首先编制一个能自动选取主元，又能手动选取主元的求解线性方程组的Gauss消去法程序。

1. function [ out ] = Gauss( A,b,flag,ptol )
2. if nargin<4,ptol=50\*eps;end
3. [m,n]=size(A);
4. if m~=n,error('A不是方阵');end
5. out=zeros(n,1);% 预先设定解的维数
6. nb=n+1;
7. Ab=[A,b];    %扩维矩阵
8. RA=rank(A);
9. RB=rank(Ab);zhica=RB-RA;
10. if zhica>0,
11. disp('请注意：因为RA~=RB，所以此方程组无解.')
12. return
13. end
14. for i=1:n-1 ;
15. i;
16. if flag==0;
17. pivot=Ab(i,i);
18. ri=i;
19. elseif flag==1;% 按照每列模最小的选取
20. [pivot,min\_index]=min(abs(A(i:n,i)));
21. ri=i+min\_index-1;
22. elseif flag==2;% 按照模最大的选取
23. [pivot,max\_index]=max(abs(A(i:n,i)));
24. ri=i+max\_index-1;
25. elseif flag==3     %方程最小非0数
26. tA=A;
27. [pivot,min\_index]=min(abs(tA(i:n,i)));
28. while pivot==0;
29. tA(min\_index+i-1,i)=inf;
30. [pivot,min\_index]=min(abs(tA(i:n,i)));
31. end
32. ri=i+min\_index-1;
33. end
34. if (pivot==0)||(pivot<ptol) ;
35. warning('系数矩阵奇异！！');
36. return;
37. end
38. if ri~=i;  % 交换行
39. tmp=A(i,:);A(i,:)=A(ri,:);A(ri,:)=tmp;
40. t=b(i);b(i)=b(ri);b(ri)=t;
41. end
42. for kk=i+1:n
43. L(kk,i)=A(kk,i)/A(i,i);
44. A(kk,i+1:n)=A(kk,i+1:n)-L(kk,i)\*A(i,i+1:n);
45. b(kk)=b(kk)-L(kk,i)\*b(i);
46. end
47. end
48. if A(n,n)==0
49. warning('系数矩阵奇异！！');
50. return;
51. end
52. x=zeros(n,1)';
53. for k=n:-1:1
54. %     k
55. %     A(k,k+1:n)
56. %     x(k+1:n)
57. if k==n
58. x(n)=b(n)/A(n,n);
59. else
60. x(k)=(b(k)-sum( A(k,k+1:n).\*x(k+1:n) ) )/(A(k,k));
61. end
62. end
63. out=x';
64. end

取矩阵，则方程有解。取*n*=10计算矩阵的条件数。让程序自动选取主元，结果如何？

解答：首先先构造出A与b：

n=10; %n值可以根据要求改变,这里根据题目选n=10

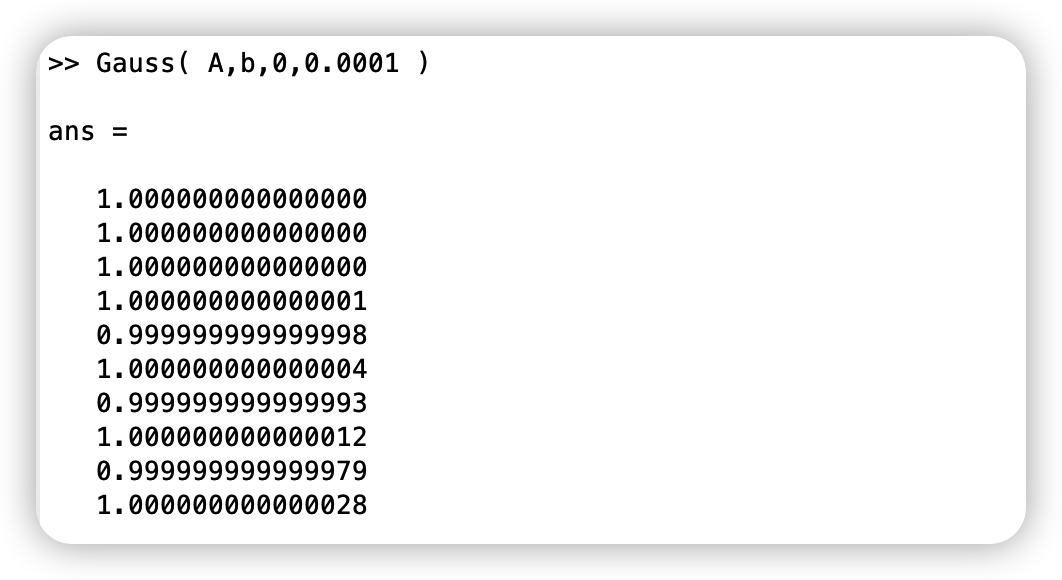
A=diag(6\*ones(1,n))+diag(ones(1,n-1),1)+diag(8\*ones(1,n-1),-1);

b=A\*ones(n,1);

接下来求出n=10矩阵的条件数，由于matlab自带cond( )函数可以求解矩阵的条件数，因此我们直接运用：



让程序自动选取主元，结果如下：



2. 选择程序中手动选取主元的功能。每步消去过程总选取按模最小或按模尽可能小的元素作为主元，观察并记录计算结果。若每步消去过程总选取按模最大的元素作为主元，结果又如何？分析实验的结果。

>> Gauss( A,b,1,0.0001 )

ans =

1.000000000000000

1.000000000000000

1.000000000000000

1.000000000000001

0.999999999999998

1.000000000000004

0.999999999999993

1.000000000000012

0.999999999999979

1.000000000000028

>> Gauss( A,b,2,0.0001 )

ans =

1.0000000000

1.0000000000

1.0000000000

1.0000000000

1.0000000000

1.0000000000

1.0000000000

1.0000000000

1.0000000000

1.0000000000

>> Gauss( A,b,3,0.0001 )

ans =

1.000000000000000

1.000000000000000

1.000000000000000

1.000000000000001

0.999999999999998

1.000000000000004

0.999999999999993

1.000000000000012

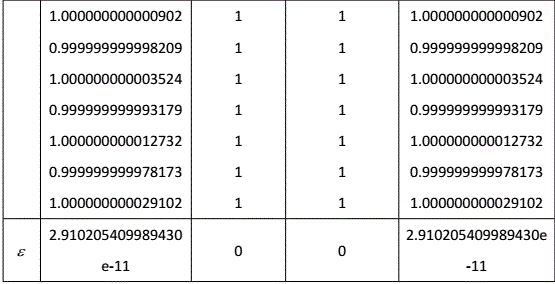
0.999999999999979

1.000000000000028

从（1）和（2）的实验结果可以发现，列主元消去法和完全主元消去法都得到了精确解，而顺序高斯消去法和以模尽量小的元素为主元的消去法没有得到精确解。在后两种消去法中，由于计算是的舍入误插，对最终结果会产生一定影响，但由于方程组的维数较低，而且元素之间相差不大，所以误差仍比较小。

1. 取矩阵阶数*n*=20或者更大，重复上述实验过程，观察记录并分析不同的问题及消去过程中选择不同的主元时计算结果的差异，说明主元素的选取在消去过程中的作用。





可以发现，列主元和完全主元的计算结果依然为精确值，但是顺序高斯消去和模尽可能小的方法会存在一定误差，和n=10的时候误差相比，n=20要更大一点。

4. 将上述矩阵A中的主元改为0.00006再重新作一次数值实验看看。

解：在准备A的时候进行改变：

n=10; %n值可以根据要求改变,这里根据题目选n=10

A=diag(0.000066\*ones(1,n))+diag(ones(1,n-1),1)+diag(8\*ones(1,n-1),-1);

B=diag(6\*ones(1,n))+diag(ones(1,n-1),1)+diag(8\*ones(1,n-1),-1);

b=B\*ones(n,1);

结果：

Gauss( A,b,2,0.01 )

ans =

1.0e+04 \*

0.0002

0.0007

0.0002

-0.0041

0.0002

0.0343

0.0002

-0.2729

0.0002

2.1847

实验总结：

消元法是将方程组中的一方程的未知数用含有另一未知数的代数式表示，并将其代入到另一方程中，这就消去了一未知数，得到一解；或将方程组中的一方程倍乘某个常数加到另外一方程中去，也可达到消去一未知数的目的。消元法主要用于二元一次方程组的求解。数学上，高斯消元法，是线性代数规划中的一个算法，可用来为[线性方程组](https://baike.baidu.com/item/%E7%BA%BF%E6%80%A7%E6%96%B9%E7%A8%8B%E7%BB%84/5904308?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%AB%98%E6%96%AF%E6%B6%88%E5%85%83%E6%B3%95/_blank)求解。但其算法十分复杂，不常用于加减消元法，求出矩阵的秩，以及求出可逆方阵的逆矩阵。不过，如果有过百万条等式时，这个算法会十分省时。一些极大的方程组通常会用迭代法以及花式消元来解决。当用于一个矩阵时，高斯消元法会产生出一个“行梯阵式”。高斯消元法可以用在电脑中来解决数千条等式及未知数。亦有一些方法特地用来解决一些有特别排列的系数的方程组。高斯消元法可以用来找出一个[可逆矩阵](https://baike.baidu.com/item/%E5%8F%AF%E9%80%86%E7%9F%A9%E9%98%B5?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%AB%98%E6%96%AF%E6%B6%88%E5%85%83%E6%B3%95/_blank)的逆矩阵。设A 为一个N \* N的[矩阵](https://baike.baidu.com/item/%E7%9F%A9%E9%98%B5?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%AB%98%E6%96%AF%E6%B6%88%E5%85%83%E6%B3%95/_blank)，其逆矩阵可被两个[分块矩阵](https://baike.baidu.com/item/%E5%88%86%E5%9D%97%E7%9F%A9%E9%98%B5?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%AB%98%E6%96%AF%E6%B6%88%E5%85%83%E6%B3%95/_blank)表示出来。将一个N \* N单位矩阵 放在A 的右手边，形成一个N \* 2N的分块矩阵B = [A,I] 。经过[高斯](https://baike.baidu.com/item/%E9%AB%98%E6%96%AF?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%AB%98%E6%96%AF%E6%B6%88%E5%85%83%E6%B3%95/_blank)消元法的计算程序后，矩阵B 的左手边会变成一个单位矩阵I ，而逆矩阵A ^(-1) 会出现在B 的右手边。假如高斯消元法不能将A 化为三角形的格式，那就代表A 是一个不可逆的矩阵。应用上，高斯消元法极少被用来求出[逆矩阵](https://baike.baidu.com/item/%E9%80%86%E7%9F%A9%E9%98%B5?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%AB%98%E6%96%AF%E6%B6%88%E5%85%83%E6%B3%95/_blank)。高斯消元法通常只为[线性方程组](https://baike.baidu.com/item/%E7%BA%BF%E6%80%A7%E6%96%B9%E7%A8%8B%E7%BB%84?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%AB%98%E6%96%AF%E6%B6%88%E5%85%83%E6%B3%95/_blank)求解。

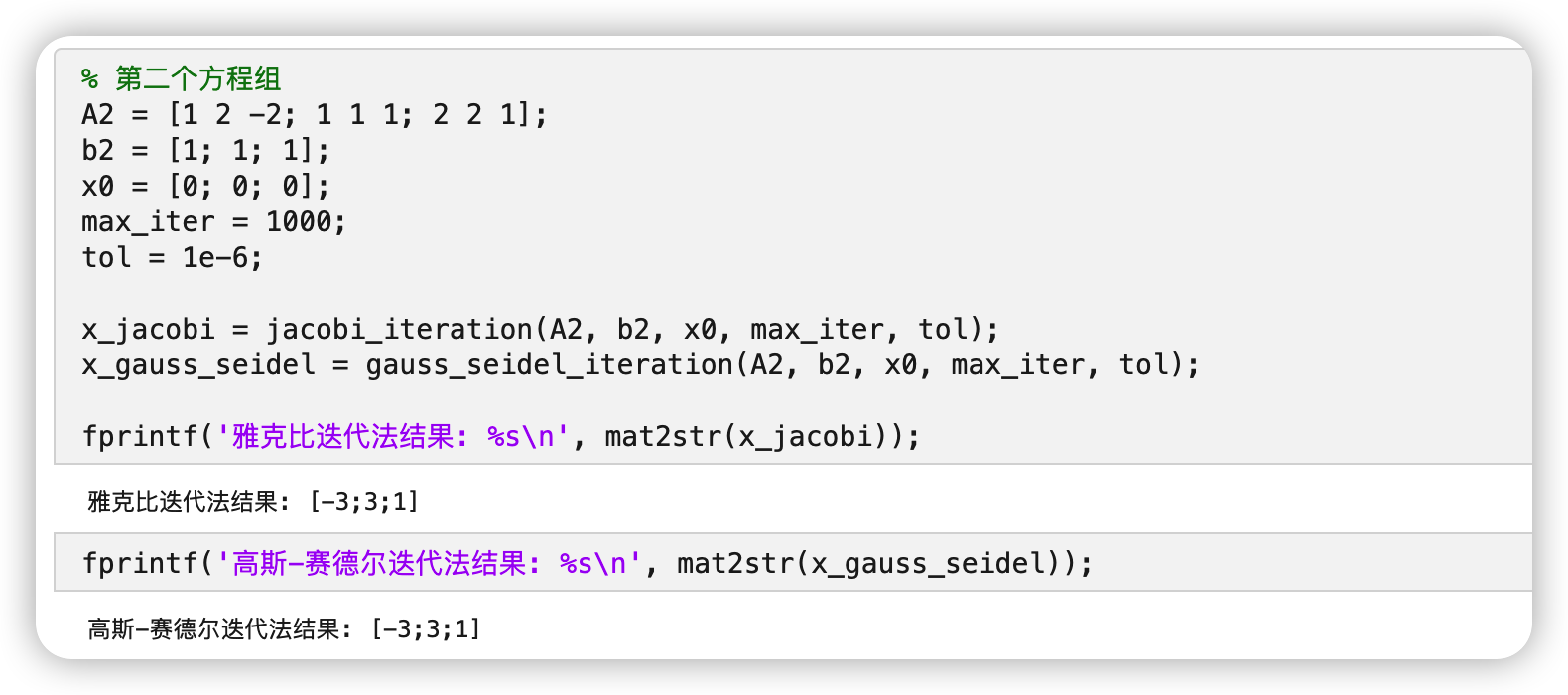
**雅克比迭代法：**

1. function x = jacobi\_iteration(A, b, x0, max\_iter, tol)
2. n = length(b);
3. x = x0;
4. for k = 1:max\_iter
5. x\_new = x;
6. for i = 1:n
7. sigma = A(i, :) \* x\_new - A(i, i) \* x\_new(i);
8. x(i) = (b(i) - sigma) / A(i, i);
9. end
10. if norm(x - x\_new, inf) < tol
11. break;
12. end
13. end
14. end

**高斯-赛德尔迭代法：**

1. function x = gauss\_seidel\_iteration(A, b, x0, max\_iter, tol)
2. n = length(b);
3. x = x0;
4. for k = 1:max\_iter
5. x\_new = x;
6. for i = 1:n
7. sigma = A(i, 1:i-1) \* x\_new(1:1) + A(i, i+1:end) \* x(i+1:end);
8. x(i) = (b(i) - sigma) / A(i, i);
9. end
10. if norm(x - x\_new, inf) < tol
11. break;
12. end
13. end
14. end





雅克比迭代法和高斯-赛德尔迭代法都是迭代求解线性方程组的方法，它们在每一步迭代中更新解向量。主要的区别在于更新解向量的方式。

雅克比迭代法每次迭代都使用前一次迭代中的全部解分量来计算下一次的解分量。具体而言，在第 **i** 步迭代中，雅克比迭代法计算新的解分量 **x(i)** 时使用的是上一次迭代的所有解分量。

而高斯-赛德尔迭代法则是在计算新的解分量时，使用了当前迭代中已经更新的解分量。例如，在计算 **x(2)** 时，高斯-赛德尔迭代法使用了已经更新的 **x(1)**。这使得高斯-赛德尔法在收敛速度上可能比雅克比法更快。

1. function [x, iter] = sor\_method(A, b, omega, x0, max\_iter, tol)
2. n = length(b);
3. x = x0;
4. iter = 0;
5. while iter < max\_iter
6. x\_old = x;
7. for i = 1:n
8. sigma = A(i, 1:i-1) \* x(1:i-1) + A(i, i+1:end) \* x\_old(i+1:end);
9. x(i) = (1 - omega) \* x\_old(i) + omega \* (b(i) - sigma) / A(i, i);
10. end
11. iter = iter + 1;
12. if norm(x - x\_old, inf) <= tol
13. break;
14. end
15. end
16. end

