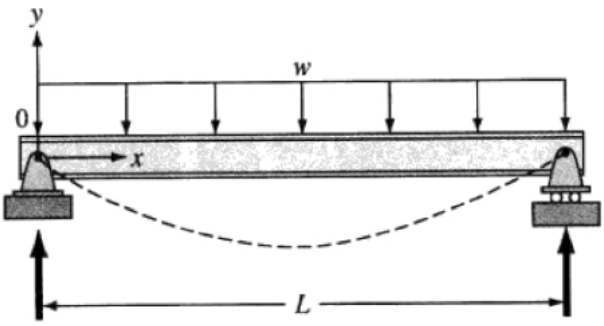
1 问题描述

某非均匀负载横梁上，弹性曲线的基本微分方程为：$EI\frac{d^2y}{dx^2}=\frac{wLx}{2}-\frac{wx^2}{2}$。
其中，E为弹性模量，I为惯性矩，利用有限差分法求横梁的挠曲。参数值为：
$E=30000ksi, I=800in^4, w=1kip/ft, L=10ft$
其中$1ft=12in$。微分方程的解析解为$y=\frac{wLx^3}{12EI}-\frac{wx^4}{24EI}-\frac{wL^3x}{24EI}$
有限差分法：$\frac{d^2y}{dx^2}=\frac{y_{i+1}-2y_i+y_{i-1}}{\Delta x^2}$
用有限差商代替导数，将区间分成n等分，取等距点，每个内点适用此方程，转化为联立的代数方程组进行求解。
要求：取不同Δx时，分别采用直接法和迭代法的不同方法求解方程组并进行对比。（如图建立坐标系）



2 问题分析

依照题意，首先统一单位in与ft，$w=\frac{1}{12}kip/in, L=120in$。然后把所有字母回代，得到：

$$30000\times 800\frac{d^2y}{dx^2}=5x-\frac{x^2}{24}$$

再将二阶导的表达式代入

$$24000000\times\frac{y_{i+1}-2y_i+y_{i-1}}{\Delta x^2}=5x-\frac{x^2}{24}$$

然后依据题意将L等分，得到若干x点的值，对每个x依次代入这个表达式，可以得到其对应的y值与相邻y值的关系式。以此可以得到一系列y的线性方程组，解得这一系列y便可以用这些一系列点拟合原曲线。

重点在于方程组的获得与求解。

3 代码实现

3.1 准备工作

均分点的工作

```
function [X,h]=separate(d,a,b)%均分点，d是点的个数，b、a是上下界，包括端点
    X=[];
    h=(b-a)/(d-1);
    for i=1:d
        x=a+h*(i-1);
        X=[X,x];
    end
end
```

3.2 矩阵的建立

对原式子来说，当x=0或120 in时，等号右侧都等于0，这符合横梁两端点y(0)=0，y(120)=0的初始值条件。
先假设一共取了4个点（包括端点），得到的等式如下：

$$\begin{cases} y_1 - 2y_2 + y_3 = b_1 \\ y_2 - 2y_3 + y_4 = b_2 \end{cases}$$

如果把y1,y2,y3,y4看作变量的话，得到的是一个4×2的矩阵，不是方阵不好求，但是y1与y4是端点值固定为0，可以直接删去，因此变成关于y2，y3的2×2的矩阵，变成方阵了，便于计算。

为了便于我们编程，我们可以先定义一个(n-2)×n的零矩阵，依次将y1到yn的方程系数更新好，然后再将y1与yn对应的首尾列删去，变成一个(n-2)×(n-2)的方阵(其中n是包括端点的所有取点数)，对应的变量为中间(n-2)个y值。

```
[X,h]=separate(11,0,120);
n=length(X);%点数
A=zeros(n-2,n);%定义初始0矩阵
B=[];%常数向量
for i=2:n-1
    k=5*X(i)-X(i)*X(i)/24;
    k=k*h*h/24000000;%常数项
    B=[B,k];
    A(i-1,i-1)=1;
    A(i-1,i)=-2;
    A(i-1,i+1)=1;
end
A_re=A(:, 2:n-1);
X_re=X(2:n-1);
```

结果如下

```
  -2     1     0     0     0     0     0     0     0
   1    -2     1     0     0     0     0     0     0
   0     1    -2     1     0     0     0     0     0
   0     0     1    -2     1     0     0     0     0
   0     0     0     1    -2     1     0     0     0
   0     0     0     0     1    -2     1     0     0
   0     0     0     0     0     1    -2     1     0
   0     0     0     0     0     0     1    -2     1
   0     0     0     0     0     0     0     1    -2

 1.0e-03 *

  0.3240    0.5760    0.7560    0.8640    0.9000    0.8640    0.7560    0.5760    0.3240
```

3.3 解方程组

直接法

原始高斯消元法

```
function x=Gauss(a,b)
    [m,n]=size(a);
    for k=1:n-1
        for i=k+1:n
            factor=a(i,k)/a(k,k);
            for j=k+1:n
                a(i,j)=a(i,j)-factor*a(k,j);
            end
            b(i)=b(i)-factor*b(k);
        end
    end
    x(n)=b(n)/a(n,n);
    for i=n-1:-1:1
        sum=b(i);
       for j=i+1:n
           sum=sum-a(i,j)*x(j);
       end
       x(i)=sum/a(i,i);
    end
end
```

列主元消元法

```
function x=ColMajorGauss(a,b)
    [m,n]=size(a);
    for k=1:n-1
        col=a(k:n,k);%取出第k列对应的行值
        col=abs(col);%取绝对值
        [max_value,max_index]=max(col);%获得最大值的索引并+k-1转为真实索引
        if max_index+k-1 ~=k%对比是否为当前行，如果不是则换行
            a([k,max_index+k-1], :) = a([max_index+k-1,k],:);
```

```
            b([k,max_index+k-1])=b([max_index+k-1,k]);%常数项也要交换
        end
        for i=k+1:n%选好主元后再进行高斯消元
            factor=a(i,k)/a(k,k);
            for j=k:n
                a(i,j)=a(i,j)-factor*a(k,j);
            end
            b(i)=b(i)-factor*b(k);
        end
    end
    x(n)=b(n)/a(n,n);
    for i=n-1:-1:1
        sum=b(i);
       for j=i+1:n
           sum=sum-a(i,j)*x(j);
       end
       x(i)=sum/a(i,i);
    end
end
```

实际上由于本题中A的特殊性(对角占优的带状方程组)，几乎不会出现换行的现象，所以列主元消元法的优势在此处并未体现，与原始高斯消元法一样。

Cholesky分解法

本题的系数矩阵同时是个对称负定矩阵，因此与其用LU分解法，用Cholesky分解法更合适。只需在代入时代入-A和-B即可。

$$\begin{cases} a_{ii} = \sum_{k=1}^{i} l_{ik} l_{ik} \\ a_{ij} = \sum_{k=1}^{j} l_{ik} l_{jk} \quad i > j \end{cases} \qquad \begin{cases} l_{jj} = (a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2)^{\frac{1}{2}} \\ l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}) / l_{jj} \end{cases} \quad i = j+1,\ldots,n, j = 1,2,\ldots,n$$

$$y_k = (b_k - \sum_{j=1}^{k-1} l_{kj} y_j) / l_{ii} \quad k = 1,\cdots,n \qquad x_k = (y_k - \sum_{j=k+1}^{n} l_{kj} x_j) / l_{kk} \quad k = n,\cdots,1$$

基于这张原理图可写出代码

```
function x=cholesky(A,b)
    n=length(b);L=zeros(n,n);y=zeros(n,1);x=zeros(n,1);
    for i=1:n%分解出L
        for k=1:i
            sumLk2=0;
            if k==i
                for j=1:k-1
                    sumLk2=sumLk2+L(k,j)^2;
                end
                L(k,k)=sqrt(A(k,k)-sumLk2);
            else
                for j=1:k-1
                    sumLk2=sumLk2+L(i,j)*L(k,j);
```

```
                end
                L(i,k)=(A(i,k)-sumLk2)/L(k,k);
            end
        end
    end
    for i=1:n %前代求解
        sumLy=0;
        for j=1:i-1
            sumLy=sumLy+L(i,j)*y(j);
        end
        y(i)=(b(i)-sumLy)/L(i,i);
    end
    for i=n:-1:1%后代求解
        sumLx=0;
        for j=i+1:n
            sumLx=sumLx+L(j,i)*x(j);
        end
        x(i)=(y(i)-sumLx)/L(i,i);
    end
    x=x';
end
```

Thomas算法

原方程是一个非常典型的三对角方程，用Thomas算法再合适不过。基于如图的原理图可以得到代码。

$$A=\begin{bmatrix}\alpha_1 & & & & \\ \gamma_2 & \alpha_2 & & & \\ & \ddots & \ddots & & \\ & & \gamma_{n-1} & \alpha_{n-1} & \\ & & & \gamma_n & \alpha_n\end{bmatrix}\begin{bmatrix}1 & \beta_1 & & & \\ & 1 & \beta_2 & & \\ & & \ddots & \ddots & \\ & & & 1 & \beta_{n-1} \\ & & & & 1\end{bmatrix}=LU$$

计算公式

（1）分解计算公式A=LU

$$\beta_1 = c_1/b_1$$
$$\beta_i = c_i/(b_i - a_i\beta_{i-1}) \quad i = 2, \ldots, n-1$$

（2）求解Ly=f的递推算式

$$y_1 = f_1/b_1$$
$$y_i = (f_i - a_i y_{i-1})/(b_i - a_i\beta_{i-1}) \quad i = 2, \ldots, n$$

（3）求解Ux=y的递推算式

$$x_n = y_n$$
$$x_i = y_i - \beta_i x_{i+1} \quad i = n-1, \ldots, 1$$

```
function x=thomas(A, f)
    n=size(A,1);
    c=zeros(1,n-1); % 上对角线元素
    b=zeros(1,n);   % 主对角线元素
    a=zeros(1,n-1); % 下对角线元素
    b(:)=diag(A);
```

```
    for i = 1:n-1
        c(i) = A(i+1, i);
    end
    for i = 2:n
        a(i-1) = A(i, i-1);
    end
    be=zeros(1,n-1);%分解A=LU
    be(1)=c(1)/b(1);
    for i=2:n-1
        be(i)=c(i)/(b(i)-a(i-1)*be(i-1));
    end
    y=zeros(1,n);%求解Ly=f
    y(1)=f(1)/b(1);
    for i=2:n
        y(i)=(f(i)-a(i-1)*y(i-1))/(b(i)-a(i-1)*be(i-1));
    end
    x=zeros(1,n);%求解Ux=y
    x(n)=y(n);
    for i=n-1:-1:1
        x(i)=y(i)-be(i)*x(i+1);
    end
end
```

迭代法

雅各比迭代法

$$x_i^{k+1} = \frac{1}{a_{ii}}(b_i - \sum_{j=1,j\neq i}^{n} a_{ij}x_j^k)$$

$$|\varepsilon_a|_i = |\frac{x_i^{k+1} - x_i^k}{x^{k+1}}| \times 100\%$$

基于以上的迭代公式以及近似相对百分比误差，代码如下

```
function [x,k]=jacobi(a,b,x0,epsilon)
    n=size(a,1);
    x=zeros(1,n);
    flag=0;%跳出迭代标志
    k=0;%迭代次数
    while 1
        k=k+1;
        for i=1:n
            sum=0;
            for j=1:n
                if i~=j
                    sum=sum+a(i,j)*x0(j);
                end
            end
```

```
            x(i)=(b(i)-sum)/a(i,i);
            if abs((x(i)-x0(i))/x(i))<epsilon%对每个都判断，即使判断出来了，也先将这一次的都迭代完，再跳出迭代
                flag=1;
            end
            x0(i)=x(i);
        end
        if flag==1
            break;
        end
    end
end
```

其中输入的a是系数矩阵，b是常数矩阵，x0是初始值，epsilon是容差
输出x是解向量，k是迭代次数

高斯-赛德尔迭代法

$$x_i^{k+1}=\frac{1}{a_{ii}}(b_i-\sum_{j=1}^{i-1}a_{ij}x_j^{k+1}-\sum_{j=i+1}^{n}a_{ij}x_j^{k})$$

基于以上的迭代公式,代码如下

```
function [x,k]=GS(a,b,x0,epsilon)
    n=size(a,1);
    x=zeros(1,n);
    flag=0;
    k=0;
    while 1
    k=k+1;
    for i=1:n
        sum1=0;
        sum2=0;
        if i~=1
            for j=1:i-1
                sum1=sum1+a(i,j)*x(j);
            end
        end
        for j=i+1:n
            sum2=sum2+a(i,j)*x0(j);
        end
        x(i)=(b(i)-sum1-sum2)/a(i,i);
        if abs((x(i)-x0(i))/x(i))<epsilon
            flag=1;
        end
        x0(i)=x(i);
    end
    if flag==1
        break;
```

```
    end
  end
  end
```

其中输入的a是系数矩阵，b是常数矩阵，x0是初始值，epsilon是容差
输出x是解向量，k是迭代次数

4 结果分析

为了显著体现不同算法之间的区别，不仅把每个解保留16位小数贴出来，也把解得的X，Y画在图象上，与解析解的图象做对比

所有需要初始值的算法，初始值均为1的向量。
定义解析解

```
syms x;
y_theroy=120*x*x*x/(12*12*24000000)-x*x*x*x/(12*24*24000000)-
x*120*120*120/(12*24*24000000);
```

```
x0=[1,1,1,1,1,1,1,1,1];
y1=Gauss(A_re,B);
y2=ColMajorGauss(A_re,B);
y3=cholesky(A_re,B);
y4=thomas(-A_re,-B);
[y5,k5]=jacobi(A_re,B,x0,eps);
[y6,k6]=GS(A_re,B,x0,eps);
```

4.1 当取点步长为30inch

此时一共有5个点
解为:

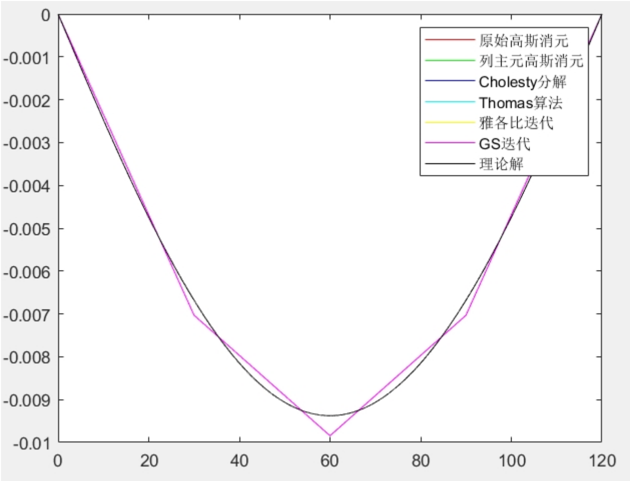
cholesty分解	原始高斯	列主元
0.0000000000000000	0.0000000000000000	0.0000000000000000
-0.0070312500000000	-0.0070312500000000	-0.0070312500000000
-0.0098437500000000	-0.0098437500000000	-0.0098437500000000
-0.0070312500000000	-0.0070312500000000	-0.0070312500000000
0.0000000000000000	0.0000000000000000	0.0000000000000000

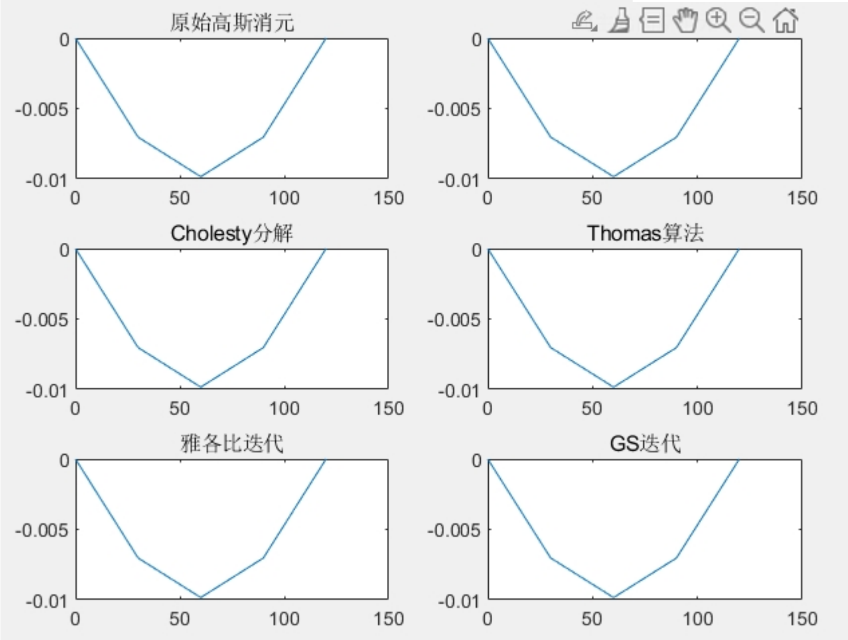
GS迭代	Thomas算法	雅各比迭代
0.0000000000000000	0.0000000000000000	0.0000000000000000
-0.0070312500000000	-0.0070312500000000	-0.0070312500000000
-0.0098437500000000	-0.0098437500000000	-0.0098437500000000
-0.0070312500000000	-0.0070312500000000	-0.0070312500000000
0.0000000000000000	0.0000000000000000	0.0000000000000000

雅各比迭代
60

GS迭代
60

图像为





4.2 当步长取20

此时一共有7个点

解为:

Thomas算法	原始高斯
0.0000000000000000	0.0000000000000000
-0.0048611111111111	-0.0048611111111111
-0.0083333333333333	-0.0083333333333333
-0.0095833333333333	-0.0095833333333333
-0.0083333333333333	-0.0083333333333333
-0.0048611111111111	-0.0048611111111111
0.0000000000000000	0.0000000000000000

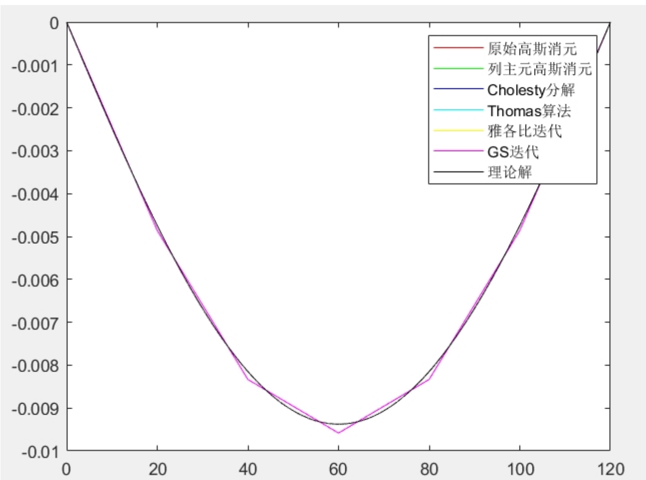
雅各比迭代	列主元
0.0000000000000000	0.0000000000000000
-0.0048611111111111	-0.0048611111111111
-0.0083333333333333	-0.0083333333333333
-0.0095833333333333	-0.0095833333333333
-0.0083333333333333	-0.0083333333333333
-0.0048611111111111	-0.0048611111111111
0.0000000000000000	0.0000000000000000

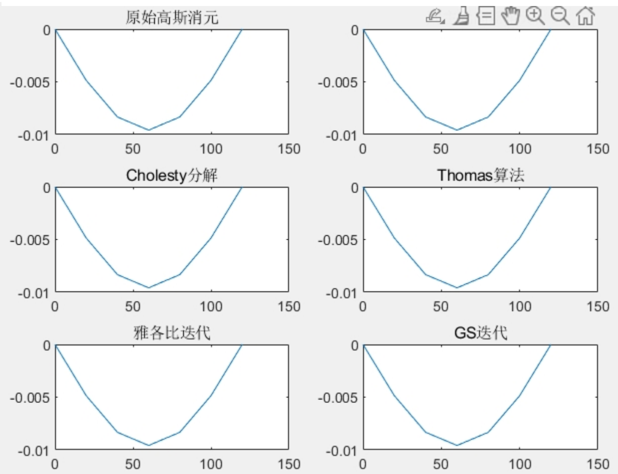
雅各比迭代
137

GS迭代
136

GS迭代	cholesty分解
0.0000000000000000	0.0000000000000000
-0.0048611111111111	-0.0048611111111111
-0.0083333333333333	-0.0083333333333333
-0.0095833333333333	-0.0095833333333333
-0.0083333333333333	-0.0083333333333333
-0.0048611111111111	-0.0048611111111111
0.0000000000000000	0.0000000000000000

图象为:





4.3 当步长取11时

此时共11个点

解为:

原始高斯	cholesty分解	雅各比迭代
0.0000000000000000	0.0000000000000000	0.0000000000000000
-0.0029700000000000	-0.0029700000000000	-0.0029700000000000
-0.0056160000000000	-0.0056160000000000	-0.0056160000000000
-0.0076860000000000	-0.0076860000000000	-0.0076860000000000
-0.0090000000000000	-0.0090000000000000	-0.0090000000000000
-0.0094500000000000	-0.0094500000000000	-0.0094500000000000
-0.0090000000000000	-0.0090000000000000	-0.0090000000000000
-0.0076860000000000	-0.0076860000000000	-0.0076860000000000
-0.0056160000000000	-0.0056160000000000	-0.0056160000000000
-0.0029700000000000	-0.0029700000000000	-0.0029700000000000
0.0000000000000000	0.0000000000000000	0.0000000000000000

列主元	Thomas算法	GS迭代
0.0000000000000000	0.0000000000000000	0.0000000000000000
-0.0029700000000000	-0.0029700000000000	-0.0029700000000000
-0.0056160000000000	-0.0056160000000000	-0.0056160000000000
-0.0076860000000000	-0.0076860000000000	-0.0076860000000000
-0.0090000000000000	-0.0090000000000000	-0.0090000000000000
-0.0094500000000000	-0.0094500000000000	-0.0094500000000000
-0.0090000000000000	-0.0090000000000000	-0.0090000000000000
-0.0076860000000000	-0.0076860000000000	-0.0076860000000000
-0.0056160000000000	-0.0056160000000000	-0.0056160000000000
-0.0029700000000000	-0.0029700000000000	-0.0029700000000000
0.0000000000000000	0.0000000000000000	0.0000000000000000

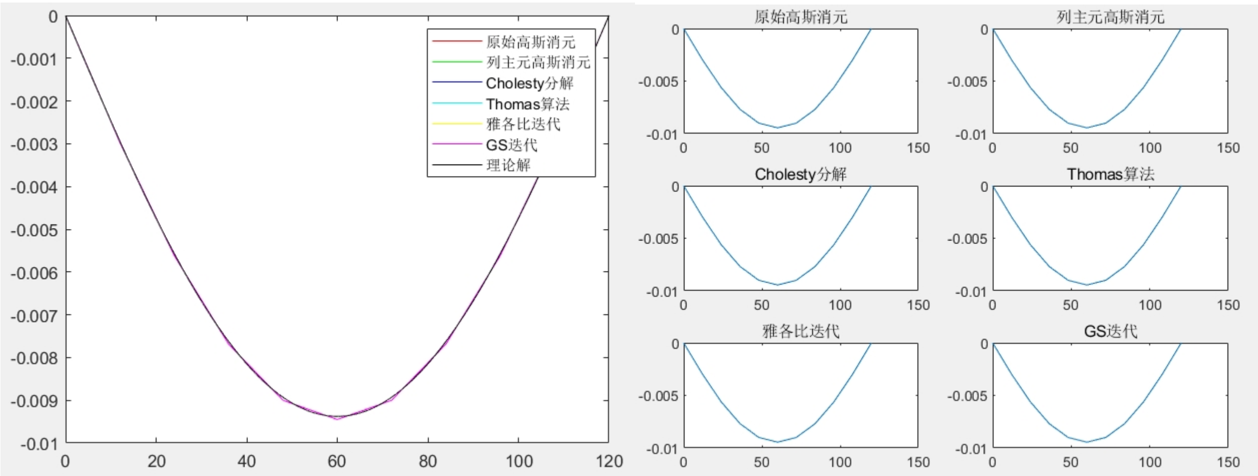
雅各比迭代

384

GS迭代

383

图象为:



4.4 结果分析

由于本题中系数矩阵的特殊性，导致了六种算法得到了完全相同的结果。虽然如此，在写代码的过程中我们仍然可以看出:

高斯消元法(包括列主元)计算量小，但是代码量大；两种分解法只需要代入数学结论即可，数学推导难度大于代码编程难度，时间复杂度小，但是代码量大；迭代法代码量小，但是时间复杂度大，以时间换空间，但是精度优秀。

5 个人心得

对于本题中的矩阵，由于较为典型和简单，使得六种算法并没有体现出显著区别，但是实际应用时势必会有许多没有这么完美的矩阵，甚至出现病态方程组，这时算法的选择就比较重要了；高斯消元法(包括列主元)适用于低阶稠密矩阵，带状矩阵；分解法适用情况与高斯类似，但是相较于高斯，其省略了中间过程，直接由系数矩阵与常数向量计算结果，避免了误差累

积；而Thomas算法只适用于带状矩阵，且效果极佳；两种迭代法，适用于大型稀疏矩阵，而且迭代次数随着元数增大而增大，但是空间不变，以时间换空间，注意可能会出现不收敛的问题。