

《数值计算实验》

实验报告

题目	求解线性方程组并用PageRank进行网页排序
姓名	刘硕
学号	16340154
班级	软件工程二班

实验一

一. 实验环境和工具

在OS X High Sierra 10.13.4 环境下利用Matlab-R2018a完成实验。

二. 问题描述

利用高斯消元法和列主元消元法，求解线性方程组 $Ax=b$ ，其中A为 $n \times n$ 维的已知矩阵，b为n维的已知向量，x为n维的未知向量。A与b中的元素服从独立同分布的正态分布。令 $n=10, 50, 100$ ，测试计算时间并绘制曲线。

三. 实验内容

1. 高斯消去法

顺序的高斯消元法按方程和未知量的顺序进行，用逐次消去未知数的方法，把原方程化成上三角矩阵方程组进行求解。求解主要分成两步：**消元过程**用初等行变换将原方程的系数矩阵化成上三角矩阵；**回代过程**用最后一个方程求得的解逐步代入到上面的方程。

1.1. 算法设计

1.1.1. 设计思路

顺序的高斯消元法依从上而下、从左到右的顺序将主对角元下方的元素化为零。这个过程不做行交换，用不为零的一个数乘以某行加至该行以消元。

消元过程

设 主元 $a_{11}^{(1)} \neq 0, a_{22}^{(2)} \neq 0, \dots, a_{nn}^{(n)} \neq 0$
消元过程

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \quad (k=1, 2, \dots, n-1), (i=k+1, k+2, \dots, n)$$

$$(i) - m_{ik}(k) \rightarrow (i)$$

回代过程是在矩阵被消元成为一个上三角矩阵之后，先计算出最后一个变量的值，通过回代，逐次求解其他变量。

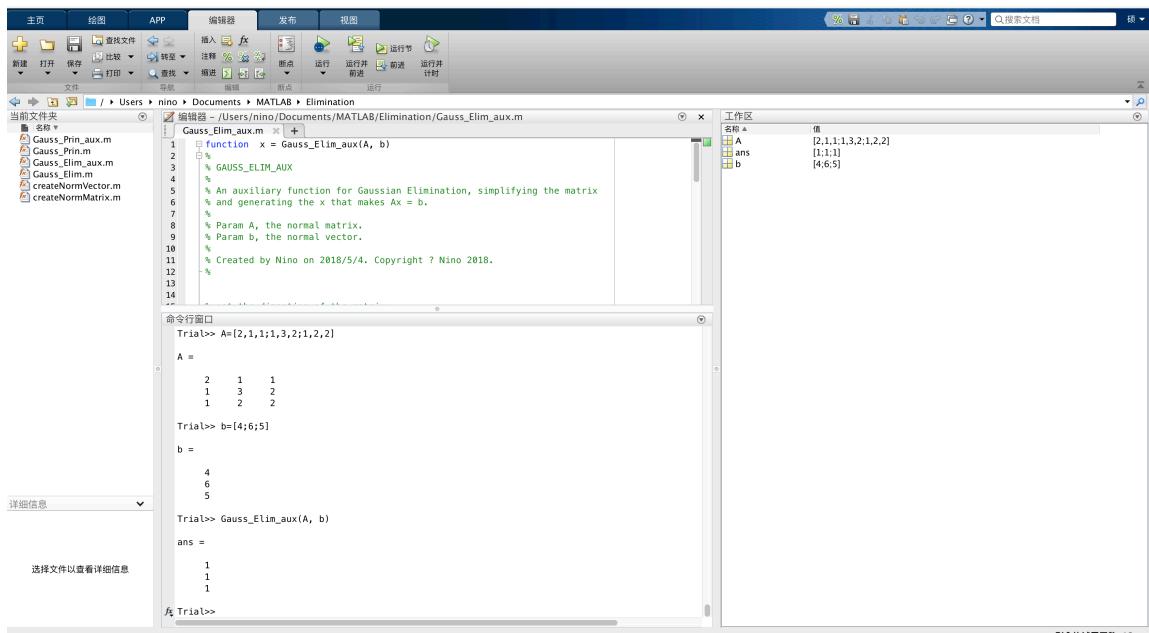
上三角形方程组 $A^{(n)}x = b^{(n)}$ 求解过程

$$\begin{cases} x_n = \frac{b_n^{(n)}}{a_{nn}^{(n)}} \\ x_i = \frac{b_i^{(i)} - \sum_{j=i+1}^n a_{ij}^{(i)} x_j}{a_{ii}^{(i)}} \quad (i=n-1, n-2, \dots, 1) \end{cases}$$

1.1.2. 简单案例

用高斯消去法解下列线性方程组

$$\begin{cases} 2x_1 + x_2 + x_3 = 4 \\ x_1 + 3x_2 + 2x_3 = 6 \\ x_1 + 2x_2 + 2x_3 = 5 \end{cases}$$



得到结果 $x=[1;1;1]$ 。

1.2. 数值实验

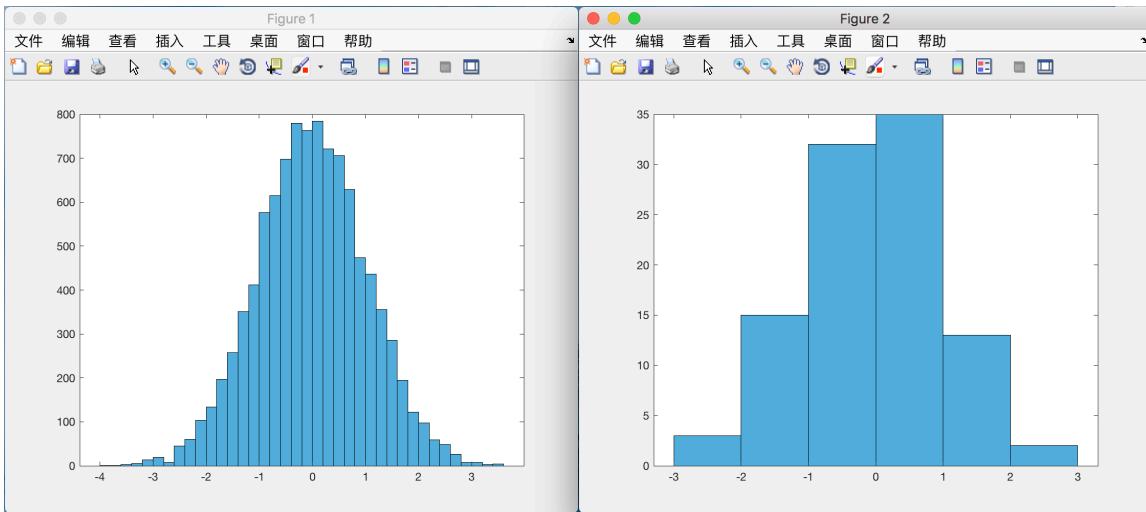
为了探索高斯消元法更广范的应用，我们把A和b设置为独立同分布的标准正态分布。

1.2.1. 实验条件

更普遍的应用需要一些条件制约。当A和b为独立同分布的标准正态分布时，需要让A、b满足以下条件：线性方程组系数矩阵A的顺序主子矩阵非奇异（有可逆矩阵）。当A是方阵的时候，也就是说，所有的主元不为零。

1.2.2. 附加步骤

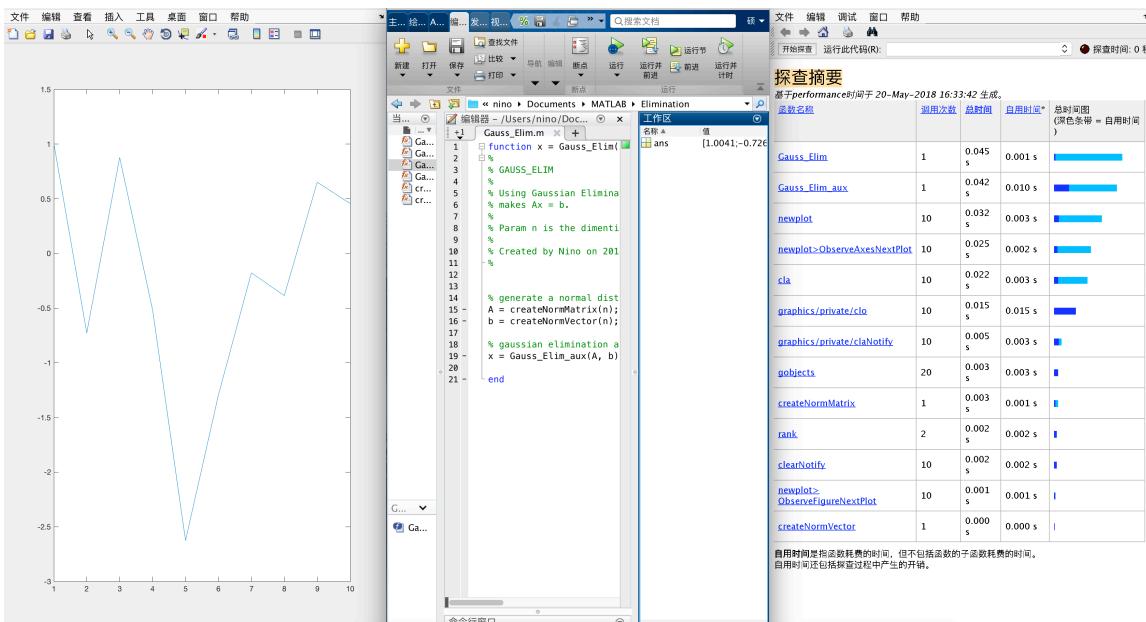
设计两个函数：创建 $n \times n$ 的标准正态分布矩阵和 $n \times 1$ 的标准正态分布向量。将得到的矩阵和向量代入到辅助函数中去。如图为当n为100时候创建出的 $n \times n$ 的标准正态分布矩阵和 $n \times 1$ 的标准正态分布向量。



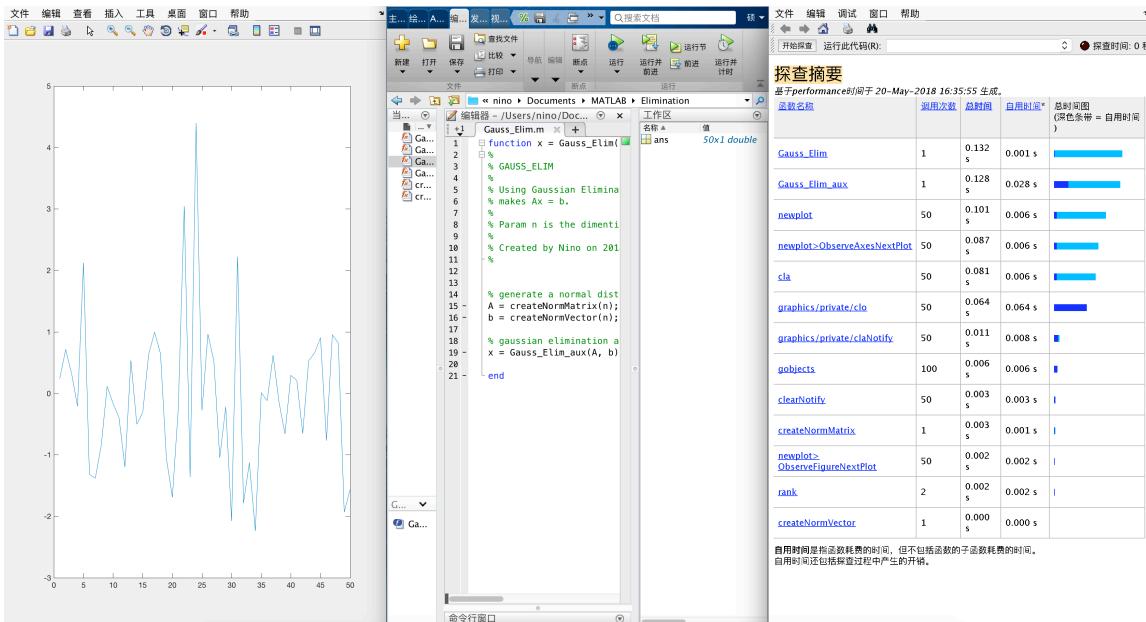
1.3. 结果分析

当n分别为10、50、100时，计算时间和目标向量x的曲线如图所示。

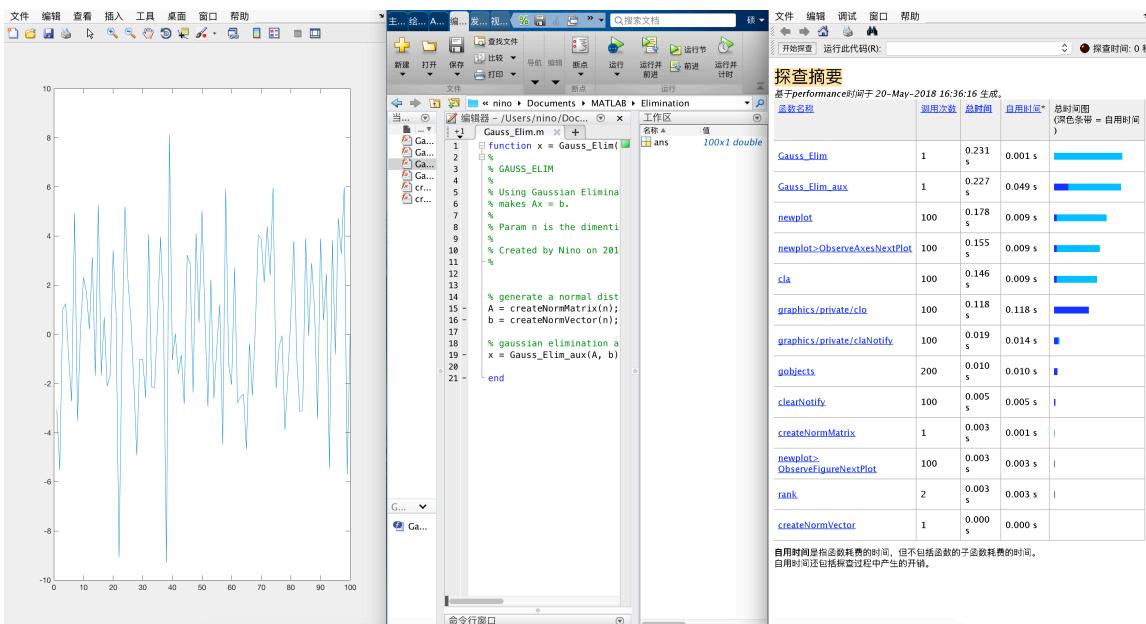
n = 10时



n = 50时



$n = 100$ 时



2. 列主元消元法

列主元消元法不同于顺序的高斯消元法，它避免了用绝对值小的元素做除数，每次消元前都通过变换，使绝对值最大的元素做主元，用这个主元做除数，可以在不增加运算量的情况下减小舍入误差。

2.1. 算法设计

2.1.1. 设计思路

顺序高斯消元法虽然保证了主元不为零，但是当主元绝对值很小的时候，会导致其他元素数量级严重增长，以及较大的舍入误差。列主元消元法与顺序的高斯消元法最大的不同在于，列主元消元法要通过运算找到主元，并通过矩阵变换转化计算问题使数值更加稳定。

当高斯消去法的主元 $a_{kk}^{(k)} = 0$ 时，尽管“当 A 非奇异时， $\det A \neq 0$ ，方程组有唯一解”，也不能实现高斯消去法求解。

例 $A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$ ， A 非奇异， $\det A \neq 0$ ，方程组有

唯一解，但 $a_{11}^{(1)} = 0$ ，不能实现高斯消去法求解。

首先进行的是主元选取和矩阵转化。

经过 $k-1$ 次消元后得到增广矩阵 $(A^{(k)} | b^{(k)})$ ，在此增广矩阵的第 k 列的元素 $a_{kk}^{(k)}, a_{k+1,k}^{(k)}, \dots, a_{nk}^{(k)}$ 中选取绝对值最大的一个，记为 $a_{rk}^{(k)}$ ，然后交换 $(A^{(k)} | b^{(k)})$ 中的第 k 行与第 r 行后，再进行第 k 次消元。

同顺序高斯消元法一样，进行消元。

消元过程

设 主元 $a_{11}^{(0)} \neq 0, a_{22}^{(1)} \neq 0, \dots, a_{nn}^{(n-1)} \neq 0$
消元过程

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \quad (k=1, 2, \dots, n-1), (i=k+1, k+2, \dots, n)$$

$$(i) - m_{ik}(k) \rightarrow (i)$$

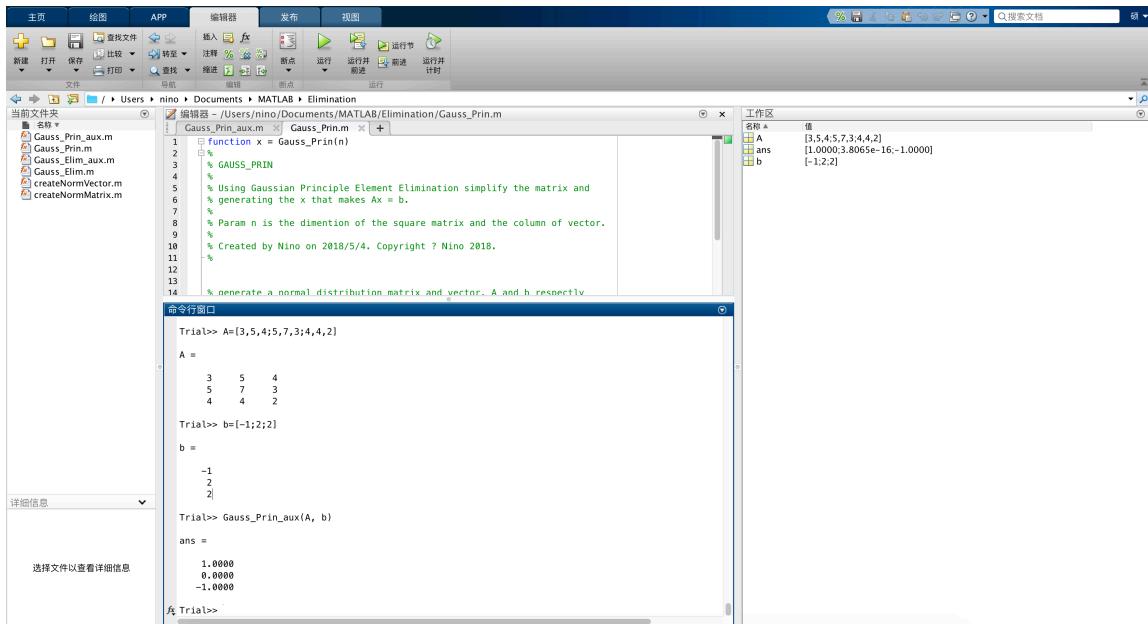
在保证了本次消元后列主元最大后，进行递归。递归函数（辅助函数）的参数是原矩阵的子矩阵和原向量的子向量：原矩阵去掉置换后的第一行和第一列构成子矩阵；原向量去掉置换后的第一行。递归终止条件是传入的矩阵只有一个方程，得到的变量值会被传到上一层函数代入。

回代过程同顺序高斯消元法。利用底层递归得到的变量值，计算本次递归的变量值，回代到上一层递归，逐次求解其他变量。

2.1.2. 简单案例

用列主元高斯消去法求解方程组

$$\begin{cases} 3x_1 + 5x_2 + 4x_3 = 1 \\ 5x_1 + 7x_2 + 3x_3 = 2 \\ 4x_1 + 4x_2 + 2x_3 = 2 \end{cases}$$



得到结果 $x=[1;0;-1]$ 。

2.2. 数值实验

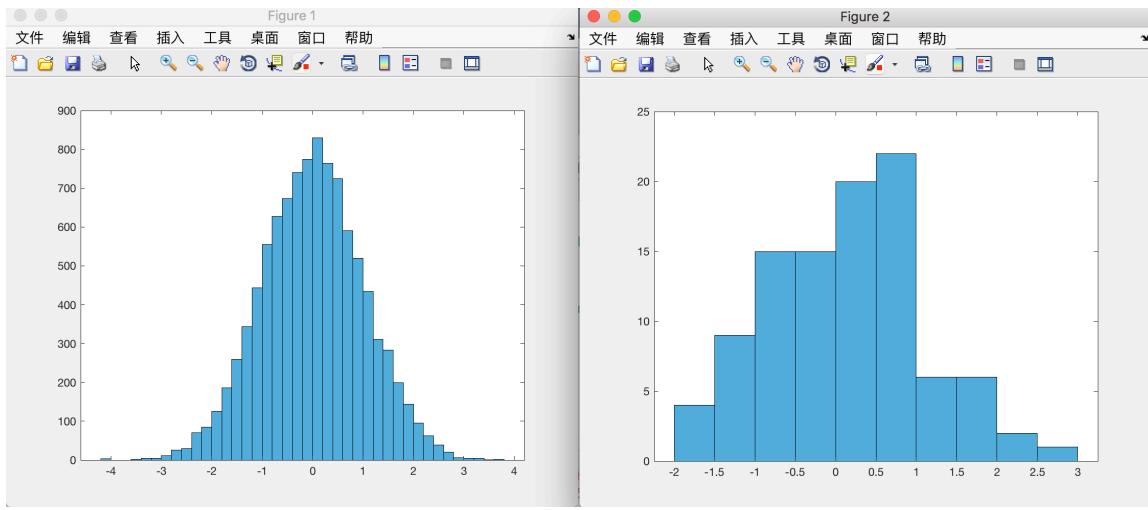
为了探索列主元高斯消元法更广范的应用，我们把A和b设置为独立同分布的标准正态分布。

2.2.1. 实验条件

更普遍的应用需要一些条件制约。当A和b为独立同分布的标准正态分布时，需要让A、b满足以下条件：线性方程组系数矩阵A的顺序主子矩阵非奇异（有可逆矩阵）。当A是方阵的时候，也就是说，所有的主元不为零。

2.2.2. 附加步骤

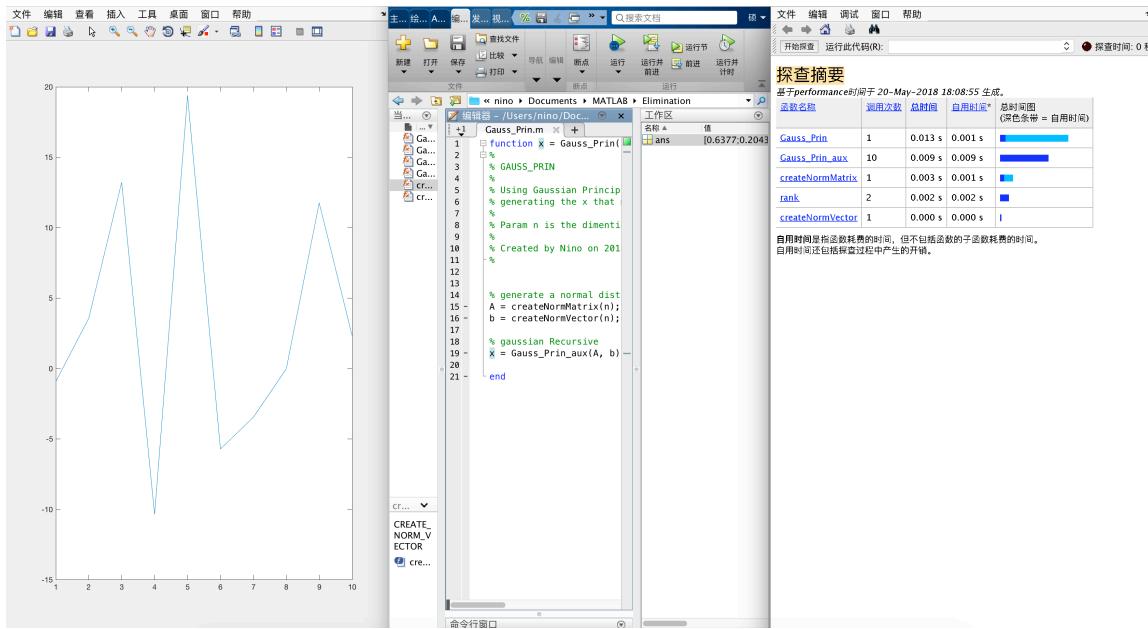
设计两个函数：创建 $n \times n$ 的标准正态分布矩阵和 $n \times 1$ 的标准正态分布向量。将得到的矩阵和向量代入到辅助函数中去。如图为当n为100时候创建出的 $n \times n$ 的标准正态分布矩阵和 $n \times 1$ 的标准正态分布向量。



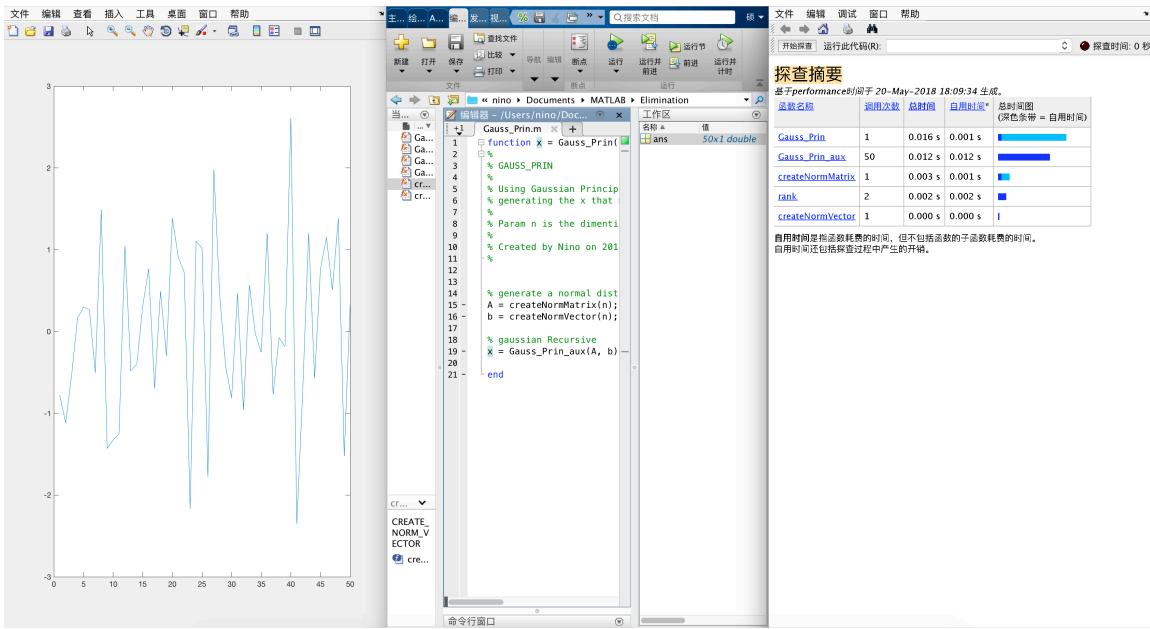
2.3. 结果分析

当n分别为10、50、100时，计算时间和目标向量x的曲线如图所示。

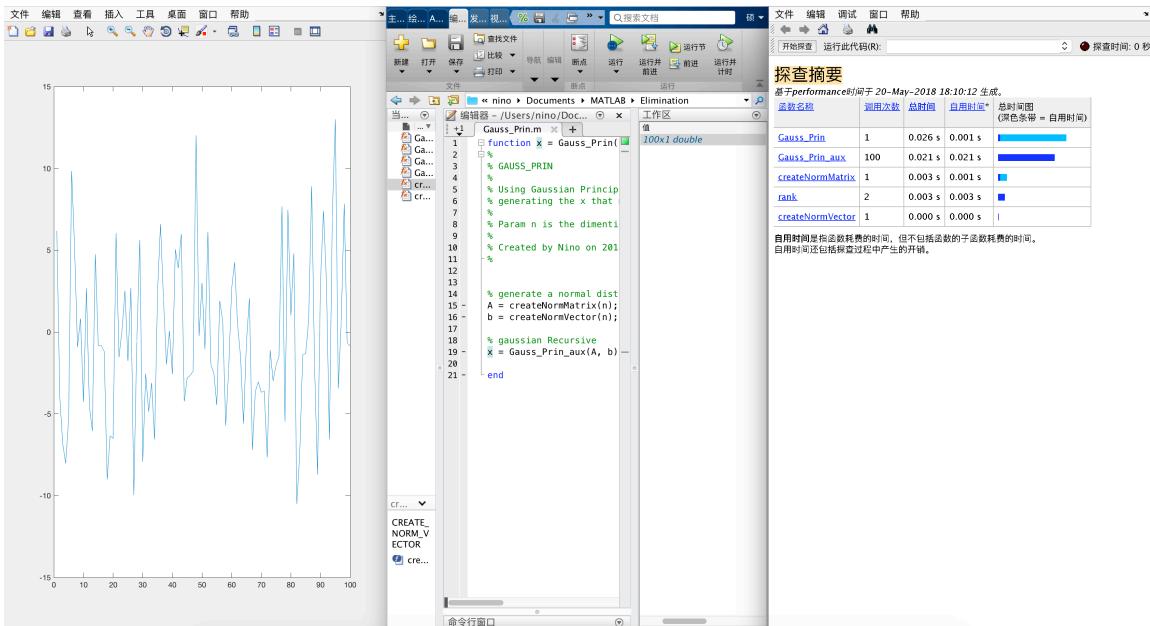
n = 10时



n = 50时



$n = 100$ 时



四. 实验心得

在这次实验中，我主要是熟悉了一下Matlab环境配置和相关语法。让我感悟最深的一点是，许多可以运行的样例在大数据的情况下不能运行。更广泛的独立同分布的正态分布中运行中发现的许多问题可以帮助我了解数值计算在计算时候的应用以及算法的应用限制条件。之前轻而易举解决的问题在大量运算中可能不适用，以后在设计算法的时候需要考虑其复杂度和误差影响。

高斯消元法较为简单，重点是消元步骤和迭代步骤。列主元法的重点是矩阵变换和相应的向量变换

消元递归和代入递归，其中矩阵变换之后对之后代入求得的变量有位置影响，很容易搞混，是这个问题的难点。

实验二

一. 实验环境和工具

在OS X High Sierra 10.13.4 环境下利用Matlab-R2018a完成实验。

二. 问题描述

分别利用Jacobi迭代法、Gauss-Seidel迭代法、逐次超松弛迭代法、共轭梯度法，求解线性方程组 $Ax=b$ ，其中A为 $n \times n$ 维的已知矩阵，b为n维的已知向量，x为n维的未知向量。A与b中的元素服从独立同分布的正态分布。令n=10、50、100，分别绘制算法的收敛曲线，横坐标为迭代步数，纵坐标为相对误差。并比较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元法、列主元消元法的计算时间。改变逐次超松弛法的松弛因子，分析其对收敛速度的影响。

三. 实验内容

1. Jacobi迭代法

顺序的高斯消元法和基于其改进的列主元消元法在处理低阶稠密矩阵的时候有良好的效果，但是当矩阵的阶n很大，零元素较多的大型方程组，利用迭代法解方程更加合适。迭代法通常适用于解零元素较多的大型方程（稀疏矩阵）。雅可比迭代法的核心思想是把方程化成一个 $x = Bx + f$ 的形式，并利用迭代使x逐渐收敛，找到x的值。

1.1. 算法设计

1.1.1. 设计思路

步骤 1. 给定初始值 $x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}$, 精度 ϵ , 最大容许迭代次数 M , 令 $k=1$ 。

步骤 2. 对 $i=1, 2, \dots, n$ 依次计算

$$x_i = (b_j - \sum_{\substack{j=1 \\ j \neq i}}^n a_{ij} x_j) / a_{ii} \quad (a_{ii} \neq 0, i=1, 2, \dots, n)$$

$$\epsilon_i = |x_i^{(1)} - x_i^{(0)}|$$

$$x_i^{(1)} \rightarrow x_i^{(0)}$$

步骤 3. 求出 $\epsilon = \max_{1 \leq i \leq n} \{ \epsilon_i \}$, 若 $\epsilon < \epsilon$, 则输出结果 $x_i^{(0)} (i=1, 2, \dots, n)$, 停止计算。

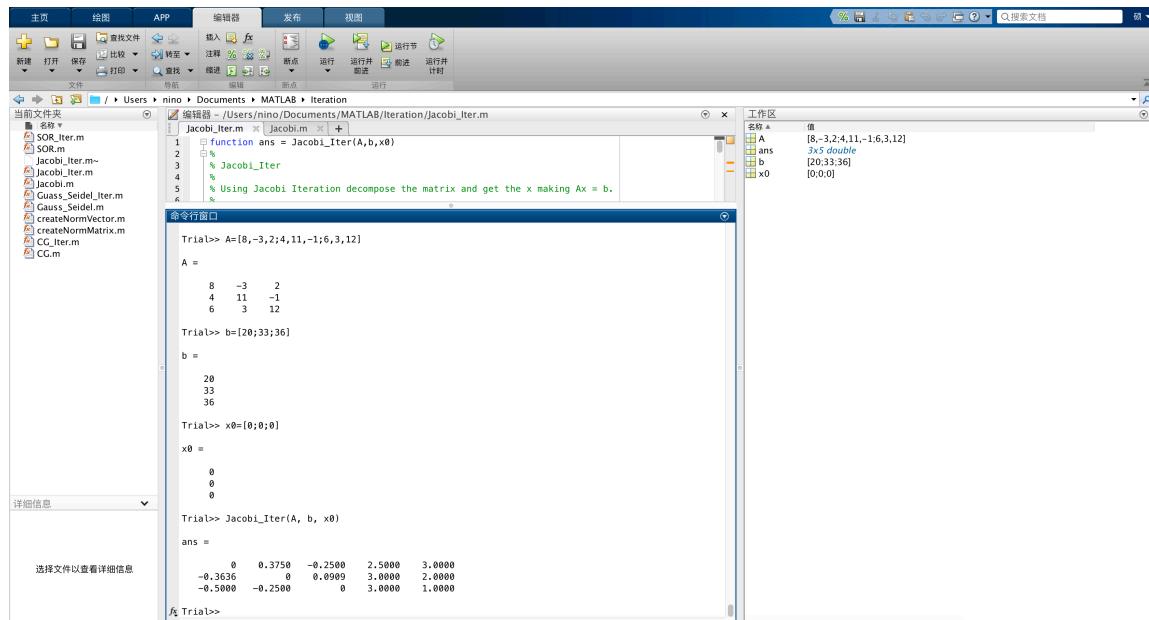
否则执行步骤 4.

步骤 4. 若 $k < M$, $k+1 \rightarrow k$, 转步骤 2 继续迭代。若 $k \geq M$, 表明迭代失败, 停止计算。

1.1.2. 简单案例

求解方程

$$\begin{cases} 8x_1 - 3x_2 + 2x_3 = 20 \\ 4x_1 + 11x_2 - x_3 = 33 \\ 6x_1 + 3x_2 + 12x_3 = 36 \end{cases}$$



得到结果 $ans = [B; f; x]$ 。其中 B 和 f 分别是迭代方程 $x = Bx + f$ 的矩阵和向量； x 是迭代结果。

1.2. 数值实验

为了探索 Gauss-Seidel 迭代法更广范的应用，我们把 A 和 b 分别设置为独立同分布的标准正态分布的对角占优矩阵和独立同分布的标准正态分布的向量。

1.2.1. 实验条件

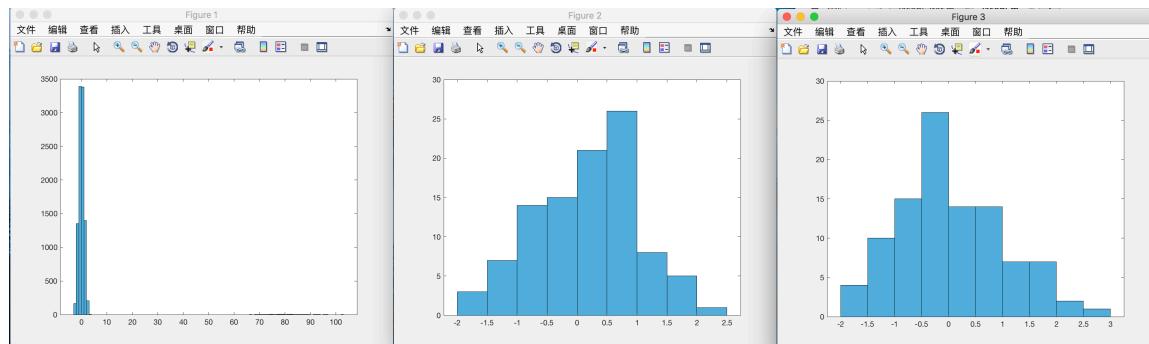
更普遍的应用需要一些条件制约。同高斯消元法和列主元消元法一样，当A和b为独立同分布的标准正态分布时，需要让A、b满足以下条件：线性方程组系数矩阵A的顺序主子矩阵非奇异（有可逆矩阵）。当A是方阵的时候，也就是说，所有的主元不为零。

不同于前面的方法，雅可比迭代法还需要让创造的正态矩阵和正态向量是对角占优矩阵以保证收敛——实验需要。为了构造这种矩阵，矩阵的对角元素的绝对值需要大于其所在行所有元素绝对值的和。

为了得到结果更加直观的显示，在辅助函数的返回值设置为每次迭代完的x和迭代次数。把这个返回值带入到主函数，得到 1^*iter 的mistake向量，每一列的值分别为相对误差。用点阵图表示误差向量。

1.2.2. 附加步骤

设计两个函数：创建 $n*n$ 的标准正态分布矩阵和 $n*1$ 的标准正态分布向量。将得到的矩阵和向量代入到迭代函数中去。除此之外，为了保证结果收敛，还要保证对角占优。如图为当n为100时候创建出的 $n*n$ 的标准正态分布矩阵和 $n*1$ 的标准正态分布b向量和 x_0 迭代初始向量。



通过图像可以发现， $n*n$ 的标准正态分布矩阵不同于之前的，虽然大部分元素大致满足正态分布，有一些极端值。这些极端值是对角占优的正态矩阵的对角元素。另外 $n*1$ 的标准正态分布b向量和 x_0 迭代初始向量虽然二者都大体上满足正态分布，但是可以看得出，两者的图像并不完全相同。

1.3. 结果分析

当n分别为10、50、100时，相对误差和目标向量x的曲线如图所示。

$n = 10$ 时

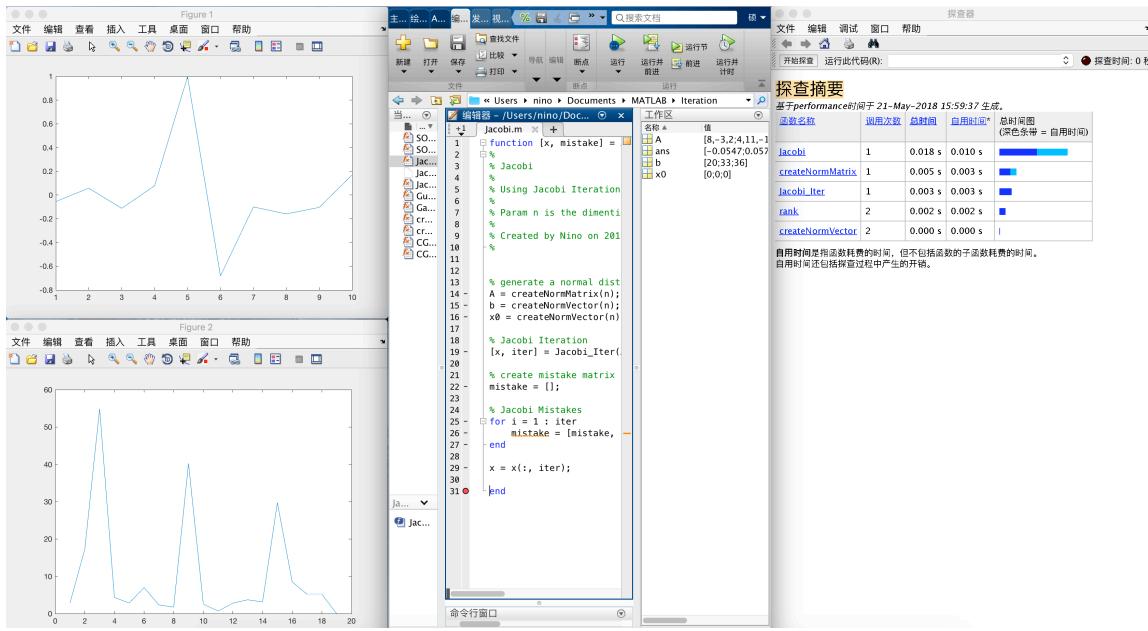


figure1是 x 的最终迭代结果；figure2是每次迭代对应的误差矩阵。通过观察，可以发现，一共迭代了19次，误差结果并不递减，但逐渐收敛。总用时0.018秒。

$n = 50$ 时

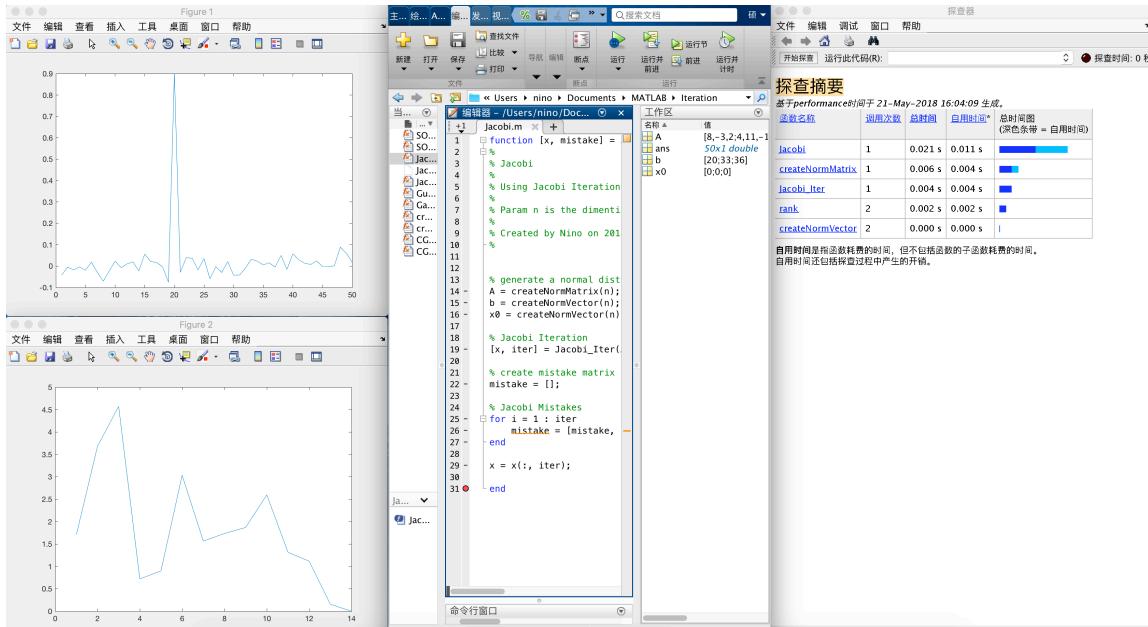


figure1是 x 的最终迭代结果；figure2是每次迭代对应的误差矩阵。通过观察，可以发现， x 的值每一项基本接近零；一共迭代了14次，误差结果并不递减，但逐渐收敛。总用时0.021 秒。在 $x(20)$ 左右的位置会出现一个偏离较大的量。

$n = 100$ 时

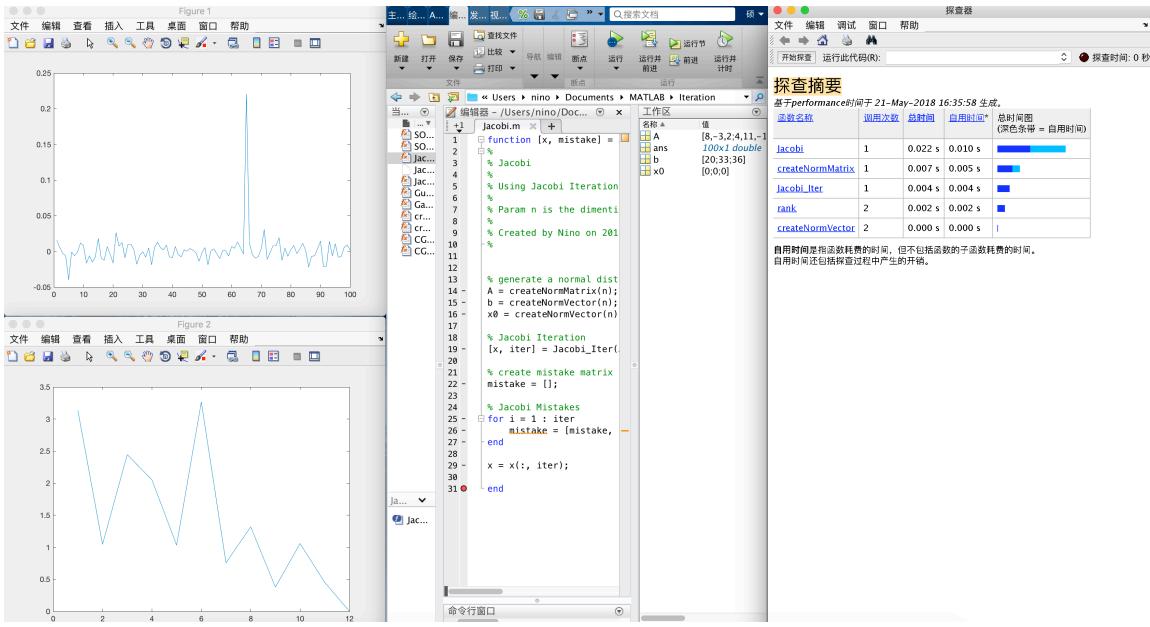


figure1是x的最终迭代结果； figure2是每次迭代对应的误差矩阵。通过观察，可以发现，x的值每一项基本接近零；一共迭代了12次，误差结果并不递减，但逐渐收敛。总用时0.022 秒。在x(65)左右的位置会出现一个偏离较大的量。似乎可以发现随着n增大，这种方法的用时和迭代次数并没有大量增长。

2. Gauss-Seidel迭代法

Gauss-Seidel迭代法是基于Jacobi迭代法的改进，比起Jacobi有更好的性能。

在Jacobi迭代中，计算次序是 $x_1^{(k+1)} \rightarrow x_2^{(k+1)} \rightarrow x_3^{(k+1)}$ 但在计算 $x_2^{(k+1)}$ 时没有利用已经计算出的 $x_1^{(k+1)}$ 而仍利用 $x_1^{(k)}$ ，同样，在计算 $x_3^{(k+1)}$ 时也没有利用最新计算出 $x_1^{(k+1)}$ 和 $x_2^{(k+1)}$ 。

同样，Guass-Seidel也是在工程中计算稀疏矩阵的时候较为常用，因为迭代法通常可以利用矩阵中有大量零元素的特点。

2.1. 算法设计

2.1.1. 设计思路

迭代过程：

有方程组 $\mathbf{Ax} = \mathbf{b}$, 其中:

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

$$A = L_* + U$$

$$L_* = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

则这个方程组可以变形为:

$$L_* \mathbf{x} = \mathbf{b} - U \mathbf{x}$$

由此建立迭代公式:

$$\mathbf{x}^{(k+1)} = L_*^{-1}(\mathbf{b} - U \mathbf{x}^{(k)})$$

这个公式还可以继续展开成这样:

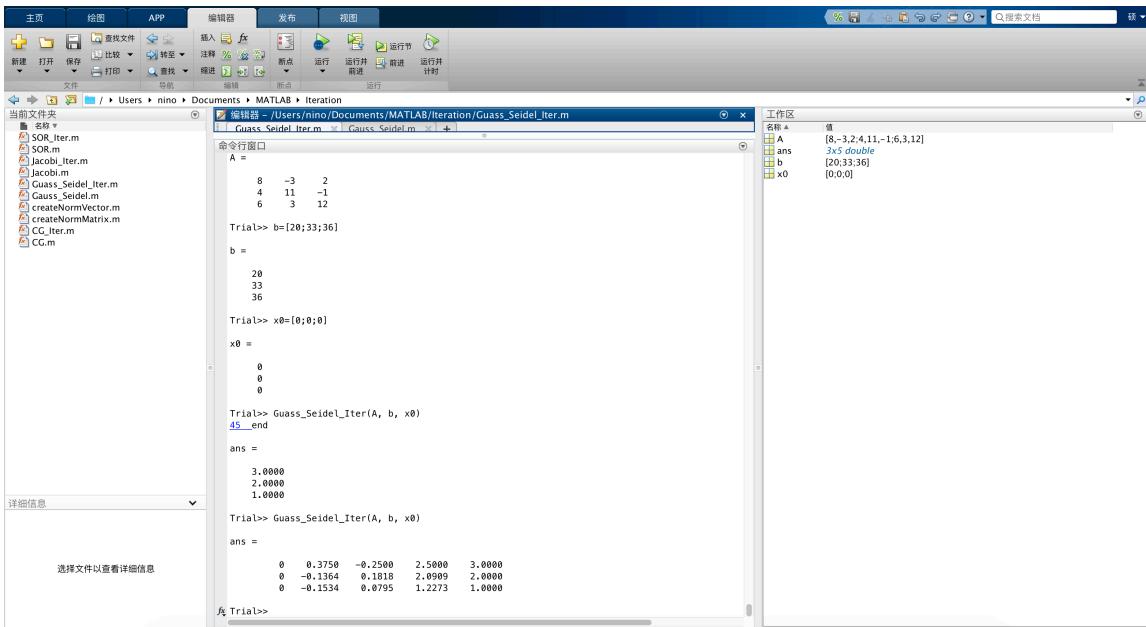
$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \dots, n.$$

展开方法是前向替换 (forward substitution)

2.1.2. 简单案例

求解方程

$$\begin{cases} 8x_1 - 3x_2 + 2x_3 = 20 \\ 4x_1 + 11x_2 - x_3 = 33 \\ 6x_1 + 3x_2 + 12x_3 = 36 \end{cases}$$



如果为通过迭代得到的结果，其中 $\text{ans} = [G, f, x]$: G和f分别是迭代过程 $X=Gx+f$ 中的矩阵和向量，x是最后的迭代结果。

2.2. 数值实验

为了探索Gauss-Seidel迭代法更广范的应用，我们把A和b分别设置为独立同分布的标准正态分布的对角占优矩阵和独立同分布的标准正态分布的向量。

2.2.1. 实验条件

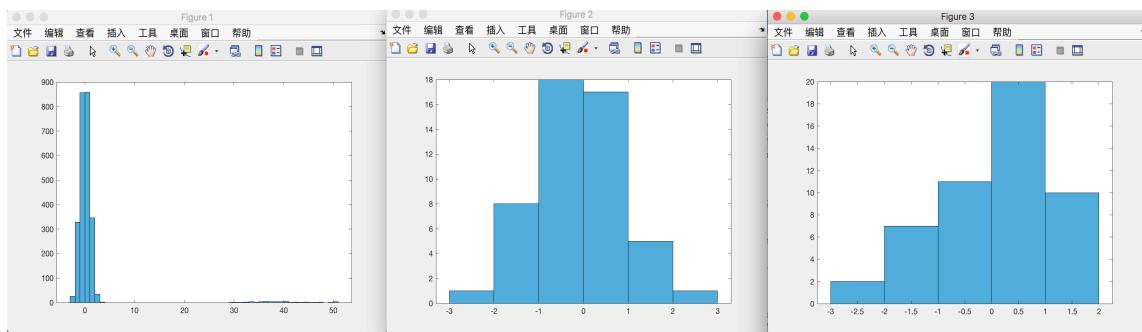
更普遍的应用需要一些条件制约。同高斯消元法和列主元消元法一样，当A和b为独立同分布的标准正态分布时，需要让A、b满足以下条件：线性方程组系数矩阵A的顺序主子矩阵非奇异（有可逆矩阵）。当A是方阵的时候，也就是说，所有的主元不为零。

不同于前面的方法，雅可比迭代法还需要让创造的正态矩阵和正态向量是对角占优矩阵以保证收敛——实验需要。为了构造这种矩阵，矩阵的对角元素的绝对值需要大于其所在行所有元素绝对值的和。

为了得到结果更加直观的显示，在辅助函数的返回值设置为每次迭代完的x和迭代次数。把这个返回值带入到主函数，得到 1^*iter 的mistake向量，每一列的值分别为相对误差。用点阵图表示误差向量。

2.2.2. 附加步骤

设计两个函数：创建 $n*n$ 的标准正态分布矩阵和 $n*1$ 的标准正态分布向量。将得到的矩阵和向量代入到辅助函数中去。如图为当n为100时候创建出的 $n*n$ 的标准正态分布矩阵和 $n*1$ 的标准正态分布向量。



通过图像可以发现， $n \times n$ 的标准正态分布矩阵不同于之前的，虽然大部分元素大致满足正态分布，有一些极端值。这些极端值是对角占优的正态矩阵的对角元素。另外 $n \times 1$ 的标准正态分布 b 向量和 x_0 迭代初始向量虽然二者都大体上满足正态分布，但是可以看得出，两者的图像并不完全相同。

2.3. 结果分析

当 n 分别为10、50、100时，计算时间和目标向量 x 的曲线如图所示。

$n = 10$ 时

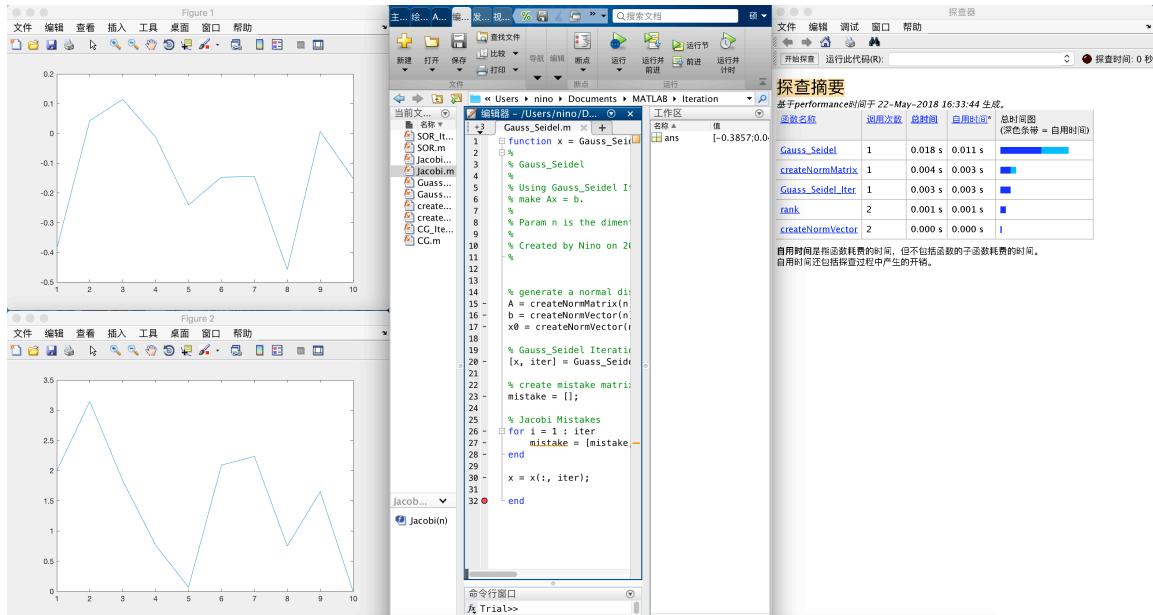


figure1是 x 的最终迭代结果；figure2是每次迭代对应的误差矩阵。通过观察，可以发现， x 的值每一项基本接近零；一共迭代了10次，误差结果并不递减，但逐渐收敛。总用时0.018秒。在第五次迭代会出现一个较准确量。

$n = 50$ 时

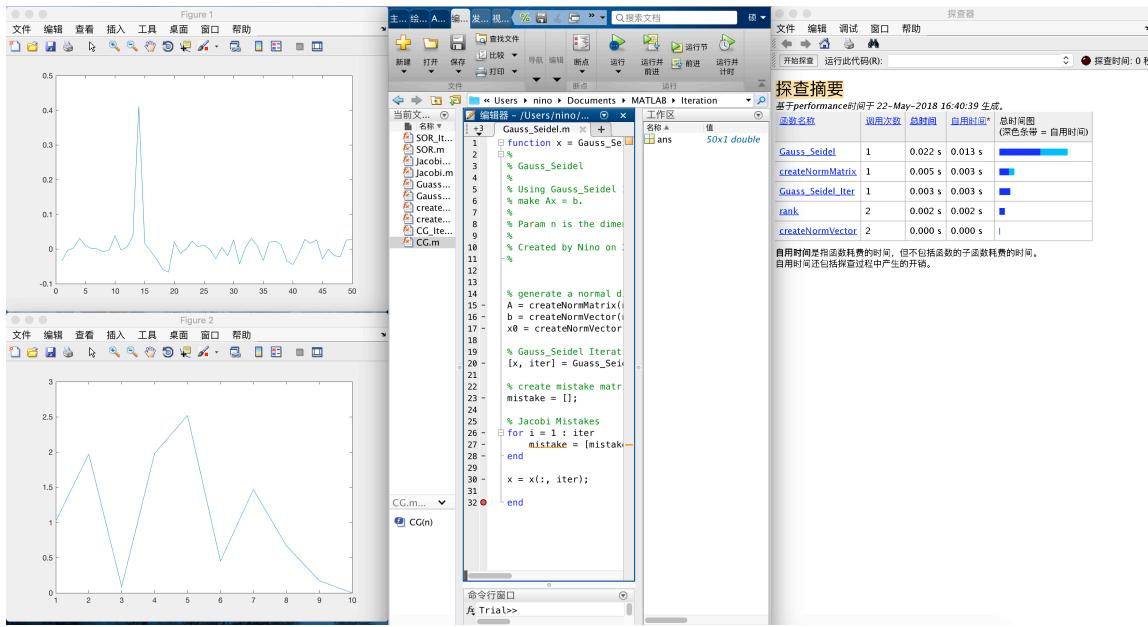


figure1是x的最终迭代结果； figure2是每次迭代对应的误差矩阵。通过观察，可以发现，x的值每一项基本接近零，在第15项左右出现了一个极端值；一共迭代了10次，误差结果并不递减，但逐渐收敛。总用时0.022秒。在第三次迭代会出现一个较准确量。

$n = 100$ 时

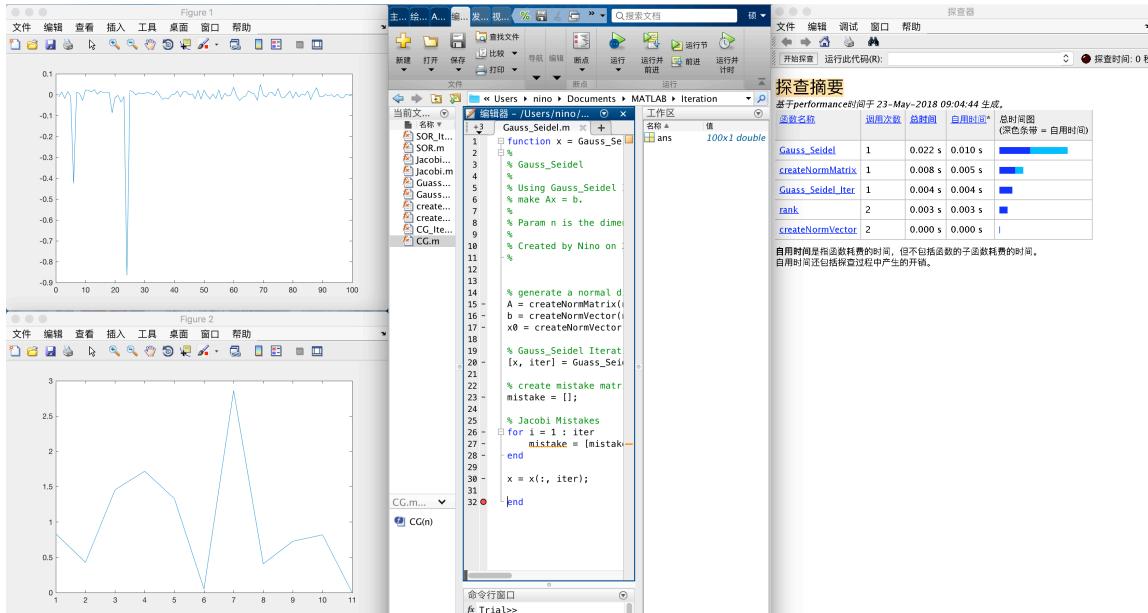


figure1是x的最终迭代结果； figure2是每次迭代对应的误差矩阵。通过观察，可以发现，x的值每一项基本接近零，在第5项和第25项左右出现了极端值；一共迭代了11次，误差结果并不递减，但逐渐收敛。总用时0.022秒，比起 $n=50$ 并无太大变化。在第六次迭代会出现一个较准确量。

3. 逐次超松弛迭代法

逐次超松弛迭代法是在**Gauss-Seidel**迭代法的基础上为了提升速度，采用加权平均的新算法。**Jacobi**迭代法和**Gauss-Seidel**迭代法都是对原矩阵做一个根本变换，破坏了变换前后方程的等价性。逐次超松弛迭代法的思路为：如果能够简单有效地确定单个样本加入样本集后对训练结果的影响，一方面，出现新的样本时可以利用原来的一训练结果而不必重新开始；另一方面，让训练样本逐个进入样本集可以简化寻优过程，提高算法速度。这实际上是将样本集中的样本数减少到一个。对于逐次超松弛迭代法，松弛因子的选取对算法的收敛速度有很大影响，通常对于方程组 $Ax = b$ ，若A为正定矩阵，则当 $0 < \omega < 2$ 时，逐次超松弛迭代恒收敛。

3.1. 算法设计

3.1.1. 设计思路

设已求得n元线性代数方程组 $Ax=b$ 第 $k-1$ 次迭代向量 $x^{(k-1)} = (x_1^{(k-1)}, x_2^{(k-1)}, \dots, x_n^{(k-1)})^T$ 及第 k 次迭代向量 $x^{(k)}$ 的分量 $x_j^{(k)} (j = 1, 2, \dots, i-1)$ ，要计算分量 $x_i^{(k)}$ 。

$$(1) \text{ 用Gauss-Seidel(GS)迭代求得: } x'_i = \frac{1}{a_{ii}} \left(- \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^n a_{ij} x_j^{(k-1)} + b_i \right);$$

$$(2) \text{ 计算 } x'_i \text{ 与第 } k-1 \text{ 次迭代值 } x_i^{(k-1)} \text{ 的加权平均 } \omega \text{ 作为第 } k \text{ 次迭代值: } x_i^{(k)} = (1-\omega)x_i^{(k-1)} + \omega x'_i$$

$$\text{或整理成: } x_i^{(k)} = x_i^{(k-1)} + \frac{\omega \left(- \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^n a_{ij} x_j^{(k-1)} + b_i \right)}{a_{ii}}, \quad i = (1, 2, \dots, n).$$

其中，参数 ω 称为松弛因子， $0 < \omega < 2$ 。当 $\omega > 1$ 时，上式称为逐次超松弛迭代法；当 $\omega = 1$ 时，上式为**Gauss-Seidel**迭代法；当 $0 < \omega < 1$ 时，上式称为低松弛迭代法。[\[2\]](#)

收敛性判别条件

SOR迭代法收敛的充分必要条件是 $\rho(\lambda\omega) < 1$ ， $\rho(\lambda\omega)$ 与松弛因子 ω 有关。 $\rho(\lambda\omega)$ 与 ω 的关系以及SOR方法收敛的条件有如下定理。

定理1: (Kahan)对任意的 $A \in R^{n \times n}$ ，设其对角元皆非零，则对所有实数 ω ，有： $\rho(\lambda\omega) \geq \omega - 1$ 。

推论：如果解 $Ax=b$ 的SOR方法收敛，则有 $\omega-1 < 1$ ，即 $0 < \omega < 2$ 。

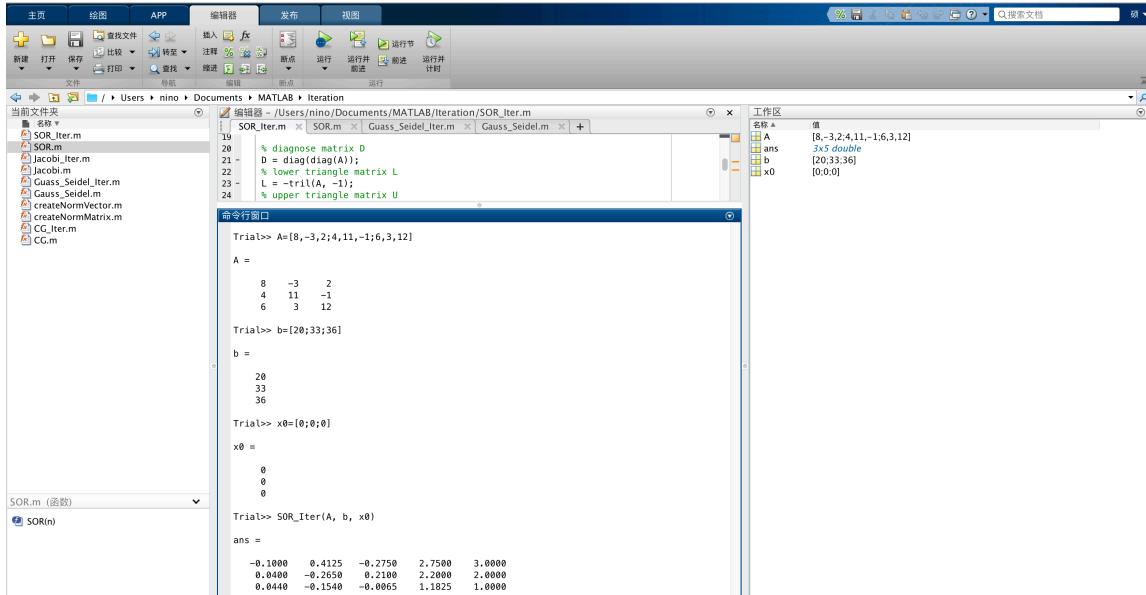
定理2: (Ostrowski-Reich)设 $A \in R^{n \times n}$ ， A 对称正定，且 $0 < \omega < 2$ ，则解 $Ax=b$ 的SOR方法收敛。

逐次超松弛迭代法中松弛因子的取值直接影响到算法的收敛性及收敛速度。若选择得当，可以加快收敛速度，甚至可以使发散的迭代变成收敛。因此，松弛因子取值的选取是逐次超松弛迭代法能否成功的关键。为了保证迭代过程的收敛，必须要求 $0 < \omega < 2$ ，对超松弛法去 $1 < \omega < 2$ 。只有在系数矩阵具有少数特殊类型的情况下，才能通过数学公式确定松弛因子。对于一般情况，有如二分比较法、逐步搜索法、黄金分割法。

3.1.2. 简单案例

求解方程

$$\begin{cases} 8x_1 - 3x_2 + 2x_3 = 20 \\ 4x_1 + 11x_2 - x_3 = 33 \\ 6x_1 + 3x_2 + 12x_3 = 36 \end{cases}$$



得到结果 $\text{ans} = [\mathbf{B}; \mathbf{f}; \mathbf{x}]$ 。其中 \mathbf{B} 和 \mathbf{f} 分别是迭代方程 $\mathbf{x} = \mathbf{B}\mathbf{x} + \mathbf{f}$ 的矩阵和向量； \mathbf{x} 是迭代结果。

3.2. 数值实验

为了探索逐次超松弛迭代法更广范的应用，我们把 \mathbf{A} 和 \mathbf{b} 分别设置为独立同分布的标准正态分布的对角占优矩阵和独立同分布的标准正态分布的向量。除此之外，我把逐次超松弛的因子暂且设为 1.1，进行实验。

3.2.1. 实验条件

更普遍的应用需要一些条件制约。同高斯消元法和列主元消元法一样，当 \mathbf{A} 和 \mathbf{b} 为独立同分布的标准正态分布时，需要让 \mathbf{A} 、 \mathbf{b} 满足以下条件：线性方程组系数矩阵 \mathbf{A} 的顺序主子矩阵非奇异（有可逆矩阵）。当 \mathbf{A} 是方阵的时候，也就是说，所有的主元不为零。

不同于前面的方法，雅可比迭代法还需要让创造的正态矩阵和正态向量是对角占优矩阵以保证收敛——实验需要。为了构造这种矩阵，矩阵的对角元素的绝对值需要大于其所在行所有元素绝对值的和。

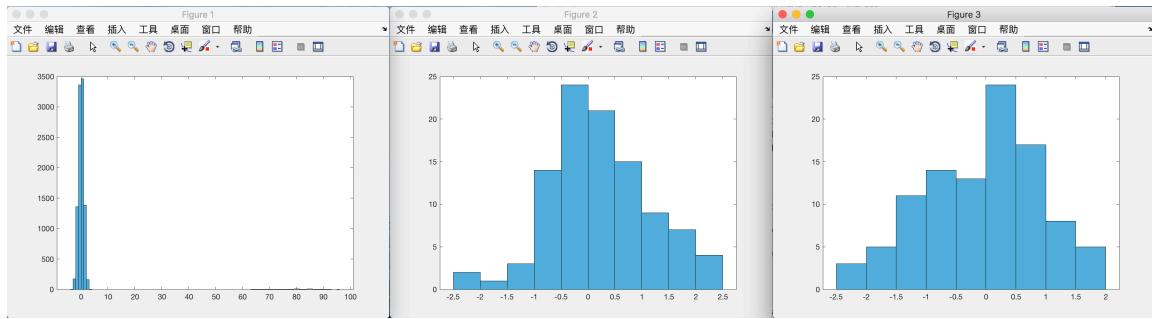
为了得到结果更加直观的显示，在辅助函数的返回值设置为每次迭代完的 \mathbf{x} 和迭代次数。把这个返回值带入到主函数，得到 1^*iter 的 mistake 向量，每一列的值分别为相对误差。用点阵图表示误差向量。

为了探索逐次超松弛因子对收敛速度的影响，改变逐次超松弛因子的大小，绘制对于同一组数据，

收敛次数与收敛因子之间的关系，找到该数据对应的最合适因子。

3.2.2. 附加步骤

设计两个函数：创建 $n \times n$ 的标准正态分布矩阵和 $n \times 1$ 的标准正态分布向量。将得到的矩阵和向量代入到迭代函数中去。除此之外，为了保证结果收敛，还要保证对角占优。如图为当 $n=100$ 时候创建出的 $n \times n$ 的标准正态分布矩阵和 $n \times 1$ 的标准正态分布 b 向量和 x_0 迭代初始向量。



通过图像可以发现， $n \times n$ 的标准正态分布矩阵不同于之前的，虽然大部分元素大致满足正态分布，有一些极端值。这些极端值是对角占优的正态矩阵的对角元素。另外 $n \times 1$ 的标准正态分布 b 向量和 x_0 迭代初始向量虽然二者都大体上满足正态分布，但是可以看得出，两者的图像并不完全相同。

改变逐次超松弛因子的大小，从0.8开始，每次增大逐次超松弛因子0.05到1.2（容易收敛的范围），绘制对于同一组数据，收敛次数与收敛因子之间的关系。

3.3. 结果分析

当 n 分别为10、50、100时，相对误差和目标向量 x 的曲线如图所示。

$n = 10$ 时

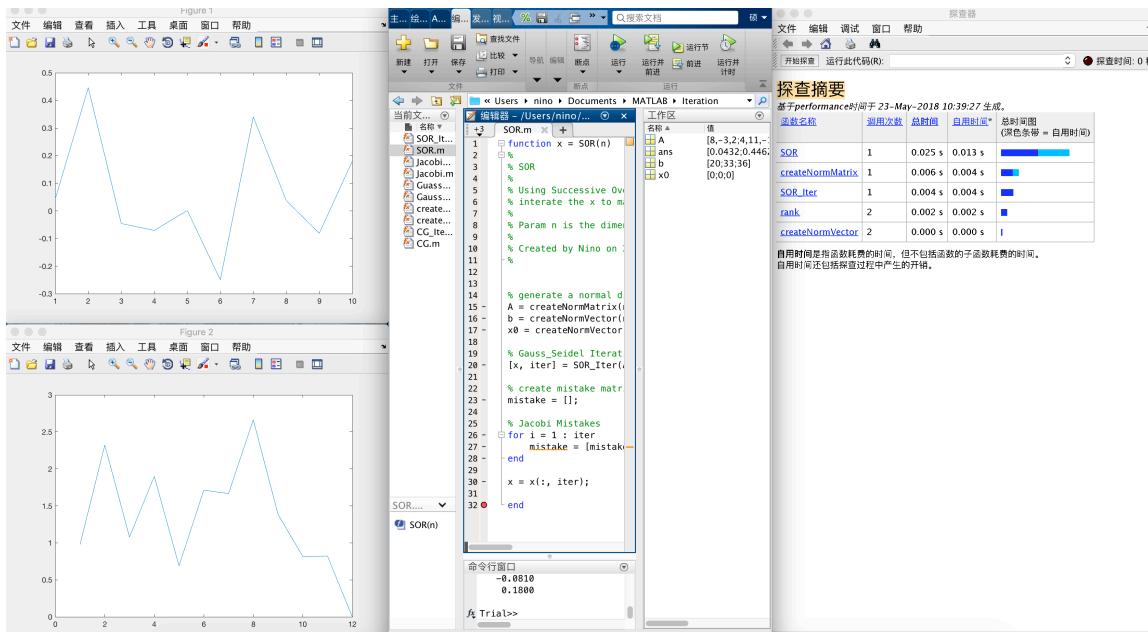


figure1是 x 的最终迭代结果；figure2是每次迭代对应的误差矩阵。通过观察，可以发现，一共迭代了12次，误差结果并不递减，但逐渐收敛。总用时0.025秒。

$n = 50$ 时

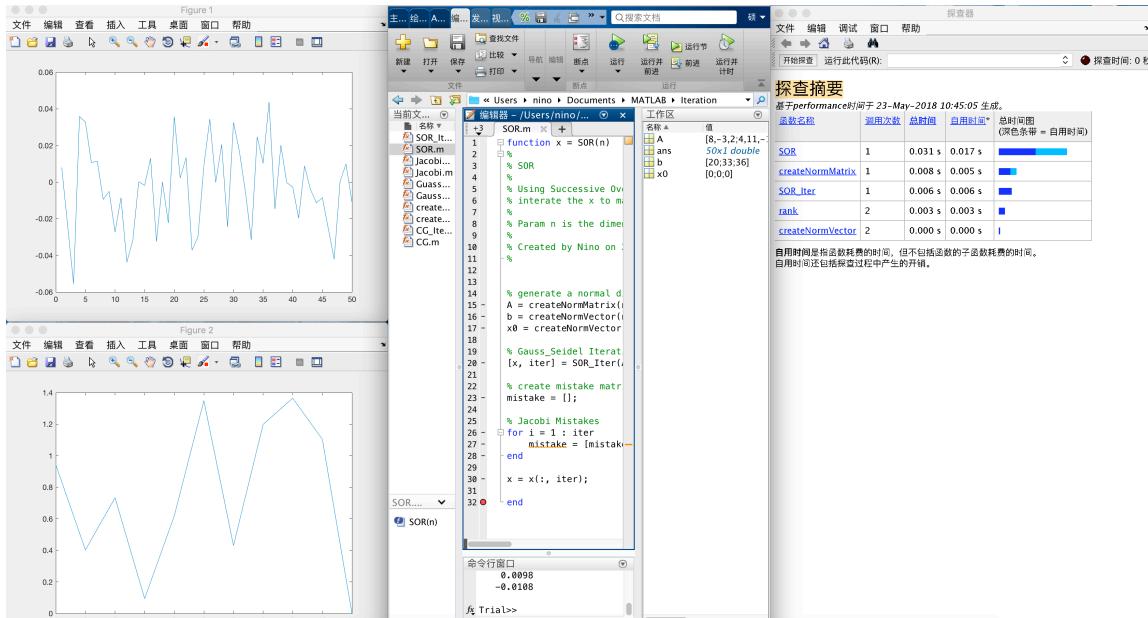


figure1是 x 的最终迭代结果；figure2是每次迭代对应的误差矩阵。通过观察，可以发现， x 的值每一项基本接近零；一共迭代了11次，误差结果并不递减，但逐渐收敛。总用时0.031秒。

$n = 100$ 时

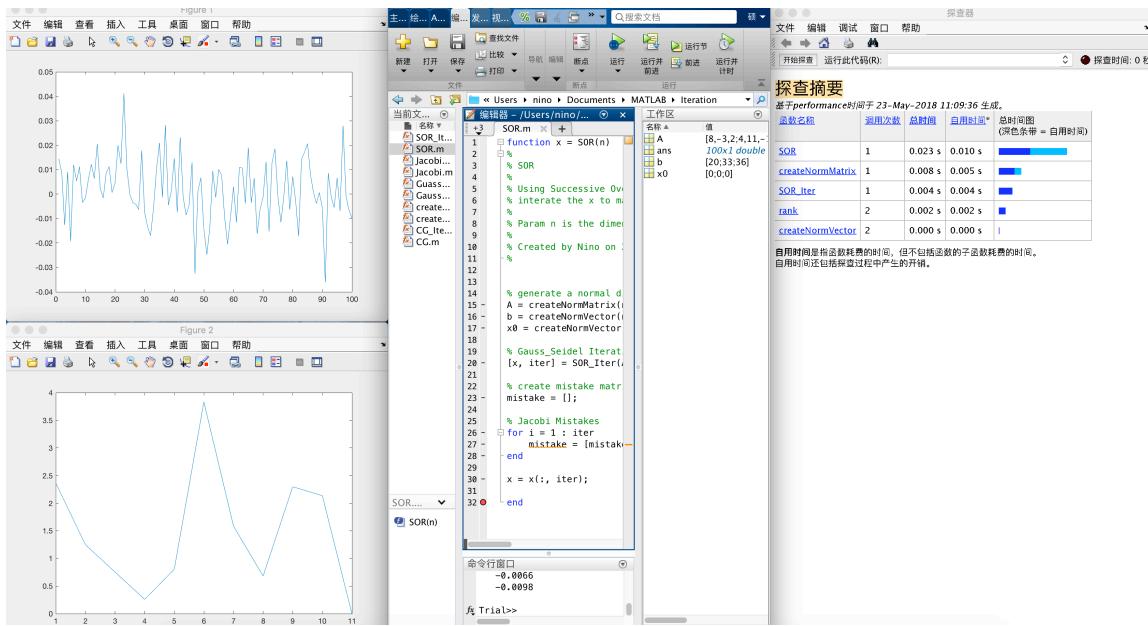
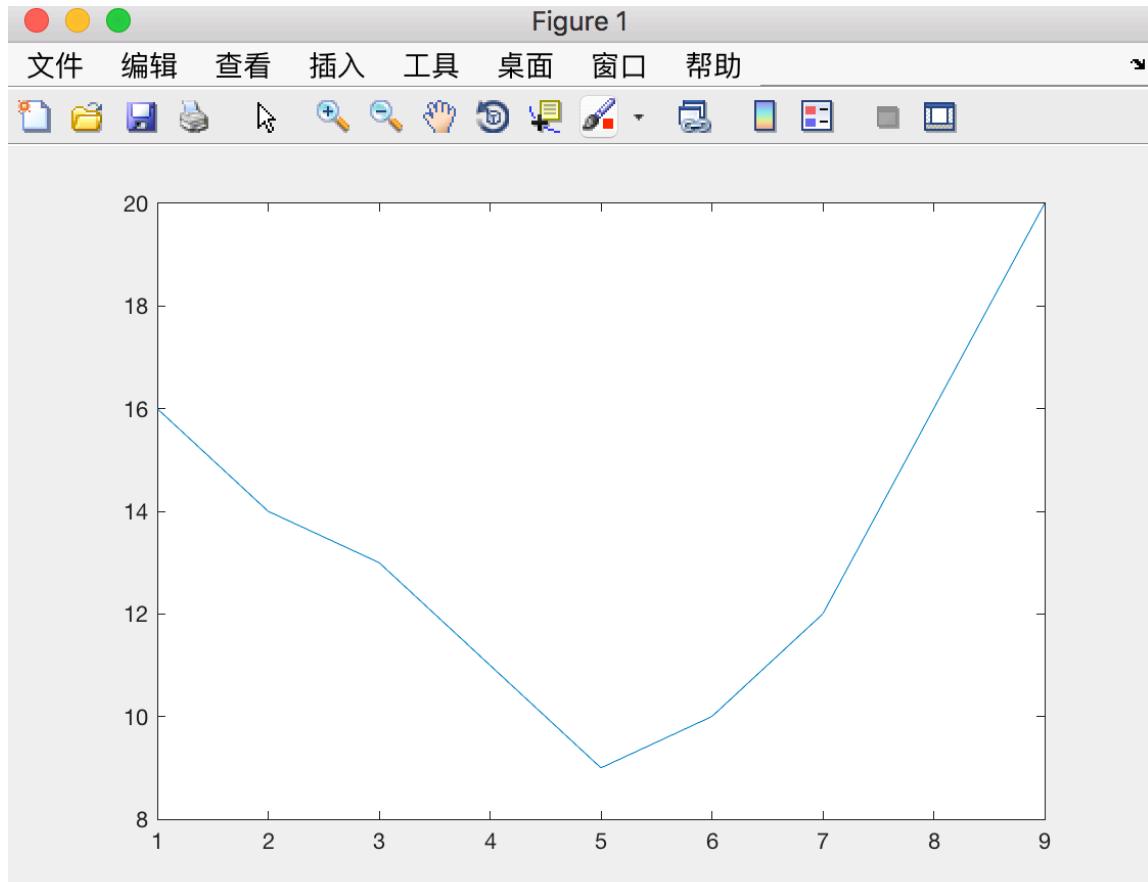


figure1是x的最终迭代结果；figure2是每次迭代对应的误差矩阵。通过观察，可以发现，x的值每一项基本接近零；一共迭代了11次，误差结果并不递减，但逐渐收敛。总用时出乎意料地变少到0.023秒。似乎可以发现随着n增大，这种方法的用时和迭代次数并没有大量增长。

对于逐次超松弛因子w的探索，利用了SOR(n)函数的大部分功能，只不过把mistake数组换成了关于w的数组。当n=100时，把w从0.8到1.2迭代次数存入数组，数组中的元素与迭代次数的关系如图。



可见，当 $w=0.8+5*0.05=1.05$ 的时候，迭代次数最好，w最适合，只用迭代9次。

4. 共轭梯度法

共轭梯度法是把一个求解方程组的问题转化成一个求解二次函数极小化的问题。从任意给定的初始点出发，沿一组关于矩阵A的共轭方向进线性搜索，在无舍入误差的假定下，最多迭代n次，就可以求出二次函数的极小点，也就是方程 $Ax = b$ 的解。

4.1. 算法设计

4.1.1. 设计思路

共轭梯度法的计算步骤如下所示：

- (1) 给定初始计算向量 $x^{(0)}$ 即精度 $\varepsilon > 0$;
- (2) 计算 $r^{(0)} = b - Ax^{(0)}$ ，取 $d^{(0)} = r^{(0)}$;
- (3) for $k=0$ to $n-1$ do

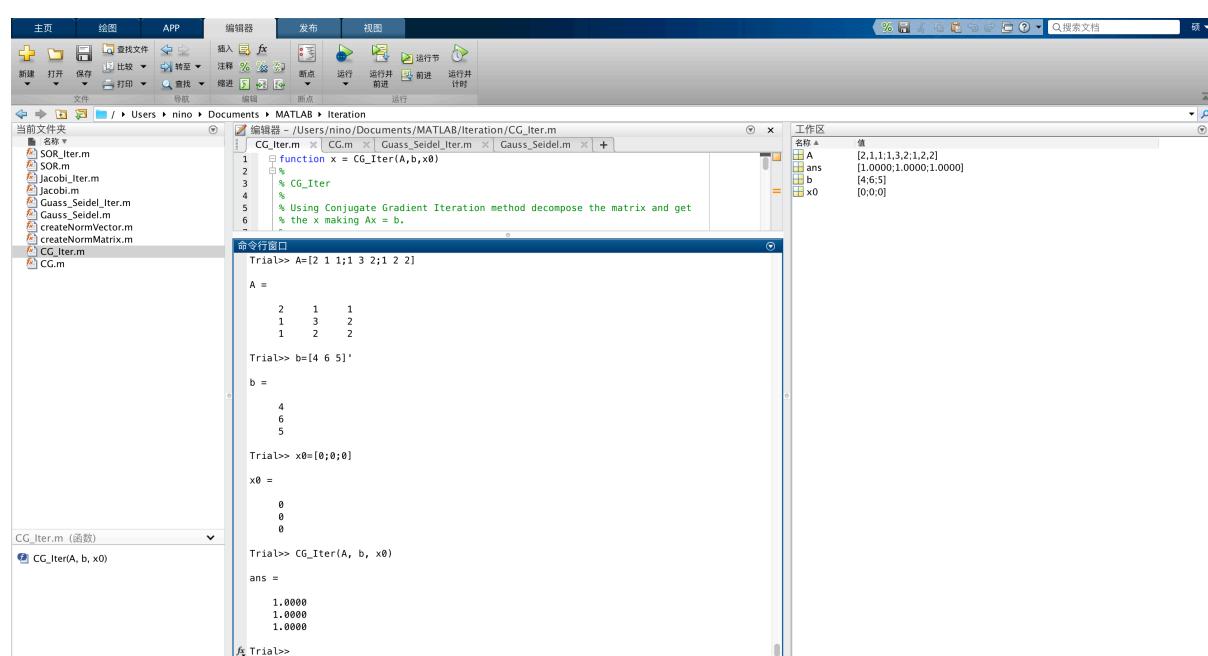
$$\textcircled{1} \quad \alpha_k = \frac{\mathbf{r}^{(k)T} \mathbf{r}^{(k)}}{\mathbf{d}^{(k)T} \mathbf{A} \mathbf{d}^{(k)}};$$

$$\textcircled{2} \quad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)};$$

4.1.2. 简单案例

用高斯消去法解下列线性方程组

$$\begin{cases} 2x_1 + x_2 + x_3 = 4 \\ x_1 + 3x_2 + 2x_3 = 6 \\ x_1 + 2x_2 + 2x_3 = 5 \end{cases}$$



得到结果 $\text{ans}=\mathbf{x}$, \mathbf{x} 是迭代结果。

4.2. 数值实验

为了探索共轭梯度法更广范的应用，我们把 \mathbf{A} 和 \mathbf{b} 分别设置为独立同分布的标准正态分布的对角占优矩阵和独立同分布的标准正态分布的向量。

4.2.1. 实验条件

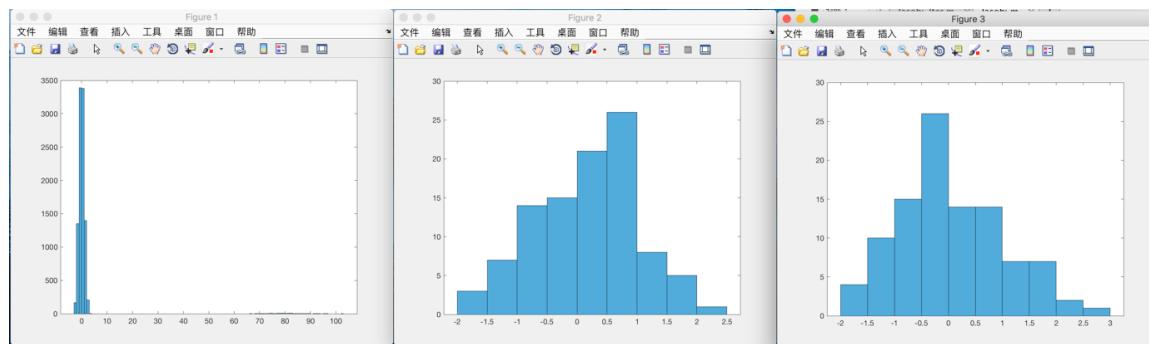
更普遍的应用需要一些条件制约。同高斯消元法和列主元消元法一样，当 \mathbf{A} 和 \mathbf{b} 为独立同分布的标准正态分布时，需要让 \mathbf{A} 、 \mathbf{b} 满足以下条件：线性方程组系数矩阵 \mathbf{A} 的顺序主子矩阵非奇异（有可逆矩阵）。当 \mathbf{A} 是方阵的时候，也就是说，所有的主元不为零。

不同于前面的方法，雅可比迭代法还需要让创造的正态矩阵和正态向量是对角占优矩阵以保证收敛——实验需要。为了构造这种矩阵，矩阵的对角元素的绝对值需要大于其所在行所有元素绝对值的和。

为了得到结果更加直观的显示，在辅助函数的返回值设置为每次迭代完的 \mathbf{x} 和迭代次数。把这个返回值带入到主函数，得到 $1 \times \text{iter}$ 的mistake向量，每一列的值分别为相对误差。用点阵图表示误差向量。

4.2.2. 附加步骤

设计两个函数：创建 $n \times n$ 的标准正态分布矩阵和 $n \times 1$ 的标准正态分布向量。将得到的矩阵和向量代入到迭代函数中去。除此之外，为了保证结果收敛，还要保证对角占优。如图为当 $n=100$ 时候创建出的 $n \times n$ 的标准正态分布矩阵和 $n \times 1$ 的标准正态分布 \mathbf{b} 向量和 \mathbf{x}_0 迭代初始向量。



通过图像可以发现， $n \times n$ 的标准正态分布矩阵不同于之前的，虽然大部分元素大致满足正态分布，有一些极端值。这些极端值是对角占优的正态矩阵的对角元素。另外 $n \times 1$ 的标准正态分布 \mathbf{b} 向量和 \mathbf{x}_0 迭代初始向量虽然二者都大体上满足正态分布，但是可以看得出，两者的图像并不完全相同。

1.3. 结果分析

由于共轭梯度法在实验中常常不收敛，所以在此只展示n为100时，CG法的性能。

n = 100时

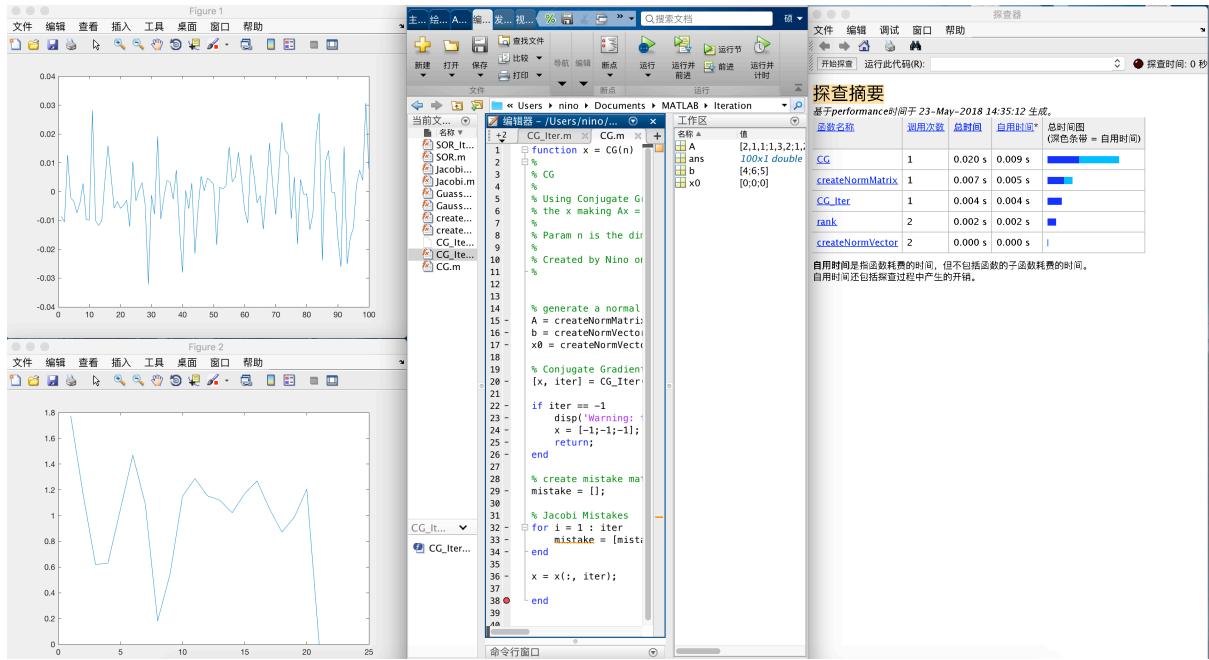


figure1是x的最终迭代结果；figure2是每次迭代对应的误差矩阵。通过观察，可以发现，x的值每一项基本接近零；一共迭代了21次，误差结果并不递减，但逐渐收敛。总用时0.020秒。这种方法虽然用时少，但结果常常不收敛。

四. 实验心得

实验中，这次主要是需要考虑到收敛问题。一开始没有考虑矩阵的对角占优，导致测试的大部分都无法收敛。但是对于共轭梯度法来说，好像仅仅对角占优是不够的，还要考虑到其他条件。我上网搜集了一些资料，但是没能运用到代码中去。

对比高斯消元法和列主元消元法，这四种方法并没有在运行时间上有很大提升，出乎意料，有时候改进的办法如逐次超松弛法在一些情况下甚至表现还不如Jacobi迭代法和Gauss-Seidel迭代法。我认为运行时间可能跟机器性能有关，所以有时候机器状况不是很好（例如开多个程序）的时候，有可能好的方法也表现不佳。

运行成功Jacobi迭代法后，Gauss-Seidel迭代法和逐次超松弛迭代法就容易一些，因为Gauss-Seidel迭代法和逐次超松弛迭代法只是在Jacobi迭代法的基础上变化了矩阵B和f，迭代公式都是相同的。重点是收敛条件控制——一开始我没有注意这个条件，没有设置好eps，浪费了好多时间。逐次超松弛迭代法要注意迭代因子，1.1左右最佳，要是过大或过小就会导致不收敛。共轭梯度法

的收敛性是这个问题的难点，我查阅了很多资料也没能改进，使我的代码能够每次都收敛。

实验三

一. 实验环境和工具

在OS X High Sierra 10.13.4 环境下利用Matlab-R2018a完成实验。

二. 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Page Rank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。

三. 实验内容

1. Page Rank

PageRank是Google研发的主要应用于评估网站可靠度和重要性的一种算法，是进行网页排名的考量指标之一。PageRank算法是一种链接分析算法，由Google提出并用于标识网页重要性。PageRank算法基于两个假设：入链数量假设：如果一个网页的入链数量越多，则其重要程度越高；入链质量假设：高质量的网页为其链接的页面带去更多权重。基于这两条假设，PageRank算法为每个页面设置初始权重值，根据网页间的链接关系，在多轮迭代中更新每个网页的权重，直至各页面的权重值稳定。不考虑作弊的情况，我们通常将最终权重值越高的节点视为越可靠网页。

1.1. 算法设计

1.1.1. 设计思路

PageRank算法总的来说就是预先给每个网页一个PR值（下面用PR值指代PageRank值），于PR

值物理意义上为一个网页被访问概率，所以一般是 $1/N$ ，其中 N 为网页总数。另外，一般情况下，所有网页的PR值的总和为1。如果不为1的话也不是不行，最后算出来的不同网页之间PR值的大小关系仍然是正确的，只是不能直接地反映概率了。预先给定PR值后，通过下面的算法不断迭代，直至达到平稳分布为止。

互联网中的众多网页可以看作一个有向图。如果B、C节点只指向A节点，那么 $PR(A) = PR(B) + PR(C)$ 。如果C节点有两条出链，另一条指向D节点，那么C节点跳转到A节点的概率就是 $PR(C)/2$ ，那么A节点的PR值计算方法是 $PR(A) = PR(B) + PR(C)/2$ 。

对于没有出链的节点，假设它到每个节点都有出链，也就是其PR值除以总的节点数。

对于成环的几个节点或者自己指向自己的节点，上面的方法显然不科学，随着迭代这些节点的PR值会一直增长。为了抑制这种情况，设置一个参数 α ， $(1-\alpha)$ 代表一个人跳转到随机网页的概率。于是一个节点的PR值可以表示成对它有出链的节点的PR值加权，以及随机跳转的网页。

所以一个网页的PR值可以表示成 $ranks = \alpha * oldrank * sparse_data + c * I * oldrank$ ，其中 $ranks$ 是网站节点的PR值向量， α 是跳转参数， $oldrank$ 是上次迭代的值， $sparse_data$ 是经过均分之后的数据矩阵， c 是参数 $c = (1-\alpha) / n$ ， I 是单位向量。

上式中因为 I 是稠密矩阵，对于实验中大容量的数据集并不适用。所以利用数学方法进行改进，把 $I * oldrank$ 换成 $c * sum(oldrank)$ 。这样就能得到网站节点的PR值向量。

1.1.2. 伪代码

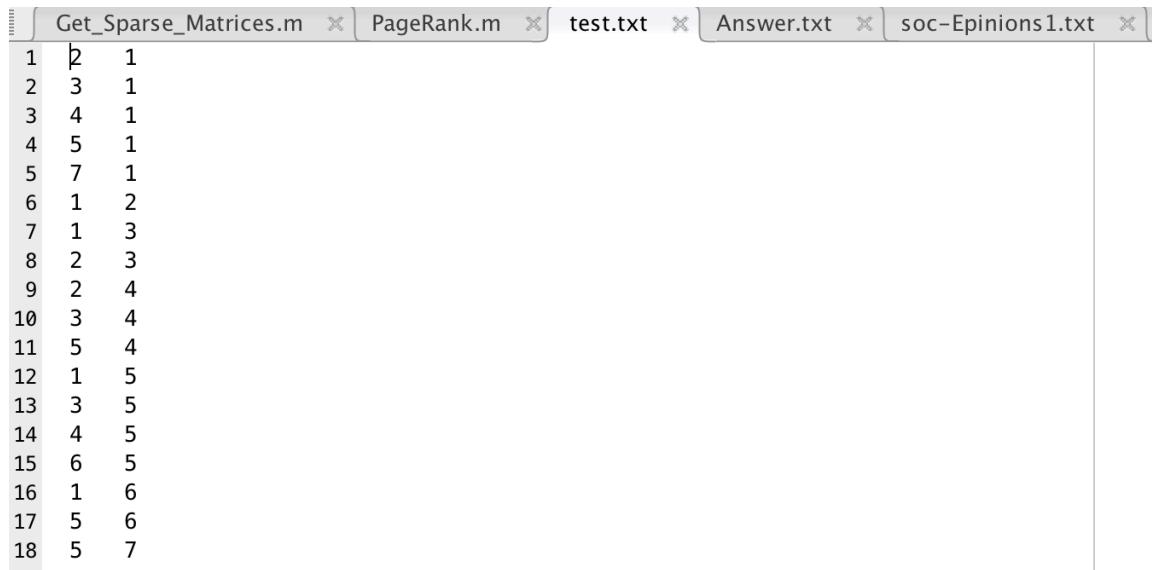
Pseudocode for PageRank (G)

Input: Let G represent set of nodes or web pages

Output: An n-element array of PR which represent PageRank for each web page

1. For $i \leftarrow 0$ to $n-1$ do
2. Let A be an array of n elements
3. $A[i] \leftarrow 1/n$
4. $d \leftarrow$ some value $0 < d < 1$, e.g. 0.15, 0.85
5. Repeat
6. For $i \leftarrow 0$ to $n-1$ do
7. Let PR be a n-element of array
8. $PR[i] \leftarrow 1-d$
9. For all pages Q such that Q links to PR[i] do
10. Let O_n be the number of outgoing edge of Q
11. $PR[i] \leftarrow PR[i] + d * A[Q]/O_n$
12. If the difference between J and PR is small do
13. Return PR
14. For $i \leftarrow 0$ to $n-1$ do
15. $A[i] \leftarrow PR[i]$

1.1.3. 简单案例

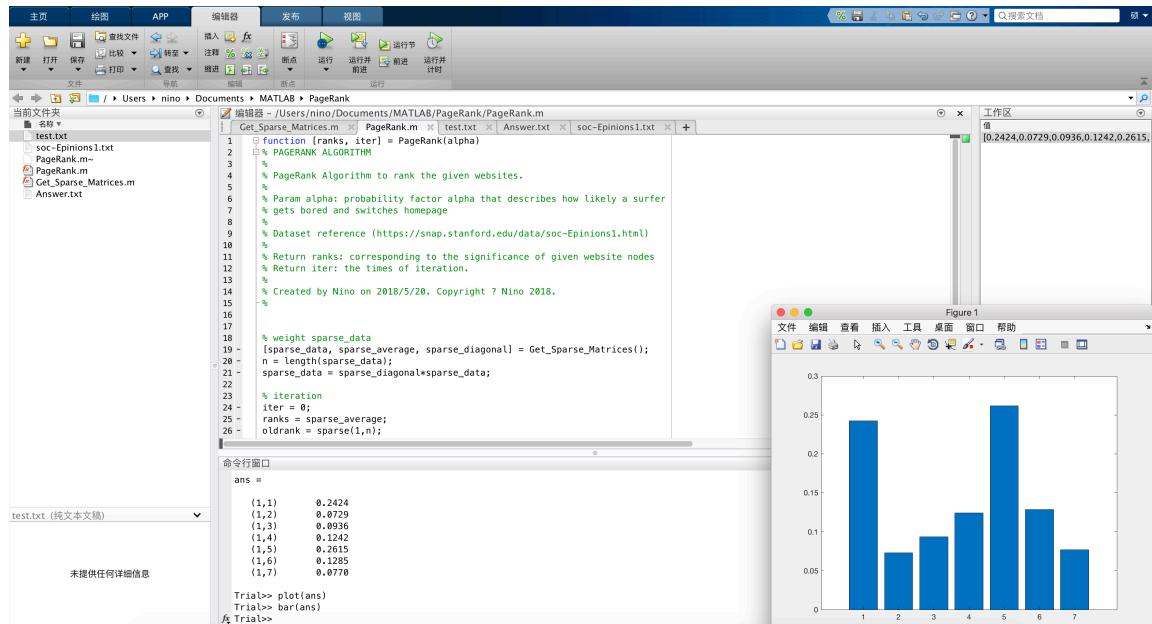


The screenshot shows the MATLAB Editor window with several tabs open. The tabs include 'Get_Sparse_Matrices.m', 'PageRank.m', 'test.txt', 'Answer.txt', and 'soc-Epinions1.txt'. The code in 'Get_Sparse_Matrices.m' displays a list of 18 edges connecting 7 nodes:

```
1 2 1
2 3 1
3 4 1
4 5 1
5 7 1
6 1 2
7 1 3
8 2 3
9 2 4
10 3 4
11 5 4
12 1 5
13 3 5
14 4 5
15 6 5
16 1 6
17 5 6
18 5 7
```

设置一组简单的节点关系如图。图中7个节点，18条关系。用PageRank对它们排序，得到权值数组的值为 $\text{ans} = [0.2424, 0.0729, 0.0936, 0.1242, 0.2615, 0.1285, 0.0770]$ 。

节点的权值的条形图如图所示。可以发现5号节点受信度最高，说明他入链数量多，且入链多为高质量的网页，带去更多权重。



1.2. 数值实验

利用在 Epinions 社交数据集(<https://snap.stanford.edu/data/soc-Epinions1.html>)Page Rank方法

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。由于网站节点的PR权重都太低，所以扩大 10^5 倍。

1.2.1. 实验条件

更普遍的应用需要一些条件制约。Epinions 社交数据集包含7万多个节点，用普通矩阵实现显然是不现实的，于是采用稀疏矩阵数据结构。

1.2.2. 附加步骤

设计三个函数：读取数据并创建相关的稀疏矩阵函数；给ranks向量排序函数；写到文件函数。

1.3. 结果分析

结果已经写到answer.txt文件中，如图所示，左边的窗口是数据集，右边的窗口是结果输出，为了方便表示，评分是ranks* 10^5 。

The screenshot shows the MATLAB workspace with the following details:

- Current Folder:** Documents > MATLAB > PageRank
- Answer.txt:** A text file containing the following data (first 27 rows):

node	score
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
48	46.174941
118	23.993882
1719	22.632075
136	21.814541
46	21.316946
143	21.315741
798	20.973382
725	17.695479
849	17.467864
481	17.455679
1619	17.195795
27	16.729748
77	16.266526
1481	16.229230
135	16.117865
1179	15.980909
1191	15.932981
128	15.558427
558	15.557938
918	15.538643
1164	15.341513
34	15.198734
- PageRank_write.m:** A function that writes the data to a sparse matrix.
- PageRank_Sort.m:** A function that sorts the data.
- Get_Sparse_Matrices.m:** A function that reads the data from a file.

四. 实验心得

在这次实验中，我主要是熟悉了PageRank算法——从计算平均出链权到进行迭代。一开始没有建立疏松矩阵，所以超内存了。后来用疏松矩阵存所有矩阵，但是计算过程中出现了单位矩阵，单位矩阵在本质上还是稠密矩阵。后来通过数学变换，找到了方法，用 $\text{sum}(\text{oldrank})$ 替换了 $\text{l}^*\text{oldrank}$ ，收效明显：得到了75888个节点，508837条指向关系的矩阵，以及对每个节点进行了排序。征服数据集快乐的同时，也不禁为Matlab强大的矩阵计算功能折服！

我认为只有通过大的数据以及对其的分析才能发现数值计算的精髓所在，小的算法改变也可以对整个工业应用有深远影响。

这次实验的重点是掌握pagerank算法的思想。其中利用疏松矩阵和数学变换解决大量数据排序是本次实验的难点。
