述。对于一个线性方程组，第k行中，如果我已知除$x_k$之外的其余$x_i(i \neq k)$，那么我就可以根据这个方程把$x_k$求解出来。因此，我可以首先假设已知$x_0$，然后不断重复以上的步骤，即迭代，最终可以求得一个解。如果这个矩阵是具有收敛性质的，那么这个解x就可以无限逼近精确解$x^*$。

我们通过变换，可以将Ax=b等价变换为$x^* = Bx + f$。构造向量序列

$$x^{(k+1)} = Bx^{(k)} + f$$

，其中$k$表示迭代次数。
如果$\lim\limits_{x \to \infty} x^{(k)}$存在，则说明**迭代法收敛**，否则，该**迭代法发散**。所以我们每次使用迭代法求解矩阵的时候就需要判断矩阵是否收敛。

这里给出一个判断矩阵收敛的定理：对于一个线性方程组，迭代法收敛的充要条件是矩阵B的谱半径ρ(B) < 1。

现在我来讲解雅可比迭代法。雅可比迭代法是对上述迭代过程的详细展开。对于线性方程组的系数矩阵A，可分解为三部分：

$$A = \begin{bmatrix} a_{11} & & & \\ & a_{22} & & \\ & & \ddots & \\ & & & a_{nn} \end{bmatrix} - \begin{bmatrix} 0 & & & & \\ -a_{21} & 0 & & & \\ \vdots & \vdots & \ddots & & \\ -a_{n-1,1} & -a_{n-1,2} & \cdots & 0 & \\ -a_{n1} & -a_{n2} & \cdots & -a_{n,n-1} & 0 \end{bmatrix} - \begin{bmatrix} 0 & -a_{12} & \cdots & -a_{1,n-1} & -a_{1n} \\ & 0 & \cdots & -a_{2,n-1} & -a_{2n} \\ & & \ddots & \vdots & \vdots \\ & & & 0 & -a_{n-1,n} \\ & & & & 0 \end{bmatrix} = D - L - U$$

设$a_{ii} \neq 0 (i = 1, 2, \ldots, n)$，可以得到Ax=b的雅可比迭代法：

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

其中$B = I - D^{-1}A = D^{-1}(L + U) = J, f = D^{-1}b$，称J为解Ax=b的雅可比迭代法的迭代矩阵。
解Ax=b的雅可比迭代法的计算公式为：

$$\begin{cases} x^{(0)} = (x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)})^T \\ x_i^{(k+1)} = (b_i - \sum\limits_{j=1, j \neq i}^{n} a_{ij} x_j^{(k)}) / a_{ii}, \\ i = 1, 2, \ldots, n; k = 0, 1, \ldots \text{表示迭代次数。} \end{cases}$$

根据上述计算公式就可以使用雅可比迭代法求解了。

### 算法设计（高斯-赛德尔迭代法）

高斯-赛德尔迭代法可以看作是对雅可比迭代法的优化改进。它的思路在于，更新每一步$x_i$的时候，可以利用已经更新$x_j(j < i)$的值，这样可以减少迭代次数，因为它计算每一个分量的时候是基于当前的最新分量。
设$a_{ii} \neq 0 (i = 1, 2, \ldots, n)$，可以得到Ax=b的高斯-赛德尔迭代法：

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

其中$B = I - (D - L)^{-1}A = (D - L)^{-1}U = G, f = (D - L)^{-1}b.$称$G = (D - L)^{-1}U$为解$Ax = b$的高斯赛德尔迭代法的迭代矩阵。
求解Ax=b的高斯赛德尔迭代法计算公式为

$$\begin{cases} x^{(0)}=(x_1^{(0)},x_2^{(0)},\dots,x_n^{(0)})^T\text{，初始向量} \\ x_i^{(k+1)}=(b_i-\sum\limits_{j=1,j\neq i}^{i-1}a_{ij}x_j^{(k+1)}-\sum\limits_{j=i+1}^{n}a_{ij}x_j^{(k)})/a_{ii}, \\ i=1,2,\dots,n;k=0,1,\dots\text{表示迭代次数。} \end{cases}$$

这就是高斯赛德尔迭代法的计算公式。

### 算法设计（超松弛迭代法）

超松弛迭代法，SOR，是对于高斯赛德尔迭代法的改进。它的关键之处在于松弛因子$\omega$，这个因子可以控制收敛的方向来控制迭代速度和迭代精度。由于迭代速度和迭代精度是相互矛盾的，如果我想得到更快的迭代速度（倾向于向新的x靠近），那么我就有可能偏离了我的正确值；如果我追求迭代精度（倾向于向旧的x靠近）那么，我的迭代速度就很慢。松弛因子$\omega$就是用来调节这两者的，务求找到一个平衡点。这里直接给出SOR的计算方法。

$$\begin{cases} x^{(0)}=(x_1^{(0)},x_2^{(0)},\dots,x_n^{(0)})^T\text{，初始向量} \\ x_i^{(k+1)}=x_i^{(k)}+\omega(b_i-\sum\limits_{j=1}^{i-1}a_{ij}x_j^{(k+1)}-\sum\limits_{j=i}^{n}a_{ij}x_j^{(k)})/a_{ii}, \\ i=1,2,\dots,n;k=0,1,\dots\text{表示迭代次数}, \\ \omega\text{为松弛因子} \end{cases}$$

使用SOR的时候，需要注意的是矩阵A必须是**对称正定矩阵**，同时$0<\omega<2$。

### 算法设计（共轭梯度法）

谈到共轭梯度法，首先要了解与线性方程组等价的变分问题。
设$A=(a_{ij})\in R^{n\times n}$是**对称正定矩阵**，$b=(b_1,b_2,\cdots,b_n)^T$，求解的线性方程组为Ax=b。考虑如下定义的二次函数

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

，该函数等价于对应的线性方程组。
性质1： 对一切$x\in R^n,\varphi(x)$的梯度为

$$\nabla\varphi(x)=Ax-b$$

性质2： 对一切$x,y\in R^n$及$\alpha\in R$，

$$\begin{aligned}\varphi(x+\alpha y)&=\frac{1}{2}(A(x+\alpha y),x+\alpha y)-(b,x+\alpha y)\\&=\varphi(x)+\alpha(Ax-b,y)+\frac{\alpha^2}{2}(Ay,y)\end{aligned}$$

性质3：设$x^*=A^{-1}b$是线性方程组Ax=b的解，则

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

，
且对一切$x\in R^n$，有

$$\begin{aligned}\varphi(x)-\varphi(x^*)&=\frac{1}{2}(Ax,x)-(Ax^*,x)+\frac{1}{2}(Ax^*,x^*)\\&=\frac{1}{2}(A(x-x^*),x-x^*\text{）}\end{aligned}$$

根据上述二次函数的性质，我们可以知道，当$x^*$为令$\varphi(x)$最小时，$x^*$是该方程的解。这样，我们就把求解矩阵的问题转化为求解二次函数极小值点的问题

共轭梯度法的思想是，对于初始的一个x，使用方向$p^{(0)},p^{(1)},\dots,p^{(k-1)}$进行k次搜索，求得$x^{(k)}$，然后确定$p^{(k)}$的方向，这样能使x^{(k+1)}更快地求得$x^*$。
其主要迭代公式为：

$$\begin{aligned}x^{(k+1)}&=x^{(k)}+\alpha_kp^{(k)},\\x^{(k)}&=\alpha_0p^{(0)}+\alpha_1p^{(1)}+\cdots+\alpha_{k-1}p^{(k-1)}。\end{aligned}$$

\alpha_k的值可以根据如下公式计算：
因为我们每步的求解过程中，都需要基于当前的x，求出下一个极小值，即

$$\varphi(x^{(k+1)})=\min_{\alpha\in R}\varphi(x^{(k)}+\alpha p^{(k)})$$

，
由于

$$\varphi(x^{(k)}+\alpha p^{(k)})=\varphi(x^{(k)})+\alpha(Ax^{(k)}-b,p^{(k)})+\frac{\alpha^2}{2}(Ap^{(k)},p^{(k)})$$

，
对上式求导，求出极小值

$$\frac{d\varphi(x^{(k)}+\alpha p^{(k)})}{d\alpha}=(Ax^{(k)}-b,p^{(k)})+\alpha(Ap^{(k)},p^{(k)})=0,$$

解得

$$\alpha_k=-\frac{(Ax^{(k)}-b,p^{(k)})}{(Ap^{(k)},p^{(k)})}$$

又因为我们希望$p^{(k)}$能使$x^{(k+1)}$能够更快地求得$x^*$，因此我们有

$$\varphi(x^{(k+1)})=\min_{x\in span\{p^{(0)},p^{(1)},\cdots,p^{(k)}\}}\varphi(x)$$

，
然后我们把x分解成$x=y+\alpha p^{(k)}$，所以得到

$$\varphi(x)=\varphi(y+\alpha p^{(k)}=\varphi(y)-\alpha(Ay,p^{(k)})-\alpha(b,p^{(k)})+\frac{\alpha^2}{2}(Ap^{(k)},p^{(k)})$$

，
由于我们需要对x求导，又因为x用y来表示，所以对y求偏导。因为要为0，所以交叉项$(Ay,p^{(k)})$必须为0，这样一组向量p称为**共轭向量。**
以上就是对于共轭梯度法的核心思想的分析。下面直接给出算法：
（1）任取初始向量$x^{(0)}$，一般取为0，计算$r^{(0)}=b-Ax^{(0)}$，取$p^{(0)}=r^{(0)}$.
(2)对$k=0,1,\cdots,$计算

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

$$x^{(k+1)}=x^{(k)}+\alpha_k p^{(k)}$$

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

$$p^{(k+1)}=r^{(k+1)}+\beta p^{(k)}$$

(3)若$r^{(k)}=0$，或$(p^{(k)},Ap^{(k)})=0$，则计算停止，得出结果。
这就是共轭梯度法的简要算法分析。

### 数值实验

迭代法依赖于程序的循环结构，对于迭代法我们要设置迭代结束的标志，为了测试需要，每个迭代法可以自定义迭代的次数以用作迭代结束的判断。在正规的迭代中，迭代的结束标志应该是一个误差的范围。
雅可比迭代

```
function[xn]=jacobi(A,b,max)
n=size(A,1);
eps = 0.000001;
if nargin == 2
    max = 200;
end
x0=zeros(n,1);
xn=zeros(n,1);
x0(1)=1;
%{
D = diag(diag(A)); % 求A的对角矩阵
L = -tril(A, -1); % 求A的下三角矩阵
U = -tril(A,1); % 求A的上三角矩阵
B = D\(L+U);
f = D\b;
x = B*x0+f;
%}
times = 0;% 迭代次数
while norm(xn-x0)>=eps && times <= max
    times=times+1;
    x0=xn;

    for i=1:n
        sum = 0;
        for j =1:n
            if (j~=i)
                sum = sum + A(i,j) * x0(j);
            end
        end
    xn(i)=(b(i)-sum)/A(i,i);
    end

    if(times>=max)
        return;
    end
```

```
end
disp(times);
disp(xn);
```

高斯赛德尔迭代

```
function[xn]=Gauss_Seidel(A,b, max)
n=size(A,1);
if nargin == 2
    max = 200;
end
eps = 0.000001;

x0=zeros(n,1);
xn=zeros(n,1);
x0(1)=1;
%{
D = diag(diag(A)); % 求A的对角矩阵
L = -tril(A, -1); % 求A的下三角矩阵
U = -tril(A,1); % 求A的上三角矩阵
B = D\(L+U);
f = D\b;
x = B*x0+f;
%}
% 判断有没有唯一解
if det(A) == 0
    return;
end

times = 0;% 迭代次数
while norm(xn-x0)>=eps && times <= max
    times=times+1;
    x0=xn;

    for i=1:n
        sum1 = 0;
        sum2 = 0;
        for j =1:i-1
                sum1 = sum1 + A(i,j) * xn(j);
        end
        for j=i+1:n
            sum2 = sum2 + A(i,j) * x0(j);
        end
    xn(i)=(b(i) - sum1 - sum2)/A(i,i);
    end

    if(times>=max)
        return;
    end
end
disp(times);
disp(xn);
```

超松驰迭代

```
function[xn]=SOR(A,b,w,max)
n=size(A,1);
eps = 0.000001;
if nargin == 3
    max = 200;
end
x0=zeros(n,1);
xn=zeros(n,1);
x0(1)=1;
%{
D = diag(diag(A)); % 求A的对角矩阵
L = -tril(A, -1); % 求A的下三角矩阵
U = -tril(A,1); % 求A的上三角矩阵
B = D\(L+U);
f = D\b;
x = B*x0+f;
%}
times = 0;% 迭代次数
while norm(xn-x0)>=eps && times <= max
    times=times+1;
    x0=xn;

    for i=1:n
        sum1 = 0;
```

```
        sum2 = 0;
        for j =1:i-1
                sum1 = sum1 + A(i,j) * xn(j);
        end
        for j=i:n
            sum2 = sum2 + A(i,j) * x0(j);
        end
    xn(i)=x0(i) + w * (b(i) - sum1 - sum2)/A(i,i);
    end

    if(times>=max)
        return;
    end
end
disp(times);
disp(xn);
```

共轭梯度法

```
function[x] = CG(A,b,max)
n = size(A,1);
x0 = zeros(n,1);
r0=b-A*x0;
p0=r0;

if nargin == 2
    max = 200;
end
eps = 1.0e-6;
times = 0;
while 1
    if ((abs(p0) < eps))
        break;
    end
    if (times > max)
        break;
    end
    times = times + 1;
    a0 = (r0' * r0) / (p0' * A*p0); % 多次使用
    x1 = x0 + a0*p0;

    r1 = r0 - a0*A*p0;

    b0 = (r1'*r1)/(r0'*r0);

    p1 = r1 + b0 *p0;

    %只是用到前后两层的向量，所以节省内存开销，计算完后面      一层，可以往回覆盖掉没用的变量
    x0 = x1;
    r0 = r1;
    p0 = p1;
end
x = x0;
end
```

## 结果分析

### 四个迭代法的误差分析对比

随机生成维数n=10,50,100,200四个矩阵，对于每一个矩阵，分用雅可比迭代法、高斯赛德尔迭代法、超松驰迭代法、共轭梯度法四种方法进行求解，记录迭代次数与误差。以迭代次数为横坐标，相对误差为纵坐标，作图.
n=10

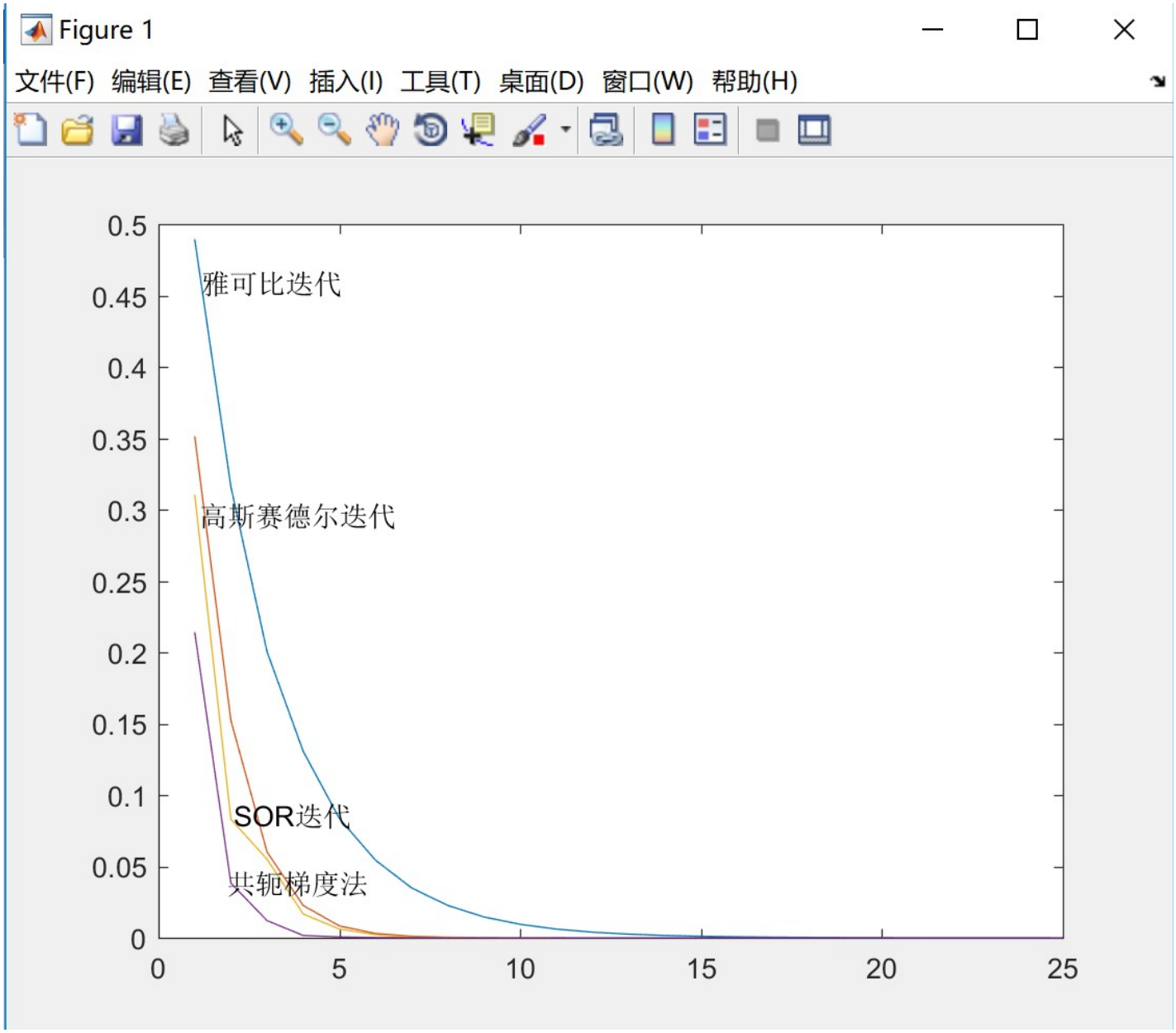


Figure 1
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
雅可比迭代
高斯赛德尔迭代
SOR迭代
共轭梯度法

n=20

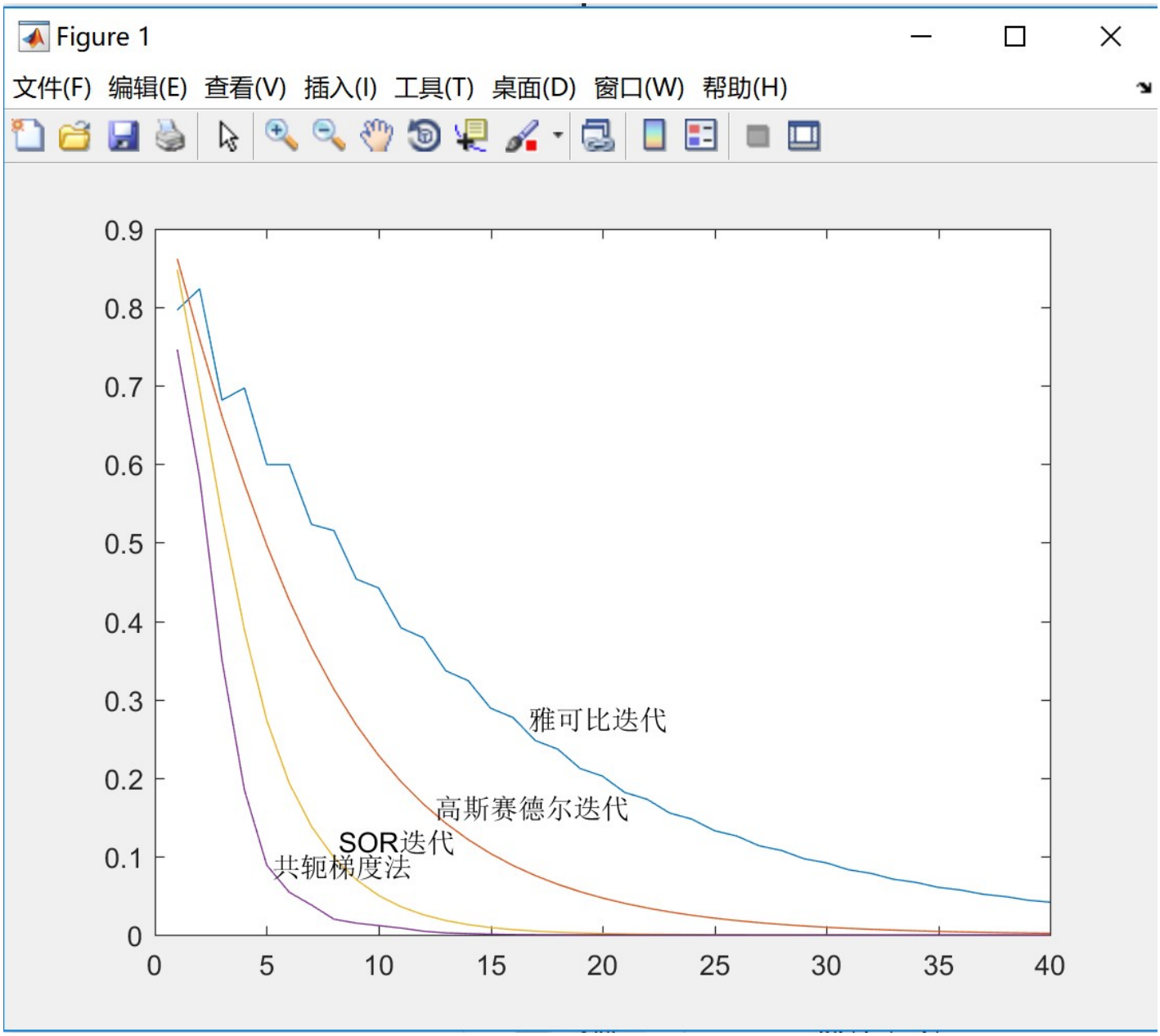


Figure 1
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
5
10
15
20
25
30
35
40
雅可比迭代
高斯赛德尔迭代
SOR迭代
共轭梯度法

n=100

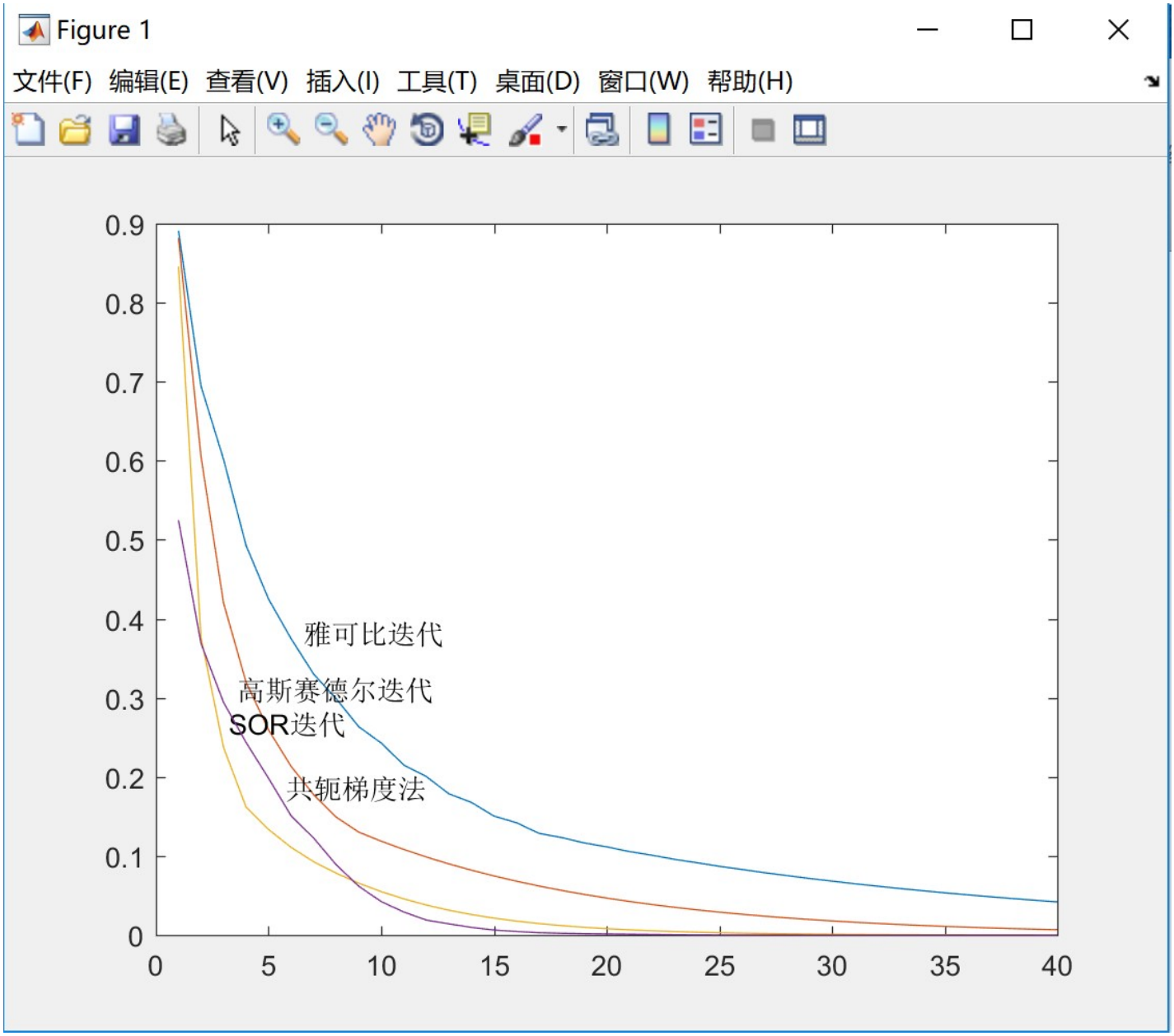
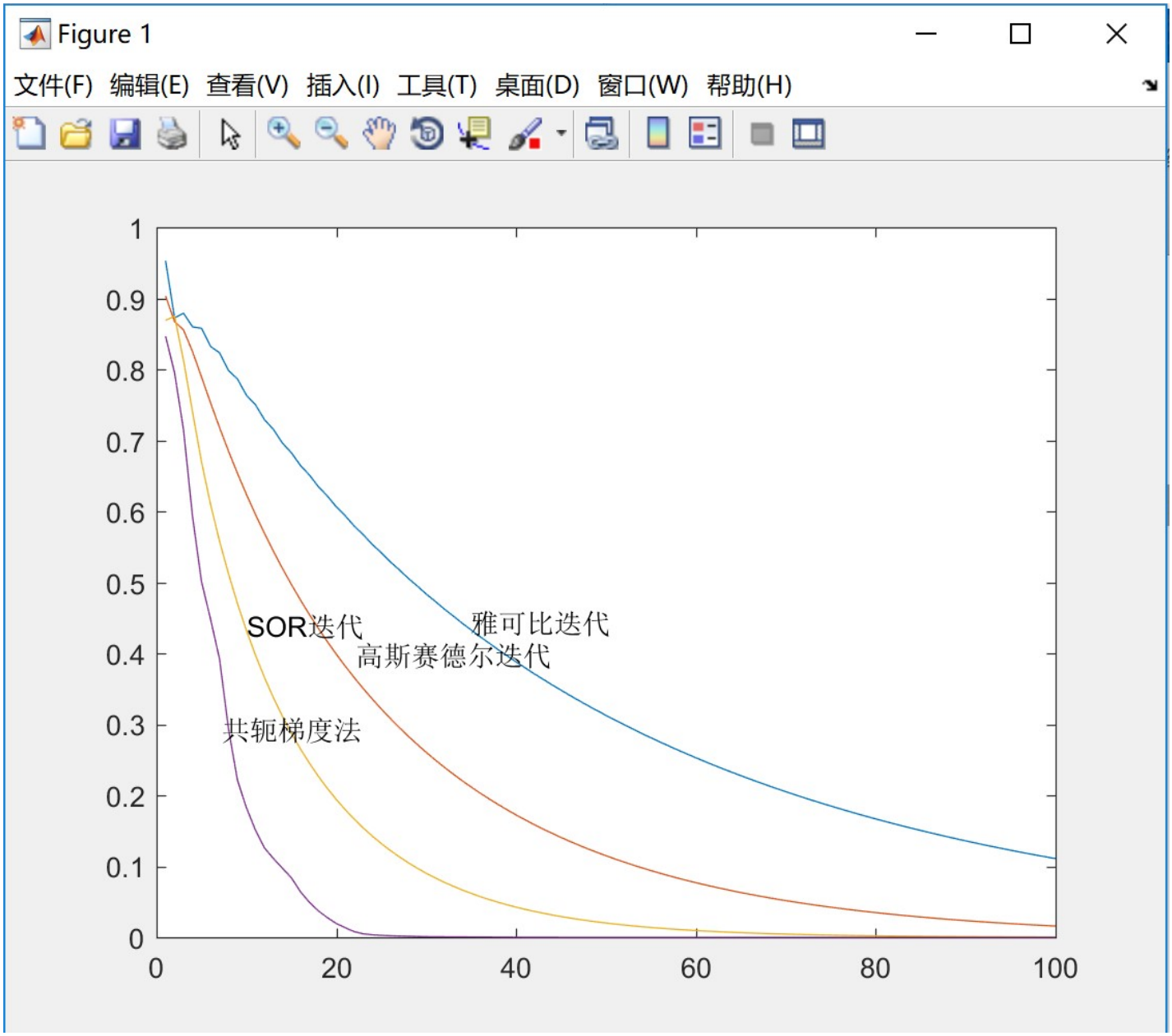


Figure 1
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 5 10 15 20 25 30 35 40
雅可比迭代
高斯赛德尔迭代
SOR迭代
共轭梯度法

n=200




蓝线为雅可比迭代；橙线为高斯赛德尔迭代；黄线为超松弛迭代；紫线为共轭梯度法
从以上四幅图可以很清晰的看出，四种迭代法中，雅可比迭代的收敛速度最慢，共轭梯度的收敛速度最快。高斯赛德尔迭代法和SOR不确定，这是因为SOR迭代法的迭代速度由松弛因子**ω**控制，对于不同矩阵有不同的最佳松弛因子，又因为测试中我设置了固定的松弛因子，所以SOR的收敛速度有时候比高斯赛德尔迭代法快，有时候比高斯赛德尔迭代法慢。

**六个求解方程组算法的时间比较**

对比六种算法对同一个矩阵的求解计算时间：

n = 10

| 方法 | 时间 |
|---|---|
| 雅可比迭代法 | 0.0002 |
| 高斯赛德尔迭代法 | 0.0003 |
| SOR迭代法 | 0.0003 |
| 共轭梯度法 | 0.0001 |
| 高斯消元法 | 0.00013 |
| 列主元消元法 | 0.0008 |

n = 100

| 方法 | 时间 |
|---|---|

| 方法 | 时间 |
| --- | --- |
| 雅可比迭代法 | 0.0053 |
| 高斯赛德尔迭代法 | 0.0044 |
| SOR迭代法 | 0.0048 |
| 共轭梯度法 | 0.0040 |
| 高斯消元法 | 0.0189 |
| 列主元消元法 | 0.0192 |

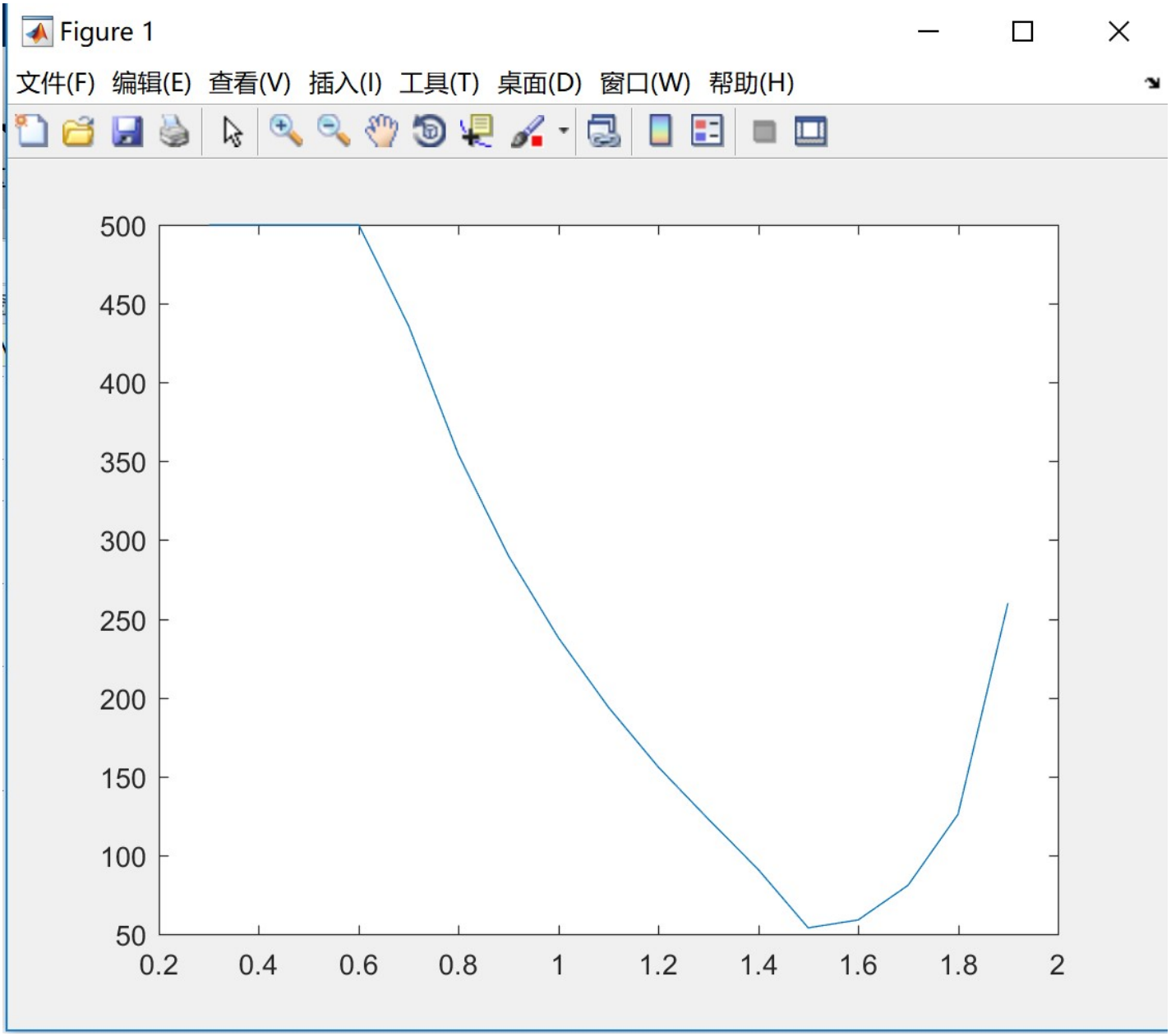
n = 200

| 方法 | 时间 |
| --- | --- |
| 雅可比迭代法 | 0.0254 |
| 高斯赛德尔迭代法 | 0.0128 |
| SOR迭代法 | 0.0120 |
| 共轭梯度法 | 0.0036 |
| 高斯消元法 | 0.1311 |
| 列主元消元法 | 0.1069 |

可以看出，迭代法的计算速度普遍较快，尤其是计算高维矩阵的时候，优势更加明显。其中，共轭梯度法的计算速度最快。

**$\omega$对迭代速度的影响**

这个测试是为了研究不同的$\omega$对SOR的速度的影响。这里采用测量迭代次数的方法，即对于不同omega，看看哪个算法的误差小于某个指定的值，记录当时的迭代次数，对于每次迭代我给了最高的迭代次数500。
结果如下：




从图中可以看出，对于测试用的这个矩阵，最佳的松弛因子$\omega$是1.5。对于不同的矩阵，最佳松弛因子是不同的。

## 内容3

### PageRank介绍

pagerank算法是目前谷歌搜索引擎使用的网页排名算法，它的作用是对现有的各种网站进行一个基于重要度的排序，然后把结果返回给搜索用户。

首先pagerank基于两个基本假设：**数量假设**和**质量假设。**数量假设，是说一个页面的入链越多，它就越重要。质量假设，一个页面的重要性，不仅由入链的数量决定，而且也受入链质量决定。入链的质量越高，这个页面越重要。其中，一个页面的PR值，即前面所说的重要性，为它的入链重要性之和。所以我们就得到了这样的一个模型：**一个页面的得票数由所有链向它的页面的重要性来决定，到一个页面的链接相当于对该页面投了一票。一个页面的PageRank是由所有链向它的页面的重要性经过递归得到的。**

### 大致思路

1. 基于页面之间的指向关系，建立一个连通图，使用矩阵表示，这个矩阵称为**网页链接矩阵。**
2. 根据网页链接矩阵，可以计算出**网页链接概率矩阵**，即对于某一个网页而已，假设它的每一个跳转都是等概率的。每个网页的总权值为1。
3. 由于有的页面它不指向任何其它页面，因此需要增加阻尼系数q，q一般取q=0.85。它的实际意义是在任何时刻，用户从当前页面跳转到另一个页面的概率。
4. 最后利用**马可洛夫性质**通过不断的迭代，直到矩阵收敛，得出来的结果就是PageRank了，即网页排名。

### 伪代码

计算网页链接矩阵

A[i][j] = 1 if page i connect to page j
计算网页链接概率矩阵A

$$A[i][j] = \frac{1}{\sum_{outdegree(page(j))}}$$

if page j connect to page i

while 没有满足精度
$A = (1-q) + q\frac{1}{n}U$,U是元素全为1的n×n的矩阵
最后得到PageRank矩阵A

**优缺点**

PageRank是一个与查询无关的静态算法，所有网页的PageRank值通过离线计算获得；有效减少在线查询时的计算量，极大降低了查询响应时间。但是旧的页面等级会比新页面高。因为即使是非常好的新页面也不会有很多上游链接，除非它是某个上游站点的子站点。