

第三章作业

Anonymity 3220100000*

* College of Control Science and Engineering, Robotics Engineering

July 8, 2024

1 问题描述

图示为一个多级萃取过程, 待萃取物的流率为 F_1 , 待萃取物的组分为 Y_{in} , 萃取剂的流率为 F_2 , 萃取剂的组分为 X_{in} 。第 i 级的质量守恒方程为

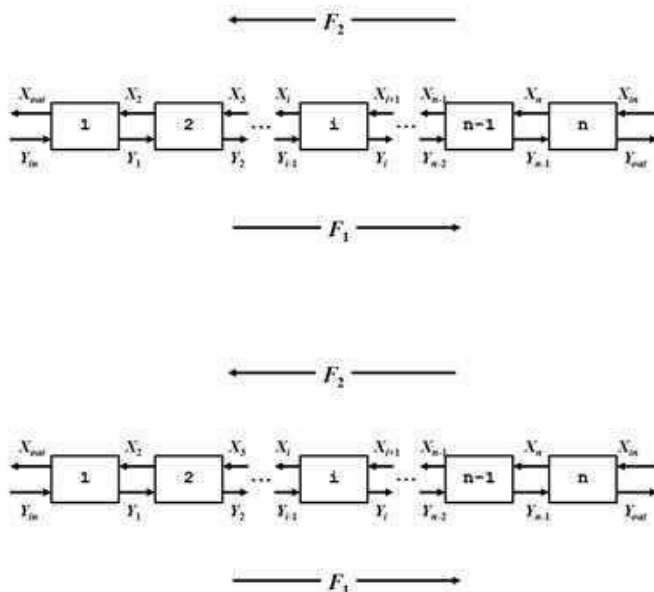
$$F_1 Y_{i-1} + F_2 X_{i+1} = F_1 Y_i + F_2 X_i$$

且每一级均为平衡级, 即 $K = \frac{X_i}{Y_i}$, $K = 4$ 为平衡常数, 因此

$$Y_{i-1} - \left(1 + \frac{F_2}{F_1} K\right) Y_i + \frac{F_2}{F_1} K Y_{i+1} = 0$$

(1) 设 $F_1 = 500 \text{ kg/h}$ $F_2 = 300 \text{ kg/h}$ $Y_{in} = 0.5$ $X_{in} = 0$, 试对不同级数 (3、5、10、20、25、50、100) 的萃取过程, 分别计算 X_{out} 和 Y_{out} , 请用不同方法求解并进行对比 (对第一级和最后一级需要修改方程)。

(2) 设 $F_1 = 500 \text{ kg/h}$ $F_2 = 300 \text{ kg/h}$ $X_{in} = 0$ 、级数 $n = 20$, 若 Y_{in} 分别为 0.3、0.5、0.7、0.9, 计算 X_{out} 和 Y_{out} , 请用不同方法求解并进行对比



2 思路分析

代入数据 $F_1 = 500 \text{ kg/h}$ $F_2 = 300 \text{ kg/h}$ $K = 4$ 得:

$$Y_{i-1} - 3.4Y_i + 2.4Y_{i+1} = 0$$

对于第一级 ($i = 1$):

$$\begin{pmatrix} -3.4 & 2.4 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix} = \begin{pmatrix} -0.5 \\ 0 \end{pmatrix}$$

对于中间级别 ($1 < i < n$):

$$\begin{pmatrix} 1 & -3.4 & 2.4 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} Y_i \\ Y_{i+1} \\ Y_{i+2} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

对于最后一级 ($i = n$):

$$\begin{pmatrix} 1 & -3.4 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} Y_n \\ Y_{\text{out}} \end{pmatrix} = \begin{pmatrix} 0 \\ -0.6X_{\text{in}} \end{pmatrix}$$

整合后可以得到对于任意级数的求解矩阵为:

$$\begin{bmatrix} -3.4 & 2.4 & & & & \\ & 1 & -3.4 & 2.4 & & \\ & & 1 & -3.4 & 2.4 & \\ & & & \cdots & \cdots & \cdots \\ & & & & \cdots & \cdots & \cdots \\ & & & & & 1 & -3.4 & 2.4 \\ & & & & & & 1 & -3.4 \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \\ \cdots \\ Y_i \\ \cdots \\ Y_{n-1} \\ Y_n \end{bmatrix} = \begin{bmatrix} -Y_{\text{in}} \\ 0 \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \end{bmatrix}$$

其中 $X_{\text{out}} = 4Y_1, Y_{\text{out}} = Y_n$

2.1 直接高斯消去法

直接高斯消去方法的过程的是将原始的线性方程组转化为一个上三角形的形式, 然后利用回代过程求解出未知数。

外层循环遍历方程组的每一行, 选定主元后, 内层循环遍历主元下方的每一行, 将它们与主元所在的行进行消元, 通过将主元下方行的元素减去一个倍乘因子乘以主元所在行对应位置的元素来实现。

之后从方程组的最后一行开始, 通过逐步求解未知数, 直到第一行。对于每一行, 计算等式左边的求和部分, 然后用已知的未知数进行修正, 得到当前行的未知数值。

2.2 列主元消去法

列主元消去法通过高斯消元法将矩阵 C 转换为上三角矩阵——对于每一列 k , 从第 k 行到第 n 行, 取出该列元素的绝对值, 并找出绝对值最大的元素 P 及其所在行 u : 如果最大值不在第 k 行, 交换第 k 行和 u 行的位置, 确保最大值所在行在第 k 行; 遍历从第 $k+1$ 行到第 n 行, 将第 i 行的元素消除为 0, 使得第 i 行第 k 列的元素变为 0。

2.3 高斯约当法

高斯-约当消元法要先获取矩阵的维度信息，假设为 $n \times (n+1)$ ，其中 n 表示未知数的个数。

而后通过自上而下的消元步骤将矩阵转化为上三角矩阵，即将矩阵的左下方元素化为 0。再通过自下而上的消元步骤将矩阵转化为对角阵，即将矩阵的上方元素化为 0。最后，将解向量中的值赋为矩阵的最后一列的值，即为方程组的解。

2.4 LU 分解法

LU 分解法利用分解后的下三角矩阵 L 和上三角矩阵 U 求解线性方程组。首先确定输入矩阵的大小，并初始化两个空的矩阵 L 和 U 它们将用于存储 LU 分解后的下三角和上三角矩阵。初始化 U 的第一行为输入矩阵的第一行，并初始化 L 的对角线为 1，以及 L 的第一列为输入矩阵的第一列除以 U 的第一行。

而后通过嵌套的循环，逐步生成 L 和 U 矩阵。分解完成后，利用带入法求解线性方程组。

2.5 矩阵求逆法

矩阵求逆法首先将输入的矩阵 C 赋值给 A ，然后确定矩阵 A 的大小，生成一个与 A 大小相同的单位矩阵 I 。

而后将矩阵 A 和单位矩阵 I 按列连接起来，形成增广矩阵，再使用高斯消元法来求解增广矩阵的行最简形，从经过高斯消去后的增广矩阵中提取出右侧单位矩阵的部分，

2.6 三对角线方程组的追赶法

三对角线方程组的追赶法的 Thomas 算法。首先从输入参数中提取出系数矩阵和常数向量，从而确定方程组的大小，即系数矩阵的行数。

接下来对系数矩阵进行三对角分解，计算出两个数组后利用分解后的两个数组，通过回代的方式求解出方程组的解向量。

2.7 雅可比迭代法

雅可比迭代法接受的是两个输入参数：初始估计值和线性方程组的系数矩阵及右端向量。

程序首先对 N 进行初始化，其中 N 表示迭代系数矩阵，对角线元素置零，其余元素为系数矩阵 C 对应位置元素除以对角线元素。同时初始化 B 为系数矩阵 C 的最后一列。

迭代过程中，利用迭代系数矩阵 N 和右端向量 B 计算下一次迭代的结果，迭代终止条件为相邻两次迭代之间的相对误差小于指定的精度。

2.8 高斯—赛德尔迭代法

高斯-赛德尔迭代方法使用嵌套的两层循环进行迭代，外层循环是迭代次数，内层循环是对每个未知数进行更新。内层循环计算了每个未知数的新值，根据高斯-赛德尔迭代方法的公式计算。在每次迭代中，计算相对误差，用于判断迭代是否收敛。

2.9 SOR 迭代法

逐次超松弛法 (SOR) 求解线性方程组使用两个嵌套循环进行迭代计算。外层循环是迭代次数 k , 内层循环是对每个未知数进行更新计算, 根据逐次超松弛法的迭代公式更新未知数的值, 在每次迭代中计算相邻两次迭代结果之间的相对误差, 如果相对误差小于给定精度, 则跳出迭代。

3 运行结果与分析

3.1 直接高斯消去法

运行结果如下:

```
对于Y_in=0.5, n = 3, X_out = 0.79707622694048041456227338130702264606952667236328  
Y_out = 0.02175426383571180816156598325505910906940698623657    历时 0.002121 秒。  
对于Y_in=0.5, n = 5, X_out = 0.82719630859434845060462748733698390424251556396484  
Y_out = 0.00368221484339098471469386986143490503309294581413    历时 0.000488 秒。  
对于Y_in=0.5, n = 10, X_out = 0.83325665870629139764247383936890400946140289306641  
Y_out = 0.00004600477622521647030201519257275322161149233580    历时 0.000624 秒。  
对于Y_in=0.5, n = 20, X_out = 0.83333332124098313808957527726306580007076263427734  
Y_out = 0.00000000725541015563912961965180247153432702145182    历时 0.000378 秒。  
对于Y_in=0.5, n = 25, X_out = 0.83333333318146951551597112484159879386425018310547  
Y_out = 0.0000000009111833085905058580231832207347707902612    历时 0.003032 秒。  
对于Y_in=0.5, n = 50, X_out = 0.83333333333333337034076748750521801412105560302734  
Y_out = 0.00000000000000000002846588645615292648443510007996    历时 0.000473 秒。  
对于Y_in=0.5, n = 100, X_out = 0.83333333333333337034076748750521801412105560302734  
Y_out = 0.00000000000000000000000000000000000000000000277819437166    历时 0.001478 秒。  
对于n=20, Y_in = 0.3, X_out = 0.49999999274458989395597541260940488427877426147461  
Y_out = 0.00000000435324609338347909528006156776487600712500    历时 0.001802 秒。  
对于n=20, Y_in = 0.5, X_out = 0.83333332124098313808957527726306580007076263427734  
Y_out = 0.00000000725541015563912961965180247153432702145182    历时 0.000402 秒。  
对于n=20, Y_in = 0.7, X_out = 1.16666664973737632671202391065889969468116760253906  
Y_out = 0.00000001015757421789478262556538103438680265000471    历时 0.000623 秒。  
对于n=20, Y_in = 0.9, X_out = 1.49999997823376984840137993160169571638107299804688  
Y_out = 0.00000001305973828015043811302079725632230289278368    历时 0.000458 秒。
```

3.2 列主元消去法

运行结果如下:

3.9.2 $w = 0.5$

[illegible]

3.9.3 $w = 0.8$

[illegible]

3.9.4 $w = 1.0$

[illegible]

3.9.5 $w = 1.2$

[illegible]

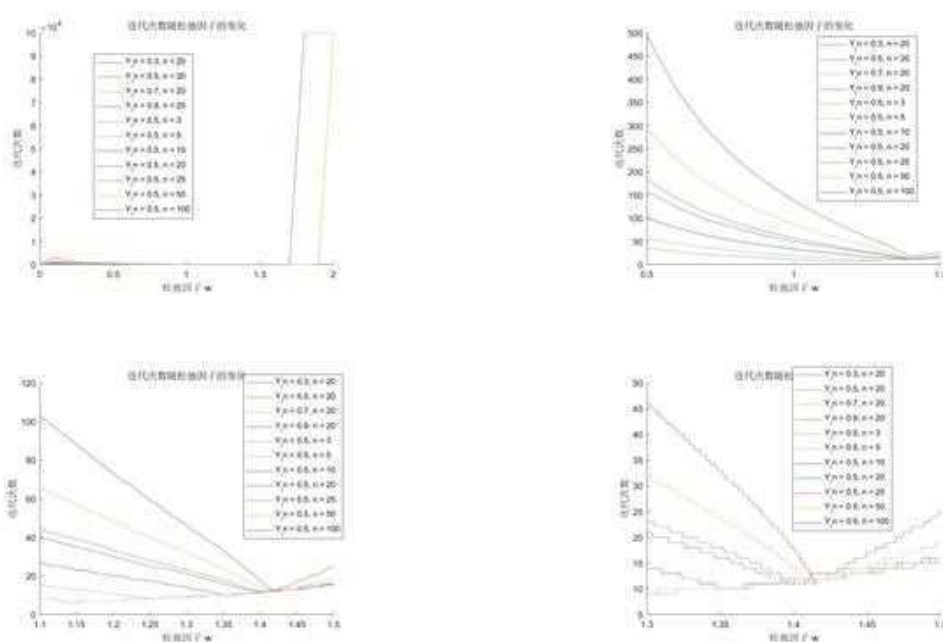
3.9.6 $w = 1.5$

[illegible]

3.9.7 $w = 1.8$

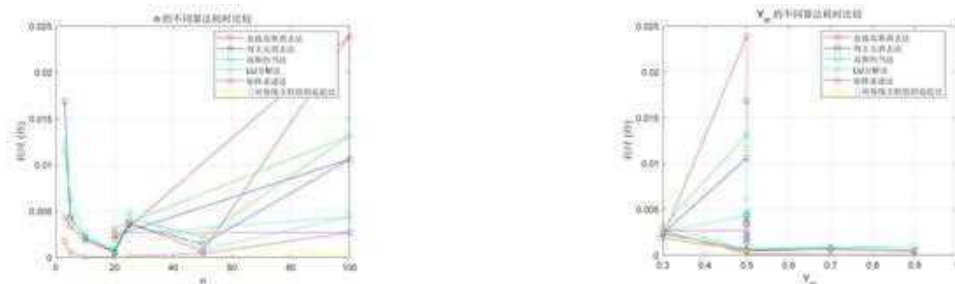
[illegible]

对于 SOR 迭代法, 不同的松弛因子迭代的次数不同, 因此绘制迭代次数随松弛因子的变化曲线。由于 w 趋于 0 和 2 时, 迭代次数较大, 因此分布放大可以得到如下的四幅图像:



3.10 小结

对于以上六种非迭代算法，绘制运行时间随级数 n 和 Y_{in} 变化的图像如下：



分析可得由于求解方程的特殊性，三对角线方程组的追赶法的效率最高，直接高斯消元法，列主元约当法以及高斯约当法等类高斯的方法效率较低。

而对于三种迭代法，由于电脑的算力有限，因此精度无法完全保证。其中对于逐次超松弛迭代法，当 $w = 1$ 时即为高斯-赛德尔迭代法。

其中 SOR 迭代法通过分析图像可知在松弛因子 w 取 1-1.4 时迭代次数较少，且如果 w 的取值不当，会导致迭代次数有明显增加，甚至可能超出算力而无法计算出结果。

总而言之，直接求解法的优点是原理较简单，计算量小、求解速度快，但容易受舍入误差影响，计算精度未必准确（本题中由于运用 *matlab* 中双精度数据类型，因此准确度较高）。

直接求解法比较适用于方程组系数为低阶稠密矩阵、带状矩阵。

而迭代法的计算精度比较优秀，但可能遇到不收敛的情况，且对于松弛因子 w 的选择有所要求。

迭代法比较适用于大型稀疏矩阵。

4 实验心得

本次实验我熟悉了多种求解线性方程组的方法，从最简单的高斯消去法，到 LU 分解、特殊矩阵和矩阵求逆，再到最后的迭代法，我对于线性方程组的认识也层层深入。

特别地，当 SOR 迭代法时，由于没有考虑到电脑算力，没有修改精度甚至导致了电脑的过载。这也提醒我在运行程序前要考虑好算力问题。

由于本次作业涉及到的文件较多，且在完成过程中会发现前面的问题再进行修正，因此使用 *latex* 完成报告可以很大程度地减少图片更迭带来的影响。

5 源代码

5.1 直接高斯消去法函数

```
1 function x = Gauss(C)
2     % 提取系数矩阵 A 和常数向量 b
3     A = C(:, 1:end-1);
4     b = C(:, end);
5
6     % 获取方程组的维度
7     [n, ~] = size(A);
8
9     % 增广矩阵
10    Augmented = [A, b];
11
12    % 高斯消去
13    for i = 1:n-1
14        % 消元操作
15        for j = i+1:n
16            factor = Augmented(j, i) / Augmented(i, i);
17            Augmented(j, :) = Augmented(j, :) - factor * Augmented(i, :);
18        end
19    end
20
21    % 回代求解
22    x = zeros(n, 1);
23    x(n) = Augmented(n, n+1) / Augmented(n, n);
24    for i = n-1:-1:1
25        x(i) = (Augmented(i, n+1) - Augmented(i, i+1:n) * x(i+1:n)) /
                Augmented(i, i);
26    end
27 end
```

5.2 直接高斯消去法主程序

```
1 levels = [3, 5, 10, 20, 25, 50, 100];
2 Y_ins = [0.3, 0.5, 0.7, 0.9];
3
4 for i = 1:length(levels)
5     tic; % 开始计时
6     % 解方程
7     n = levels(i);
8     A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n-1),-1);
9     A(1,1) = -3.4;
10    A(n,n) = -3.4;
11    b = zeros(n, 1);
12    b(1) = -0.5;
13    Y = Gauss([A, b]);
14
15    % 计算输出值
16    X_out = 4 * Y(1);
17    Y_out = Y(n);
18
19    fprintf('对于Y_in=0.5, n = %d, X_out = %.50f\nY_out = %.50f\t', n,
20           X_out, Y_out);
21    toc; % 结束计时并显示执行时间
22 end
23 for i = 1:length(Y_ins)
24     tic; % 开始计时
25     % 解方程
26     n = 20;
27     A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n-1),-1);
28     A(1,1) = -3.4;
29     A(n,n) = -3.4;
30     b = zeros(n, 1);
31     b(1) = -Y_ins(i);
32     Y = Gauss([A, b]);
33
34     % 计算输出值
35     X_out = 4 * Y(1);
36     Y_out = Y(n);
37
```



```

38     fprintf('对于n=20, Y_in = %.1f, X_out = %.50f\n Y_out = %.50f\t',
           Y_ins(i), X_out, Y_out);
39     toc; % 结束计时并显示执行时间
40 end

```

5.3 列主元消去法函数

```

1 function x=Column_principal(C)
2 n=size(C,1);
3 %消去过程
4 for k=1:n
5     a=C(k:n,k);
6     P=max(abs(a));
7     u=find(abs(a)==P);
8     if(u~=1)
9         C([k,(u+k-1)],:)=C([(u+k-1),k],:);
10    end
11    for i=k+1:n
12        factor=C(i,k)/C(k,k);
13        for j=1:(n+1)
14            C(i,j)=C(i,j)-C(k,j)*factor;
15        end
16    end
17 end
18
19 %回代过程
20 x(n)=C(n,(n+1))/C(n,n);
21 for i=(n-1):-1:1
22     sum=C(i,(n+1));
23     for j=i+1:n
24         sum=sum-C(i,j)*x(j);
25     end
26     x(i)=sum/C(i,i);
27 end

```

5.4 列主元消去法主程序

```

1 levels = [3, 5, 10, 20, 25, 50, 100];
2 Y_ins = [0.3, 0.5, 0.7, 0.9];
3
4 for i = 1:length(levels)

```

```

5     tic; % 开始计时
6     % 解方程
7     n = levels(i);
8     A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n-1),-1);
9     A(1,1) = -3.4;
10    A(n,n) = -3.4;
11    b = zeros(n, 1);
12    b(1) = -0.5;
13    Y = Column_principal([A, b]);
14
15    % 计算输出值
16    X_out = 4 * Y(1);
17    Y_out = Y(n);
18
19    fprintf('对于Y_in=0.5, n = %d, X_out = %.50f\nY_out = %.50f\t', n,
20           X_out, Y_out);
21    toc; % 结束计时并显示执行时间
22 end
23 for i = 1:length(Y_ins)
24     tic; % 开始计时
25     % 解方程
26     n = 20;
27     A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n-1),-1);
28     A(1,1) = -3.4;
29     A(n,n) = -3.4;
30     b = zeros(n, 1);
31     b(1) = -Y_ins(i);
32     Y = Column_principal([A, b]);
33
34     % 计算输出值
35     X_out = 4 * Y(1);
36     Y_out = Y(n);
37
38     fprintf('对于n=20, Y_in = %.1f, X_out = %.50f\n Y_out = %.50f\t',
39            Y_ins(i), X_out, Y_out);
40     toc; % 结束计时并显示执行时间
41 end

```

5.5 高斯约当法函数

```
1 function x=Gauss_Jordan(C)
2 n=size(C,1);
3 %自上而下消元
4 for i=1:n
5     factor1=C(i,i);
6     for j=i:(n+1)
7         C(i,j)=C(i,j)/factor1;
8     end
9     for k=(i+1):n
10        factor2=C(k,i);
11        for m=1:(n+1)
12            C(k,m)=C(k,m)-C(i,m)*factor2;
13        end
14    end
15 end
16
17
18 %自下而上消元
19 for k=n:(-1):2
20     for i=(k-1):(-1):1
21         factor3=C(i,k);
22         for j=k:(n+1)
23             C(i,j)=C(i,j)-C((i+1),j)*factor3;
24         end
25     end
26 end
27 for i=1:n
28     x(i)=C(i,n+1);
29 end
```

5.6 高斯约当法主程序

```
1 levels = [3, 5, 10, 20, 25, 50, 100];
2 Y_ins = [0.3, 0.5, 0.7, 0.9];
3
4 for i = 1:length(levels)
5     tic; % 开始计时
6     % 解方程
7     n = levels(i);
```

```

8      A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n
      -1),-1);
9      A(1,1) = -3.4;
10     A(n,n) = -3.4;
11     b = zeros(n, 1);
12     b(1) = -0.5;
13     Y = Gauss_Jordan([A, b]);
14
15     % 计算输出值
16     X_out = 4 * Y(1);
17     Y_out = Y(n);
18
19     fprintf('对于Y_in=0.5, n = %d, X_out = %.50f\nY_out = %.50f\t', n,
      X_out, Y_out);
20     toc; % 结束计时并显示执行时间
21 end
22
23 for i = 1:length(Y_ins)
24     tic; % 开始计时
25     % 解方程
26     n = 20;
27     A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n
      -1),-1);
28     A(1,1) = -3.4;
29     A(n,n) = -3.4;
30     b = zeros(n, 1);
31     b(1) = -Y_ins(i);
32     Y = Gauss_Jordan([A, b]);
33
34     % 计算输出值
35     X_out = 4 * Y(1);
36     Y_out = Y(n);
37
38     fprintf('对于n=20, Y_in = %.1f, X_out = %.50f\n Y_out = %.50f\t',
      Y_ins(i), X_out, Y_out);
39     toc; % 结束计时并显示执行时间
40 end

```

5.7 LU 分解法函数

```

1 function x=Doolittle(C)

```

```

2  n=size(C,1);
3  L=zeros(n);
4  U=zeros(n);
5
6  %U为n*n上三角矩阵，初始化U第一行
7  U(1,:)=C(1,1:n);
8  %L为n*n下三角矩阵，初始化L对角线及第一列
9  for i=1:n
10     L(i,i)=1;
11 end
12 for i=2:n
13     L(i,1)=C(i,1)/U(1,1);
14 end
15
16 %Doolittle 分解
17 %生成L、U
18 for k=2:n
19     for i=k:n
20         sum2=0;
21         for j=1:(k-1)
22             sum2=sum2+L(k,j)*U(j,i);
23         end
24         U(k,i)=C(k,i)-sum2; %产生第k行U
25         for j=1:(k-1)
26             sum2=sum2+L(i,j)*U(j,k);
27         end
28         L(i,k)=(C(i,k)-sum2)/U(k,k); %产生第k列L
29     end
30 end
31
32 %向前带入求解Y
33 y(1)=C(1,n+1);
34 for i=2:n
35     sum1=0;
36     for j=1:i-1
37         sum1=sum1+L(i,j)*y(j);
38     end
39     y(i)=C(i,n+1)-sum1;
40 end
41
42 %向后代入求解X

```



```

43 x(n)=y(n)/U(n,n);
44 for k=(n-1):-1:1
45     sum2=0;
46     for j=k+1:n
47         sum2=sum2+U(k,j)*x(j);
48     end
49     x(k)=(y(k)-sum2)/U(k,k);
50 end

```

5.8 LU 分解法主程序

```

1  levels = [3, 5, 10, 20, 25, 50, 100];
2  Y_ins = [0.3, 0.5, 0.7, 0.9];
3
4  for i = 1:length(levels)
5      tic; % 开始计时
6      % 解方程
7      n = levels(i);
8      A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n-1),-1);
9      A(1,1) = -3.4;
10     A(n,n) = -3.4;
11     b = zeros(n, 1);
12     b(1) = -0.5;
13     Y = Doolittle([A, b]);
14
15     % 计算输出值
16     X_out = 4 * Y(1);
17     Y_out = Y(n);
18
19     fprintf('对于Y_in=0.5, n = %d, X_out = %.5f\nY_out = %.5f\t', n,
20             X_out, Y_out);
21     toc; % 结束计时并显示执行时间
22 end
23 for i = 1:length(Y_ins)
24     tic; % 开始计时
25     % 解方程
26     n = 20;
27     A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n-1),-1);

```

```

28     A(1,1) = -3.4;
29     A(n,n) = -3.4;
30     b = zeros(n, 1);
31     b(1) = -Y_ins(i);
32     Y = Doolittle([A, b]);
33
34     % 计算输出值
35     X_out = 4 * Y(1);
36     Y_out = Y(n);
37
38     fprintf('对于n=20, Y_in = %.1f, X_out = %.50f\n Y_out = %.50f\t',
39           Y_ins(i), X_out, Y_out);
40 end

```

5.9 矩阵求逆法函数

```

1 function x = Inverse(C)
2     % 提取系数矩阵 A 和单位矩阵 I
3     A = C;
4     n = size(A, 1);
5     I = eye(n);
6
7     % 增广矩阵
8     Augmented = [A, I];
9
10    % 高斯消去
11    for i = 1:n
12        % 主元归一化
13        factor = Augmented(i, i);
14        Augmented(i, :) = Augmented(i, :) / factor;
15        % 消元操作
16        for j = 1:n
17            if j ~= i
18                factor = Augmented(j, i) / Augmented(i, i);
19                Augmented(j, :) = Augmented(j, :) - factor * Augmented
20                    (i, :);
21            end
22        end
23    end

```

```

24 % 提取逆矩阵部分
25 Inverse_A = Augmented(:, n+1:end);
26
27 % 返回逆矩阵
28 x = Inverse_A;
29 end

```

5.10 矩阵求逆主程序

```

1 levels = [3, 5, 10, 20, 25, 50, 100];
2 Y_ins = [0.3, 0.5, 0.7, 0.9];
3
4 for i = 1:length(levels)
5     tic; % 开始计时
6     % 解方程
7     n = levels(i);
8     A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n-1),-1);
9     A(1,1) = -3.4;
10    A(n,n) = -3.4;
11    b = zeros(n, 1);
12    b(1) = -0.5;
13    Y = Inverse([A, b]);
14
15    % 计算输出值
16    X_out = 4 * Y(1);
17    Y_out = Y(n);
18
19    fprintf('对于Y_in=0.5, n = %d, X_out = %.5f\nY_out = %.5f\t', n,
20           X_out, Y_out);
21    toc; % 结束计时并显示执行时间
22 end
23
24 for i = 1:length(Y_ins)
25     tic; % 开始计时
26     % 解方程
27     n = 20;
28     A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n-1),-1);
29     A(1,1) = -3.4;
30     A(n,n) = -3.4;

```

```

30     b = zeros(n, 1);
31     b(1) = -Y_ins(i);
32     Y = Inverse([A, b]);
33
34     % 计算输出值
35     X_out = 4 * Y(1);
36     Y_out = Y(n);
37
38     fprintf('对于n=20, Y_in = %.1f, X_out = %.50f\n Y_out = %.50f\t',
39           Y_ins(i), X_out, Y_out);
40 end

```

5.11 三对角线方程组的追赶法函数

```

1 function x = Thomas(C)
2     % 提取系数矩阵 A 和常数向量 b
3     A = C(:, 1:end-1);
4     b = C(:, end);
5
6     % 获取方程组的维度
7     n = size(A, 1);
8
9     % 分解系数矩阵
10    alpha = zeros(n, 1);
11    beta = zeros(n, 1);
12
13    alpha(1) = A(1, 1);
14    beta(1) = b(1) / alpha(1);
15
16    for i = 2:n
17        alpha(i) = A(i, i) - A(i, i-1) * A(i-1, i) / alpha(i-1);
18        beta(i) = (b(i) - A(i, i-1) * beta(i-1)) / alpha(i);
19    end
20
21    % 回代求解
22    x = zeros(n, 1);
23    x(n) = beta(n);
24
25    for i = n-1:-1:1
26        x(i) = beta(i) - A(i, i+1) / alpha(i) * x(i+1);

```

```
27     end
28 end
```

5.12 三对角线方程组的追赶法主程序

```
1  levels = [3, 5, 10, 20, 25, 50, 100];
2  Y_ins = [0.3, 0.5, 0.7, 0.9];
3
4  for i = 1:length(levels)
5      tic; % 开始计时
6      % 解方程
7      n = levels(i);
8      A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n-1),-1);
9      A(1,1) = -3.4;
10     A(n,n) = -3.4;
11     b = zeros(n, 1);
12     b(1) = -0.5;
13     Y = Thomas([A, b]);
14
15     % 计算输出值
16     X_out = 4 * Y(1);
17     Y_out = Y(n);
18
19     fprintf('对于Y_in=0.5, n = %d, X_out = %.50f\nY_out = %.50f\t', n,
20             X_out, Y_out);
21     toc; % 结束计时并显示执行时间
22 end
23 for i = 1:length(Y_ins)
24     tic; % 开始计时
25     % 解方程
26     n = 20;
27     A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n-1),-1);
28     A(1,1) = -3.4;
29     A(n,n) = -3.4;
30     b = zeros(n, 1);
31     b(1) = -Y_ins(i);
32     Y = Thomas([A, b]);
33
```



```

34 % 计算输出值
35 X_out = 4 * Y(1);
36 Y_out = Y(n);
37
38 fprintf('对于n=20, Y_in = %.1f, X_out = %.50f\n Y_out = %.50f\t',
        Y_ins(i), X_out, Y_out);
39 toc; % 结束计时并显示执行时间
40 end

```

5.13 雅可比迭代法函数

```

1 function x = Jacobi(T1, C)
2 % 设置迭代次数上限
3 max_iter = 1000;
4
5 % 确定问题维度
6 n = length(T1);
7
8 % 初始化迭代
9 x = T1;
10
11 % 迭代
12 for iter = 1:max_iter
13     x_new = zeros(n, 1);
14     for i = 1:n
15         x_new(i) = (C(i, end) - C(i, 1:end-1) * x + C(i, i) * x(i)) /
            C(i, i);
16     end
17
18 % 检查是否收敛
19 if norm(x_new - x) < 1e-10
20     x = x_new;
21     fprintf('迭代次数: %d\t', iter);
22     return;
23 end
24
25 % 更新解
26 x = x_new;
27 end
28
29 error('未达到收敛条件');

```

5.14 雅可比迭代法主程序

```
1 levels = [3, 5, 10, 20, 25, 50, 100];
2 Y_ins = [0.3, 0.5, 0.7, 0.9];
3
4 for i = 1:length(levels)
5     tic; % 开始计时
6     % 解方程
7     n = levels(i);
8     A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n-1),-1);
9     A(1,1) = -3.4;
10    A(n,n) = -3.4;
11    b = zeros(n, 1);
12    b(1) = -0.5;
13    x = Jacobi(zeros(n,1), [A, b]); % 使用 Jacobi 方法求解
14
15    % 计算输出值
16    X_out = 4 * x(1);
17    Y_out = x(n);
18
19    fprintf('对于Y_in=0.5, n = %d, X_out = %.80f\nY_out = %.80f\t', n,
20           X_out, Y_out);
21    toc; % 结束计时并显示执行时间
22 end
23 for i = 1:length(Y_ins)
24     tic; % 开始计时
25     % 解方程
26     n = 20;
27     A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n-1),-1);
28     A(1,1) = -3.4;
29     A(n,n) = -3.4;
30     b = zeros(n, 1);
31     b(1) = -Y_ins(i);
32     x = Jacobi(zeros(n,1), [A, b]); % 使用 Jacobi 方法求解
33
34     % 计算输出值
35     X_out = 4 * x(1);
36     Y_out = x(n);
37
```

```

38     fprintf('对于n=20, Y_in = %.1f, X_out = %.80f\n Y_out = %.80f\t',
           Y_ins(i), X_out, Y_out);
39     toc; % 结束计时并显示执行时间
40 end

```

5.15 高斯—赛德尔迭代法函数

```

1 function [x, iter] = Gauss_Seidel(T1,C)
2 n=size(C,1);
3 maxn=1e10; %最大迭代次数
4 espa=1e-40; %精度
5 x1=T1; %赋初始值
6 x2=zeros(n,1); %创建向量x2保存迭代结果
7
8 %开始迭代
9 for k=1:maxn
10     for i=1:n
11         sum1=0;
12         sum2=0;
13         for j=1:(i-1)
14             sum1=sum1+C(i,j)*x2(j);
15         end
16         for j=(i+1):n
17             sum2=sum2+C(i,j)*x1(j);
18         end
19         x2(i)=(C(i,n+1)-sum1-sum2)/C(i,i);
20     end
21     %计算相对误差
22     z=0;
23     for m=1:n
24         z1=abs((x2(m)-x1(m))/x2(m));
25         if z1>z
26             z=z1;
27         end
28     end
29     if z<espa %迭代终止准则
30         break;
31     end
32     x1=x2;
33 end
34 if k==maxn

```

```

35     fprintf('迭代次数达到上限\n');
36     x=zeros(1,n);
37     iter = maxn;
38 else
39     x=transpose(x2);
40     iter = k;
41 end

```

5.16 高斯—赛德尔迭代法主程序

```

1  levels = [3, 5, 10, 20, 25, 50, 100];
2  Y_ins = [0.3, 0.5, 0.7, 0.9];
3
4  for i = 1:length(levels)
5      tic; % 开始计时
6      % 解方程
7      n = levels(i);
8      A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n-1),-1);
9      A(1,1) = -3.4;
10     A(n,n) = -3.4;
11     b = zeros(n, 1);
12     b(1) = -0.5;
13     [x, iter] = Gauss_Seidel(zeros(n,1), [A, b]);
14     fprintf('迭代次数: %d\t', iter);
15     % 计算输出值
16     X_out = 4 * x(1);
17     Y_out = x(n);
18
19     fprintf('对于Y_in=0.5, n = %d, X_out = %.80f\nY_out = %.80f\t', n,
20             X_out, Y_out);
21     toc; % 结束计时并显示执行时间
22 end
23 for i = 1:length(Y_ins)
24     tic; % 开始计时
25     % 解方程
26     n = 20;
27     A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n-1),-1);
28     A(1,1) = -3.4;

```

```

29     A(n,n) = -3.4;
30     b = zeros(n, 1);
31     b(1) = -Y_ins(i);
32     [x, iter] = Gauss_Seidel(zeros(n,1), [A, b]);
33     fprintf('迭代次数: %d\t', iter);
34     % 计算输出值
35     X_out = 4 * x(1);
36     Y_out = x(n);
37
38     fprintf('对于n=20, Y_in = %.1f, X_out = %.80f\n Y_out = %.80f\t',
39           Y_ins(i), X_out, Y_out);
40     toc; % 结束计时并显示执行时间
end

```

5.17 SOR 迭代法函数

```

1 function x = SOR(T1, C)
2     n = size(C, 1);
3     maxn = 1e10; % 最大迭代次数
4     espa = 1e-10; % 精度
5     x1 = T1; % 赋初始值
6     x2 = zeros(n, 1); % 创建向量x2保存迭代结果
7     w = 1.0;
8
9     % 开始迭代
10    for k = 1:maxn
11        for i = 1:n
12            sum1 = 0;
13            sum2 = 0;
14            for j = 1:(i - 1)
15                sum1 = sum1 + C(i, j) * x2(j);
16            end
17            for j = (i + 1):n
18                sum2 = sum2 + C(i, j) * x1(j);
19            end
20            x2(i) = (1 - w) * x1(i) + w * (C(i, n + 1) - sum1 - sum2)
21                / C(i, i);
22        end
23        % 计算相对误差
24        z = 0;
25        for m = 1:n

```



```

25         z1 = abs((x2(m) - x1(m)) / x2(m));
26         if z1 > z
27             z = z1;
28         end
29     end
30     if z < espa % 迭代终止准则
31         break;
32     end
33     x1 = x2;
34 end
35 if k == maxn
36     fprintf('迭代次数达到上限\n');
37     x = zeros(1, n);
38 else
39     fprintf('迭代次数: %d\t', k);
40     x = transpose(x2);
41 end
42 end

```

5.18 SOR 迭代法主程序

```

1 levels = [3, 5, 10, 20, 25, 50, 100];
2 Y_ins = [0.3, 0.5, 0.7, 0.9];
3
4 for i = 1:length(levels)
5     tic; % 开始计时
6     % 解方程
7     n = levels(i);
8     A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n-1),-1);
9     A(1,1) = -3.4;
10    A(n,n) = -3.4;
11    b = zeros(n, 1);
12    b(1) = -0.5;
13    x = SOR(zeros(n,1), [A, b]); % 使用 SOR 方法求解
14
15    % 计算输出值
16    X_out = 4 * x(1);
17    Y_out = x(n);
18
19    fprintf('对于Y_in=0.5, n = %d, X_out = %.80f\nY_out = %.80f\t', n,

```

```

        X_out, Y_out);
20     toc; % 结束计时并显示执行时间
21 end
22
23 for i = 1:length(Y_ins)
24     tic; % 开始计时
25     % 解方程
26     n = 20;
27     A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones(1,n-1),-1);
28     A(1,1) = -3.4;
29     A(n,n) = -3.4;
30     b = zeros(n, 1);
31     b(1) = -Y_ins(i);
32     x = SOR(zeros(n,1), [A, b]); % 使用 SOR 方法求解
33
34     % 计算输出值
35     X_out = 4 * x(1);
36     Y_out = x(n);
37
38     fprintf('对于n=20, Y_in = %.1f, X_out = %.80f\n Y_out = %.80f\t',
        Y_ins(i), X_out, Y_out);
39     toc; % 结束计时并显示执行时间
40 end

```

5.19 松弛因子与迭代次数关系函数

```

1
2 function [x, iteration_count] = SORplus(T1, C, w)
3     n = size(C, 1);
4     maxn = 10e4; % 最大迭代次数
5     espa = 1e-4; % 精度
6     x1 = T1; % 赋初始值
7     x2 = zeros(n, 1); % 创建向量x2保存迭代结果
8
9     % 开始迭代
10    for iteration_count = 1:maxn
11        for i = 1:n
12            sum1 = 0;
13            sum2 = 0;
14            for j = 1:(i - 1)

```

```

15         sum1 = sum1 + C(i, j) * x2(j);
16     end
17     for j = (i + 1):n
18         sum2 = sum2 + C(i, j) * x1(j);
19     end
20     x2(i) = (1 - w) * x1(i) + w * (C(i, n + 1) - sum1 - sum2)
        / C(i, i);
21 end
22 % 计算相对误差
23 z = 0;
24 for m = 1:n
25     z1 = abs((x2(m) - x1(m)) / x2(m));
26     if z1 > z
27         z = z1;
28     end
29 end
30 if z < espa % 迭代终止准则
31     break;
32 end
33 x1 = x2;
34 end
35
36 if iteration_count == maxn
37     fprintf('迭代次数达到上限\n');
38     x = zeros(1, n);
39 else
40     fprintf('迭代次数: %d\t', iteration_count);
41     x = transpose(x2);
42 end
43 end

```

5.20 松弛因子与迭代次数关系主程序

```

1
2     levels = [3, 5, 10, 20, 25, 50, 100];
3     Y_ins = [0.3, 0.5, 0.7, 0.9];
4     w_values = 1.3:0.001:1.5;
5
6     % 存储迭代次数的矩阵
7     iteration_counts_Yin = zeros(length(levels), length(w_values));
8     iteration_counts_n = zeros(length(Y_ins), length(w_values));

```

```

9
10 % 对每个Y_in进行迭代
11 for i = 1:length(Y_ins)
12     Y_in = Y_ins(i);
13     n = 20;
14     A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones
        (1,n-1),-1);
15     A(1,1) = -3.4;
16     A(n,n) = -3.4;
17     b = zeros(n, 1);
18     b(1) = -Y_in;
19
20     for j = 1:length(w_values)
21         w = w_values(j);
22         [~, iteration_count] = SORplus(zeros(n,1), [A, b], w);
23         iteration_counts_n(i, j) = iteration_count;
24     end
25 end
26
27 % 对每个n进行迭代
28 for i = 1:length(levels)
29     n = levels(i);
30     A = diag(-3.4*ones(1,n)) + diag(2.4*ones(1,n-1),1) + diag(ones
        (1,n-1),-1);
31     A(1,1) = -3.4;
32     A(n,n) = -3.4;
33     b = zeros(n, 1);
34     b(1) = -0.5;
35
36     for j = 1:length(w_values)
37         w = w_values(j);
38         [~, iteration_count] = SORplus(zeros(n,1), [A, b], w);
39         iteration_counts_Yin(i, j) = iteration_count;
40     end
41 end
42
43 % 绘制迭代次数随松弛因子的变化曲线（对每个Y_in和n分别绘制）
44 figure;
45 hold on;
46 for i = 1:length(Y_ins)
47     plot(w_values, iteration_counts_n(i, :), 'DisplayName',

```

```

        sprintf('Y_in = %.1f, n = 20', Y_ins(i)));
48     end
49     for i = 1:length(levels)
50         plot(w_values, iteration_counts_Yin(i, :), 'DisplayName',
            sprintf('Y_in = 0.5, n = %d', levels(i)));
51     end
52     hold off;
53     xlabel('松弛因子 w');
54     ylabel('迭代次数');
55     title('迭代次数随松弛因子的变化');
56     legend('Location', 'best');

```

5.21 运行时间随级数 n 和 Y_{in} 图像绘制

```

1  % 绘制 n 的图像
2  figure;
3  plot(n_direct, time_direct, 'ro-', 'DisplayName', '直接高斯消去法');
4  hold on;
5
6  plot(n_column, time_column, 'bo-', 'DisplayName', '列主元消去法');
7
8  plot(n_jordan, time_jordan, 'go-', 'DisplayName', '高斯约当法');
9
10 plot(n_lu, time_lu, 'co-', 'DisplayName', 'LU分解法');
11
12 plot(n_inverse, time_inverse, 'mo-', 'DisplayName', '矩阵求逆法');
13
14 plot(n_tridiagonal, time_tridiagonal, 'yo-', 'DisplayName', '三对角线
    方程组的追赶法');
15
16 xlabel('n');
17 ylabel('耗时 (秒)');
18 title('n 的不同算法耗时比较');
19 legend('Location', 'best');
20 grid on;
21 hold off;
22 saveas(gcf, 'n_comparison.png');
23
24 % 绘制 Y_in 的图像
25 figure;
26 plot(Y_in_direct, time_direct, 'ro-', 'DisplayName', '直接高斯消去法')

```

```

    ;
27 hold on;
28
29 plot(Y_in_column, time_column, 'bo-', 'DisplayName', '列主元消去法');
30
31 plot(Y_in_jordan, time_jordan, 'go-', 'DisplayName', '高斯约当法');
32
33 plot(Y_in_lu, time_lu, 'co-', 'DisplayName', 'LU分解法');
34
35 plot(Y_in_inverse, time_inverse, 'mo-', 'DisplayName', '矩阵求逆法');
36
37 plot(Y_in_tridiagonal, time_tridiagonal, 'yo-', 'DisplayName', '三对角
    线方程组的追赶法');
38
39 xlabel('Y_{in}');
40 ylabel('耗时 (秒)');
41 title('Y_{in} 的不同算法耗时比较');
42 legend('Location', 'best');
43 grid on;
44 hold off;
45 saveas(gcf, 'Y_in_comparison.png');

```