Lecture 1 Iterative（迭代）

Methods for Ax=b （矩阵方程）

问题背景

- 线代方法：高斯消元法

> $O(n^3)$

- 问题：当n足够大时，有没有方便的方法?
- 想法：让矩阵具有一些特殊的性质便于分析->让A变得**稀疏**（有很多个0）

从“一元一次方程”获得的启示

如果我们将其他未知量当成已知，得出一个未知量与其他未知量的表达式，然后给定初值，进行迭代，就能逼近真实值。

> 如：$4x-y+z=7, 4x-8y+z=-21, -2x+y+5z=15$,得到迭代式：
> $x_{k+1}=(y_k-z_k+7)/4, y_{k+1}=(4x_k+x_z+21)/8, z_{k+1}=(2x_k-y_k+15)/5$
> **用线代语言描述：x_{k+1}=Mx_k+b_2，又称Jacob迭代**
> matlab代码如下：

```
clear all

A = [4,-1,1;
     4,-8,2;
     -2,1,5];

b = [7,-21,15];

% 直接求解（注意：应该使用左除而不是右除）
x_direct = A \ b';

% 迭代方法：x_{k+1}=Mx_k+b_2
M = [0 ,1/4,-1/4;
     1/2,0,1/8;
     2/5,-1/5,0];

b_2 = [7/4;21/8;3];

x_0 = [2;3;3];

% 迭代算法
tol = 1e-5;
max_iter = 100;
x_history = zeros(3, max_iter+1);
x_history(:, 1) = x_0;

err = tol * 2;
index = 1;

while (err > tol) && (index < max_iter)
    x_history(:, index+1) = M * x_history(:, index) + b_2;
    err = norm(x_history(:, index+1) - x_history(:, index), Inf);
    index = index + 1;
end

fprintf('经过 %d 次迭代后收敛\n', index-1);
fprintf('最终解: [%.6f, %.6f, %.6f]\n', x_history(:, index));
fprintf('直接法的解: [%.6f, %.6f, %.6f]\n', x_direct');
```

> 补充：norm函数：找出向量中的最大值，Inf是让计算无穷范数

- 发现问题：把1、3方程互换，发现计算结果发散。**说明这种算法对A、b有要求。**

迭代算法修正

- **Strict Diagonal Dominance(SDD条件)**：$|a_{kk}| > \sum_{j=1,j\neq k}^{n}|a_{kj}|$,即对角元素绝对值比该行其他元素绝对值之和都要大。

> 该条件是充分非必要条件。

- 将M和b_2用A,b表示：
  将A的对角元素提取出来为D，即$A=D+T$，那么$x=-D^{-1}Tx+D^{-1}b$
  这样，我们的代码就不用手动计算这两个矩阵了：

```matlab
clear all

A = [4,-1,1;
     4,-8,2;
     -2,1,5];

b = [7,-21,15];

% 直接求解（注意：应该使用左除而不是右除）
x_direct = A \ b';  % 正确的直接解法

% 迭代方法2：x_{k+1}=Mx_k+b_2
D = diag(diag(A));
T = A - D;

M = -inv(D) * T;
b_2 = inv(D) * b';

x_0 = [2,2,3];

% 迭代算法修正
tol = 1e-5;
max_iter = 100;
x_history = zeros(3, max_iter+1);
x_history(:, 1) = x_0;

err = tol * 2;
index = 1;

while (err > tol) && (index < max_iter)
    x_history(:, index+1) = M * x_history(:, index) + b_2;
    err = norm(x_history(:, index+1) - x_history(:, index), Inf);
    index = index + 1;
end

fprintf('经过 %d 次迭代后收敛\n', index-1);
fprintf('最终解: [%.6f, %.6f, %.6f]\n', x_history(:, index));
fprintf('直接法的解: [%.6f, %.6f, %.6f]\n', x_direct');
```

我们还可以将求解的过程封装成一个函数，并保存为一个单独的文件：

```
function [x,err,iter] = JacobiIter(A,b,x_0,tol,maxIter)
    D = diag(diag(A));
    T = A - D;

    M = -inv(D) * T;
    b_2 = inv(D) * b';

    x_0 = [2,2,3];

    tol = 1e-5;
    max_iter = 100;
    x_history = zeros(3, max_iter+1);
    x_history(:, 1) = x_0;

    err = tol * 2;
    index = 1;

    while (err > tol) && (index < max_iter)
        x_history(:, index+1) = M * x_history(:, index) + b_2;
        err = norm(x_history(:, index+1) - x_history(:, index), Inf);
        index = index + 1;
    end
end
```

改进后的代码如下：

```matlab
clear all

A = [4,-1,1;
     4,-8,2;
     -2,1,5];

b = [7,-21,15];

% 直接求解（注意：应该使用左除而不是右除）
x_direct = A \ b';  % 正确的直接解法

% 迭代方法2：x_{k+1}=Mx_k+b_2
xest = JacobiIter(A,b,[1;2;2],1e-6,100);

fprintf('经过 %d 次迭代后收敛\n', index-1);
fprintf('最终解: [%.6f, %.6f, %.6f]\n', x_history(:, index));
fprintf('直接法的解: [%.6f, %.6f, %.6f]\n', x_direct');
```