

学号 WA2214014 专业 人工智能 姓名 杨跃浙  
实验日期 24.12.20 教师签字 成绩

# 实验报告

【实验名称】 实验二——线性方程的解法

## 【实验目的】

### 1. 掌握线性方程组的数值解法

通过实验，学习和实现常用的线性方程组数值解法，包括列主元消去法、全主元消去法、LU 分解法、追赶法和 Gauss-Seidel 法。

### 2. 理解数值方法的基本原理

理解不同算法的核心思想与计算过程，并通过编程实现这些算法，进一步熟悉它们的适用场景与优缺点。

### 3. 验证算法的稳定性与效率

比较不同方法在求解线性方程组中的稳定性、计算效率和精度，培养数值计算的分析能力。

### 4. 提高编程与实际问题解决能力

通过 MATLAB 编程，实现各类数值算法，提升工程计算能力，并将理论应用于实际问题求解。

## 5. 增强数值计算结果的可视化与解释能力

在每一步算法的实现过程中，清晰显示解题步骤和中间结果，帮助理解算法的解题过程，并为后续实验分析提供依据。

### 【实验内容】

## 1. 编程实现列主元或全主元消去法，求解下列方程组。

$$\begin{pmatrix} 1 & 2 & 3 \\ 5 & 4 & 10 \\ 3 & -0.1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}$$

代码：（列主元消去法）

```

1 % 任务 1: 列主元消去法
2 % 实验人: 杨跃浙
3 A = [1 2 3; 5 4 10; 3 -0.1 1];
4 b = [1; 0; 2];
5
6 n = length(b);
7
8 disp('初始矩阵 A 和向量 b:');
9 disp(A);
10 disp(b);
11
12 for k = 1:n-1
13     % 找到列主元并交换
14     [~, maxIndex] = max(abs(A(k:n, k)));
15     maxIndex = maxIndex + k - 1;
16     if maxIndex ~= k
17         % 交换矩阵行
18         A([k, maxIndex], :) = A([maxIndex, k], :);
19         b([k, maxIndex]) = b([maxIndex, k]);
20         disp(['第 ', num2str(k), ' 列主元交换:']);
21         disp('矩阵 A:');
22         disp(A);
23         disp('向量 b:');
24     end
25 end

```

0  
2.0000  
-1.4000  
迭代第 1 行后 x:  
1.2000  
2.0000  
-1.4000  
列主元消去法最终解:  
1.2000  
2.0000  
-1.4000

% 任务 1: 列主元消去法

% 实验人: 杨跃浙

A = [1 2 3; 5 4 10; 3 -0.1 1];

b = [1; 0; 2];

n = length(b);

disp('初始矩阵 A 和向量 b:');

disp(A);

disp(b);

for k = 1:n-1

% 找到列主元并交换

[~, maxIndex] = max(abs(A(k:n, k)));

maxIndex = maxIndex + k - 1;

if maxIndex ~= k

% 交换矩阵行

A([k, maxIndex], :) = A([maxIndex, k], :);

b([k, maxIndex]) = b([maxIndex, k]);

disp(['第 ', num2str(k), ' 列主元交换:']);

disp('矩阵 A:');

disp(A);

disp('向量 b:');

```

disp(b);
end
% 消元过程
for i = k+1:n
    factor = A(i, k) / A(k, k);
    A(i, k:n) = A(i, k:n) - factor * A(k, k:n);
    b(i) = b(i) - factor * b(k);
end
% 显示消元后的矩阵和向量
disp(['第 ', num2str(k), ' 列消元后:']);
disp('矩阵 A:');
disp(A);
disp('向量 b:');
disp(b);
end

% 回代求解
x1 = zeros(n, 1);
for i = n:-1:1
    x1(i) = (b(i) - A(i, i+1:n) * x1(i+1:n)) / A(i, i);
% 显示每一步回代结果
disp(['回代第 ', num2str(i), ' 行后 x:']);
disp(x1);
end

disp('列主元消去法最终解:');
disp(x1);

```

## 结果:

```
>> code1_yyz
```

初始矩阵 A 和向量 b:

|        |         |         |
|--------|---------|---------|
| 1.0000 | 2.0000  | 3.0000  |
| 5.0000 | 4.0000  | 10.0000 |
| 3.0000 | -0.1000 | 1.0000  |

1  
0  
2

第 1 列主元交换:

矩阵 A:

|        |        |         |
|--------|--------|---------|
| 5.0000 | 4.0000 | 10.0000 |
|--------|--------|---------|

|        |         |        |
|--------|---------|--------|
| 1.0000 | 2.0000  | 3.0000 |
| 3.0000 | -0.1000 | 1.0000 |

向量 b:

0  
1  
2

第 1 列消元后:

矩阵 A:

|        |         |         |
|--------|---------|---------|
| 5.0000 | 4.0000  | 10.0000 |
| 0      | 1.2000  | 1.0000  |
| 0      | -2.5000 | -5.0000 |

向量 b:

0  
1  
2

第 2 列主元交换:

矩阵 A:

|        |         |         |
|--------|---------|---------|
| 5.0000 | 4.0000  | 10.0000 |
| 0      | -2.5000 | -5.0000 |
| 0      | 1.2000  | 1.0000  |

向量 b:

0  
2  
1

第 2 列消元后:

矩阵 A:

|        |         |         |
|--------|---------|---------|
| 5.0000 | 4.0000  | 10.0000 |
| 0      | -2.5000 | -5.0000 |
| 0      | 0       | -1.4000 |

向量 b:

0  
2.0000  
1.9600

回代第 3 行后 x:

0  
0  
-1.4000

回代第 2 行后 x:

0  
2.0000  
-1.4000

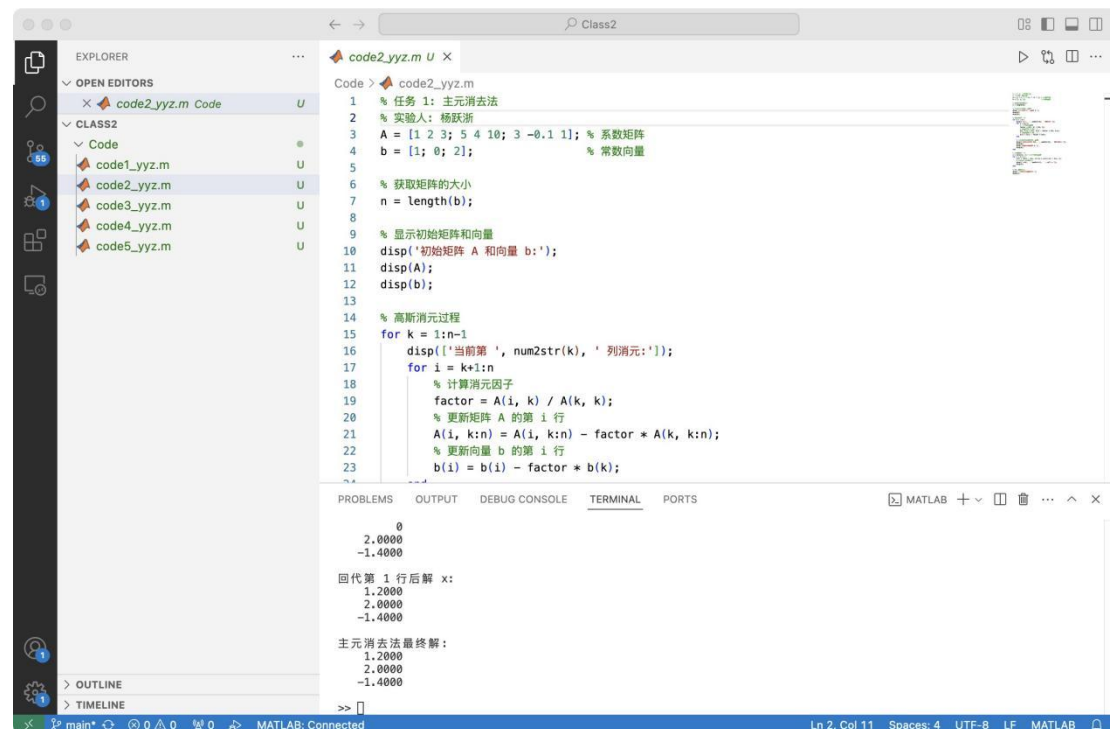
回代第 1 行后 x:

1.2000  
2.0000  
-1.4000

列主元消去法最终解:

1.2000  
2.0000  
-1.4000

## 代码：（全主元消去法）



```

1 % 任务 1: 主元消去法
2 % 实验人: 杨跃浙
3 A = [1 2 3; 5 4 10; 3 -0.1 1]; % 系数矩阵
4 b = [1; 0; 2]; % 常数向量
5
6 % 获取矩阵的大小
7 n = length(b);
8
9 % 显示初始矩阵和向量
10 disp('初始矩阵 A 和向量 b:');
11 disp(A);
12 disp(b);
13
14 % 高斯消元过程
15 for k = 1:n-1
16     disp(['当前第 ', num2str(k), ' 列消元:']);
17     for i = k+1:n
18         % 计算消元因子
19         factor = A(i, k) / A(k, k);
20         % 更新矩阵 A 的第 i 行
21         A(i, k:n) = A(i, k:n) - factor * A(k, k:n);
22         % 更新向量 b 的第 i 行
23         b(i) = b(i) - factor * b(k);
24     end
25 end
26
27 % 回代求解
28 x = zeros(n, 1);
29 x(n) = b(n) / A(n, n);
30 for i = n-1:-1:1
31     x(i) = (b(i) - A(i, i+1:n) * x(i+1:n)) / A(i, i);
32 end
33
34 % 显示最终解
35 disp('主元消去法最终解:');
36 disp(x);

```

0  
2.0000  
-1.4000

回代第 1 行后解 x:

1.2000  
2.0000  
-1.4000

主元消去法最终解:

1.2000  
2.0000  
-1.4000

% 任务 1: 主元消去法

% 实验人: 杨跃浙

A = [1 2 3; 5 4 10; 3 -0.1 1]; % 系数矩阵

b = [1; 0; 2]; % 常数向量

% 获取矩阵的大小

n = length(b);

% 显示初始矩阵和向量

disp('初始矩阵 A 和向量 b:');

disp(A);

disp(b);

% 高斯消元过程

for k = 1:n-1

disp(['当前第 ', num2str(k), ' 列消元:']);

for i = k+1:n

% 计算消元因子

factor = A(i, k) / A(k, k);

% 更新矩阵 A 的第 i 行

A(i, k:n) = A(i, k:n) - factor \* A(k, k:n);

% 更新向量 b 的第 i 行

```

b(i) = b(i) - factor * b(k);
end
% 显示消元后的矩阵和向量
disp(['消元后矩阵 A (第 ', num2str(k), ' 列完成):']);
disp(A);
disp('更新后的向量 b:');
disp(b);
end

% 回代求解过程
x = zeros(n, 1); % 初始化解向量
for i = n:-1:1
x(i) = (b(i) - A(i, i+1:n) * x(i+1:n)) / A(i, i);
% 显示当前回代结果
disp(['回代第 ', num2str(i), ' 行后解 x:']);
disp(x);
end

% 输出最终结果
disp('主元消去法最终解:');
disp(x);

```

## 结果:

```
>> code2_yyz
```

初始矩阵 A 和向量 b:

|        |         |         |
|--------|---------|---------|
| 1.0000 | 2.0000  | 3.0000  |
| 5.0000 | 4.0000  | 10.0000 |
| 3.0000 | -0.1000 | 1.0000  |

1  
0  
2

当前第 1 列消元:

消元后矩阵 A (第 1 列完成) :

|        |         |         |
|--------|---------|---------|
| 1.0000 | 2.0000  | 3.0000  |
| 0      | -6.0000 | -5.0000 |
| 0      | -6.1000 | -8.0000 |

更新后的向量 b:

1  
-5  
-1

当前第 2 列消元:

消元后矩阵 A (第 2 列完成) :

|        |         |         |
|--------|---------|---------|
| 1.0000 | 2.0000  | 3.0000  |
| 0      | -6.0000 | -5.0000 |
| 0      | 0       | -2.9167 |

更新后的向量 b:

1.0000  
-5.0000  
4.0833

回代第 3 行后解 x:

0  
0  
-1.4000

回代第 2 行后解 x:

0  
2.0000  
-1.4000

回代第 1 行后解 x:

1.2000  
2.0000  
-1.4000

主元消去法最终解:

1.2000  
2.0000  
-1.4000

>>

2.编程实现LU分解，求线性方程组

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

**代码:**



The screenshot shows the MATLAB IDE with a file explorer on the left containing several code files. The main editor displays the code for 'code3\_yyz.m'. The code performs LU decomposition on matrix A2 and outputs the results. The terminal window at the bottom shows the execution output, including the initial matrix A, the intermediate L and U matrices after the first step, and the final L and U matrices.

```

1 % 任务 2: LU 分解法
2 % 实验人: 杨跃浙
3 A2 = [1 2 -1; 4 1 1; 2 3 2];
4 b2 = [4; 1; -4];
5
6 n = length(b2);
7 L = eye(n); % 初始化 L 为单位矩阵
8 U = A2; % 初始化 U 为 A2
9
10 disp('初始矩阵 A:');
11 disp(A2);
12
13 % LU 分解
14 for k = 1:n-1
15     for i = k+1:n
16         L(i, k) = U(i, k) / U(k, k);
17         U(i, k:n) = U(i, k:n) - L(i, k) * U(k, k:n);
18     end
19     % 输出当前 L 和 U
20     disp(['第 ', num2str(k), ' 步 LU 分解后:']);
21     disp('矩阵 L:');
22     disp(L);
23     disp('矩阵 U:');
24     disp(U);
25 end

```

Terminal Output:

```

0
-0.0000
-3.0000
第 1 步回代后, x:
1.0000
-0.0000
-3.0000
LU 分解法最终解:
1.0000
-0.0000
-3.0000

```

% 任务 2: LU 分解法

% 实验人: 杨跃浙

A2 = [1 2 -1; 4 1 1; 2 3 2];

b2 = [4; 1; -4];

n = length(b2);

L = eye(n); % 初始化 L 为单位矩阵

U = A2; % 初始化 U 为 A2

disp('初始矩阵 A:');

disp(A2);

% LU 分解

for k = 1:n-1

for i = k+1:n

L(i, k) = U(i, k) / U(k, k);

U(i, k:n) = U(i, k:n) - L(i, k) \* U(k, k:n);

end

% 输出当前 L 和 U

disp(['第 ', num2str(k), ' 步 LU 分解后:']);

disp('矩阵 L:');

disp(L);

disp('矩阵 U:');

```

disp(U);
end

disp('LU 分解完成:');
disp('矩阵 L:');
disp(L);
disp('矩阵 U:');
disp(U);

% 解  $Ly = b2$  (前向替代)
y = zeros(n, 1);
disp('开始前向替代求解  $Ly = b$ ');
for i = 1:n
    y(i) = b2(i) - L(i, 1:i-1) * y(1:i-1);
    % 输出当前 y
    disp(['第 ', num2str(i), ' 步前向替代后, y:']);
    disp(y);
end

% 解  $Ux = y$  (回代)
x2 = zeros(n, 1);
disp('开始回代求解  $Ux = y$ ');
for i = n:-1:1
    x2(i) = (y(i) - U(i, i+1:n) * x2(i+1:n)) / U(i, i);
    % 输出当前 x2
    disp(['第 ', num2str(i), ' 步回代后, x:']);
    disp(x2);
end

disp('LU 分解法最终解:');
disp(x2);

```

## 结果:

>> code3\_yyz

初始矩阵 A:

|   |   |    |
|---|---|----|
| 1 | 2 | -1 |
| 4 | 1 | 1  |
| 2 | 3 | 2  |

第 1 步 LU 分解后:

矩阵 L:

|   |   |   |
|---|---|---|
| 1 | 0 | 0 |
| 4 | 1 | 0 |
| 2 | 0 | 1 |

矩阵 U:

|   |    |    |
|---|----|----|
| 1 | 2  | -1 |
| 0 | -7 | 5  |
| 0 | -1 | 4  |

第 2 步 LU 分解后:

矩阵 L:

|        |        |   |        |   |
|--------|--------|---|--------|---|
| 1.0000 |        | 0 |        | 0 |
| 4.0000 | 1.0000 |   |        | 0 |
| 2.0000 | 0.1429 |   | 1.0000 |   |

矩阵 U:

|        |         |         |
|--------|---------|---------|
| 1.0000 | 2.0000  | -1.0000 |
| 0      | -7.0000 | 5.0000  |
| 0      | 0       | 3.2857  |

LU 分解完成:

矩阵 L:

|        |        |   |        |   |
|--------|--------|---|--------|---|
| 1.0000 |        | 0 |        | 0 |
| 4.0000 | 1.0000 |   |        | 0 |
| 2.0000 | 0.1429 |   | 1.0000 |   |

矩阵 U:

|        |         |         |
|--------|---------|---------|
| 1.0000 | 2.0000  | -1.0000 |
| 0      | -7.0000 | 5.0000  |
| 0      | 0       | 3.2857  |

开始前向替代求解  $Ly = b$

第 1 步前向替代后, y:

|   |
|---|
| 4 |
| 0 |
| 0 |

第 2 步前向替代后, y:

|     |
|-----|
| 4   |
| -15 |
| 0   |

第 3 步前向替代后, y:

|          |
|----------|
| 4.0000   |
| -15.0000 |
| -9.8571  |

开始回代求解  $Ux = y$

第 3 步回代后, x:

|   |
|---|
| 0 |
|---|

0  
-3.0000

第 2 步回代后, x:

0  
-0.0000  
-3.0000

第 1 步回代后, x:

1.0000  
-0.0000  
-3.0000

LU 分解法最终解:

1.0000  
-0.0000  
-3.0000

>>

3. 编程实现追赶法, 求三对角方程组

$$\begin{cases} 9x_1 + 6x_2 = 1 \\ 6x_1 + 9x_2 + 9x_3 = 2 \\ 9x_2 + 7x_3 + 9x_4 = 2 \\ 10x_3 + 3x_4 = 4 \end{cases}$$

代码:

```

1 % 任务 3: 追赶法
2 % 实验人: 杨跃浙
3 a = [0; 6; 0; 10]; % 下对角线元素
4 b = [9; 9; 7; 3]; % 主对角线元素
5 c = [6; 9; 9; 0]; % 上对角线元素
6 d = [1; 2; 2; 4]; % 右端向量
7
8 n = length(b);
9 P = zeros(n, 1); % 存储修改后的对角线
10 Q = zeros(n, 1); % 存储右端向量的变换
11
12 disp('初始数据:');
13 disp('下对角线元素 a:');
14 disp(a);
15 disp('主对角线元素 b:');
16 disp(b);
17 disp('上对角线元素 c:');
18 disp(c);
19 disp('右端向量 d:');
20 disp(d);
21
22 % 前向消元
23 P(1) = b(1);
24 Q(1) = d(1);
25 for i = 2:n
26     P(i) = b(i) - c(i-1)*P(i-1)/a(i-1);
27     Q(i) = d(i) - c(i-1)*Q(i-1)/a(i-1);
28 end
29 % 回代过程
30 x(n) = Q(n)/P(n);
31 for i = n-1:-1:1
32     x(i) = (Q(i) - c(i)*x(i+1))/P(i);
33 end
34 % 追赶法最终解:
35 x

```

Q(3) = 0.043478  
 P(4) = 12.7826  
 Q(4) = 0.27891  
 回代过程:  
 x(4) = 0.27891  
 x(3) = 0.31633  
 x(2) = -0.30272  
 x(1) = 0.31293  
 追赶法最终解:  
 0.3129  
 -0.3027  
 0.3163  
 0.2789

% 任务 3: 追赶法

% 实验人: 杨跃浙

```

a = [0; 6; 9; 10]; % 下对角线元素
b = [9; 9; 7; 3]; % 主对角线元素
c = [6; 9; 9; 0]; % 上对角线元素
d = [1; 2; 2; 4]; % 右端向量

n = length(b);
P = zeros(n, 1); % 存储修改后的对角线
Q = zeros(n, 1); % 存储右端向量的变换

disp('初始数据:');
disp('下对角线元素 a:');
disp(a);
disp('主对角线元素 b:');
disp(b);
disp('上对角线元素 c:');
disp(c);
disp('右端向量 d:');
disp(d);

% 前向消元
P(1) = b(1);
Q(1) = d(1) / P(1);
disp('前向消元过程:');
disp(['P(1) = ', num2str(P(1))]);
disp(['Q(1) = ', num2str(Q(1))]);

for i = 2:n
    P(i) = b(i) - a(i) * c(i-1) / P(i-1);
    Q(i) = (d(i) - a(i) * Q(i-1)) / P(i);
    % 输出每一步计算的 P 和 Q
    disp(['P(', num2str(i), ') = ', num2str(P(i))]);
    disp(['Q(', num2str(i), ') = ', num2str(Q(i))]);
end

% 回代求解
x3 = zeros(n, 1);
x3(n) = Q(n);
disp('回代过程:');
disp(['x(', num2str(n), ') = ', num2str(x3(n))]);

```

```

for i = n-1:-1:1
x3(i) = Q(i) - c(i) * x3(i+1) / P(i);
% 输出每一步计算的 x
disp(['x(', num2str(i), ') = ', num2str(x3(i))]);
end

disp('追赶法最终解:');
disp(x3);

```

## 结果:

>> code4\_yyz

初始数据:

下对角线元素 a:

```

0
6
9
10

```

主对角线元素 b:

```

9
9
7
3

```

上对角线元素 c:

```

6
9
9
0

```

右端向量 d:

```

1
2
2
4

```

前向消元过程:

```

P(1) = 9
Q(1) = 0.11111
P(2) = 5
Q(2) = 0.26667
P(3) = -9.2
Q(3) = 0.043478
P(4) = 12.7826
Q(4) = 0.27891

```

回代过程:

```

x(4) = 0.27891
x(3) = 0.31633
x(2) = -0.30272
x(1) = 0.31293

```

追赶法最终解:

```

0.3129
-0.3027
0.3163

```

0.2789

&gt;&gt;

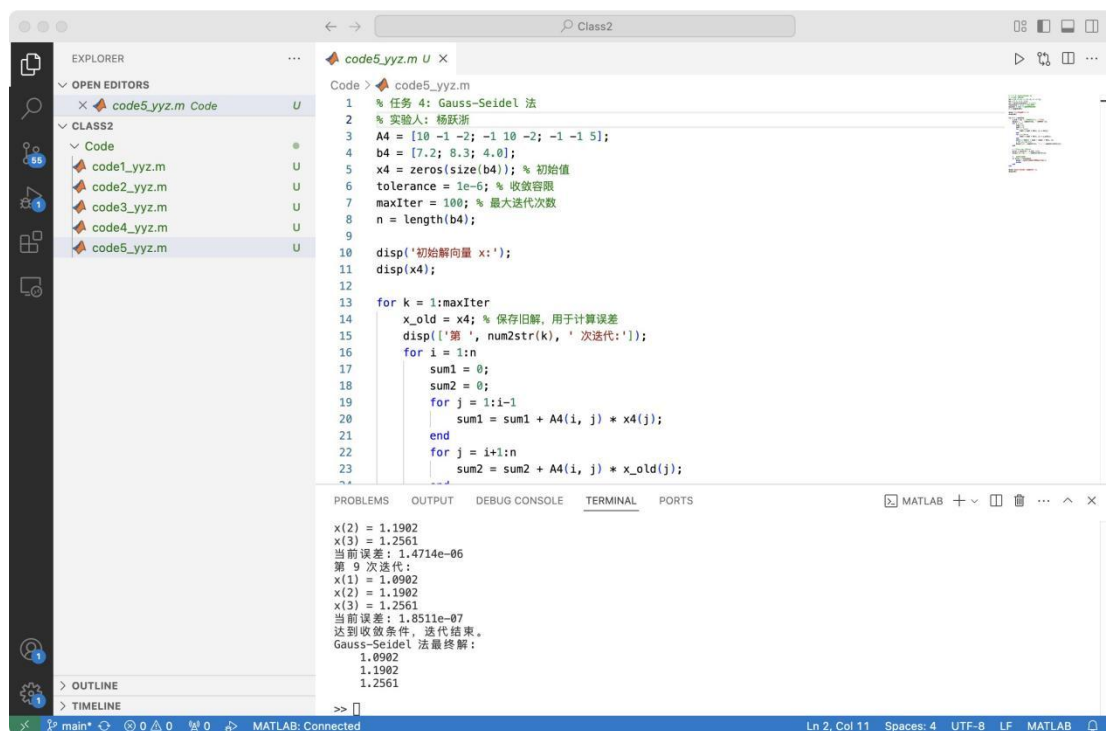
#### 4.编程实现 Gauss - Seidel 法, 求方程组

$$10x_1 - x_2 - 2x_3 = 7.2$$

$$\{-x_1 + 10x_2 - 2x_3 = 8.3$$

$$-x_1 - x_2 + 5x_3 = 42$$

代码:



```

1 % 任务 4: Gauss-Seidel 法
2 % 实验人: 杨跃浙
3 A4 = [10 -1 -2; -1 10 -2; -1 -1 5];
4 b4 = [7.2; 8.3; 4.0];
5 x4 = zeros(size(b4)); % 初始值
6 tolerance = 1e-6; % 收敛容限
7 maxIter = 100; % 最大迭代次数
8 n = length(b4);
9
10 disp('初始解向量 x:');
11 disp(x4);
12
13 for k = 1:maxIter
14     x_old = x4; % 保存旧解, 用于计算误差
15     disp(['第 ', num2str(k), ' 次迭代:']);
16     for i = 1:n
17         sum1 = 0;
18         sum2 = 0;
19         for j = 1:i-1
20             sum1 = sum1 + A4(i, j) * x4(j);
21         end
22         for j = i+1:n
23             sum2 = sum2 + A4(i, j) * x_old(j);
24         end
25         x4(i) = (b4(i) - sum1 - sum2) / A4(i, i);
26     end
27     % 计算当前误差
28     error = max(abs(x4 - x_old));
29     if error < tolerance
30         break;
31     end
32 end
33
34 x(2) = 1.1902
35 x(3) = 1.2561
36 当前误差: 1.4714e-06
37 第 9 次迭代:
38 x(1) = 1.0902
39 x(2) = 1.1902
40 x(3) = 1.2561
41 当前误差: 1.8511e-07
42 达到收敛条件, 迭代结束。
43 Gauss-Seidel 法最终解:
44     1.0902
45     1.1902
46     1.2561

```

% 任务 4: Gauss-Seidel 法

% 实验人: 杨跃浙

A4 = [10 -1 -2; -1 10 -2; -1 -1 5];

b4 = [7.2; 8.3; 4.0];

x4 = zeros(size(b4)); % 初始值

tolerance = 1e-6; % 收敛容限

maxIter = 100; % 最大迭代次数

n = length(b4);

disp('初始解向量 x:');

disp(x4);

```

for k = 1:maxIter
x_old = x4; % 保存旧解, 用于计算误差
disp(['第 ', num2str(k), ' 次迭代:']);
for i = 1:n
sum1 = 0;
sum2 = 0;
for j = 1:i-1
sum1 = sum1 + A4(i, j) * x4(j);
end
for j = i+1:n
sum2 = sum2 + A4(i, j) * x_old(j);
end
x4(i) = (b4(i) - sum1 - sum2) / A4(i, i);
% 输出当前的 x(i)
disp(['x(', num2str(i), ') = ', num2str(x4(i))]);
end
% 计算当前误差并显示
error = norm(x4 - x_old, inf);
disp(['当前误差: ', num2str(error)]);
% 检查收敛条件
if error < tolerance
disp('达到收敛条件, 迭代结束。');
break;
end
end

disp('Gauss-Seidel 法最终解:');
disp(x4);

```

## 结果:

```

>> code5_yyz
初始解向量 x:
    0
    0
    0

第 1 次迭代:
x(1) = 0.72
x(2) = 0.902
x(3) = 1.1244
当前误差: 1.1244

```



```
第 2 次迭代:
x(1) = 1.0351
x(2) = 1.1584
x(3) = 1.2387
当前误差: 0.31508
第 3 次迭代:
x(1) = 1.0836
x(2) = 1.1861
x(3) = 1.2539
当前误差: 0.048498
第 4 次迭代:
x(1) = 1.0894
x(2) = 1.1897
x(3) = 1.2558
当前误差: 0.0058191
第 5 次迭代:
x(1) = 1.0901
x(2) = 1.1902
x(3) = 1.2561
当前误差: 0.00074098
第 6 次迭代:
x(1) = 1.0902
x(2) = 1.1902
x(3) = 1.2561
当前误差: 9.2929e-05
第 7 次迭代:
x(1) = 1.0902
x(2) = 1.1902
x(3) = 1.2561
当前误差: 1.1699e-05
第 8 次迭代:
x(1) = 1.0902
x(2) = 1.1902
x(3) = 1.2561
当前误差: 1.4714e-06
第 9 次迭代:
x(1) = 1.0902
x(2) = 1.1902
x(3) = 1.2561
当前误差: 1.8511e-07
达到收敛条件, 迭代结束。
Gauss-Seidel 法最终解:
    1.0902
    1.1902
    1.2561

>>
```

## 【小结或讨论】

在本次实验中, 我通过编程实现了多种求解线性方程组的数值方法, 包括列主元消去法、全主元消去法、LU 分解法、追赶法和

Gauss-Seidel 法, 并通过 MATLAB 验证了每种方法的正确性和效率。

以下是各方法的分析与讨论:

## 1. 列主元和全主元消去法

列主元消去法和全主元消去法均基于高斯消元的思想, 通过逐列消元将系数矩阵  $A$  化为上三角矩阵, 从而利用回代过程求解方程组。两者的区别在于主元选择策略:

- 列主元消去法选择当前列中绝对值最大的元素作为主元, 主要用于提高计算的数值稳定性。
- 全主元消去法不仅在列中选择主元, 还在行中寻找最优位置交换, 进一步提高算法的稳定性, 但实现复杂度更高。

主元消去法的数学过程如下:

$$L_{ij} = \frac{A_{ij}}{A_{ii}} \quad (i > j)$$

$$A[i, j] = A[i, j] - L_{ij} \cdot A[j, i], \quad b[i] = b[i] - L_{ij} \cdot b[j].$$

结果表明, 列主元消去法和全主元消去法在正确性上均表现良好, 但在某些特殊矩阵中, 全主元法的稳定性更高。

## 2. LU 分解法

LU 分解法通过将矩阵  $A$  分解为一个下三角矩阵  $L$  和一个上三角矩阵  $U$ , 从而将原方程组的求解分为两步:

1. 求解  $Ly = b$  (前向替代)。
2. 求解  $Ux = y$  (回代)。

LU 分解的公式如下:

$$A = L \cdot U, \quad L_{ij} = \frac{A_{ij}}{A_{jj}}, \quad U_{ij} = A_{ij} - \sum_{k=1}^{j-1} L_{ik} U_{kj}.$$

LU 分解法适用于系数矩阵较大且需要多次求解的情况, 因为矩阵分解只需要计算一次, 而前向替代和回代的计算量较小。

实验结果表明, LU 分解法的结果与主元消去法一致, 并且在矩阵重复使用的情况下效率更高。

### 3. 追赶法

追赶法是一种专门用于三对角矩阵的数值方法。通过简化高斯消元的过程, 追赶法利用矩阵的稀疏性特点, 大幅降低计算复杂度

(从  $O(n^3)$  降至  $O(n)$ )。其主要过程如下:

前向消元:

$$P_i = b_i - a_i \frac{c_{i-1}}{P_{i-1}}, \quad Q_i = \frac{d_i - a_i Q_{i-1}}{P_i}$$

其中  $P_1 = b_1, Q_1 = d_1/P_1$ 。

回代求解:

$$x_n = Q_n, \quad x_i = Q_i - \frac{c_i x_{i+1}}{P_i} \quad (i = n-1, n-2, \dots, 1).$$

实验结果表明, 追赶法在解决三对角矩阵时计算量显著减少, 效率远高于一般的高斯消元法。

#### 4. Gauss-Seidel 法

Gauss-Seidel 法是一种迭代法，适用于系数矩阵条件较好（如对角占优矩阵）的情况。通过不断迭代更新解向量，直至收敛满足精度要求：

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

其中  $k$  表示迭代次数。

实验中，我通过设置误差限  $\epsilon = 10^{-6}$  控制收敛条件，迭代结果表明 Gauss-Seidel 法可以快速收敛到高精度解。但其收敛速度和稳定性较强依赖于矩阵的特性。

#### 5. 各方法的比较与总结

| 方法     | 特点                    | 计算量      | 适用场景          |
|--------|-----------------------|----------|---------------|
| 列主元消去法 | 稳定性较高，适合一般矩阵求解        | $O(n^3)$ | 一般线性方程组       |
| 全主元消去法 | 最稳定，但操作复杂，适合数值问题敏感的场景 | $O(n^3)$ | 稳定性要求较高的线性方程组 |

|                |                           |          |                    |
|----------------|---------------------------|----------|--------------------|
| LU 分解法         | 矩阵分解后多次求解效率高，适合重复求解场景     | $O(n^3)$ | 矩阵不变，右端向量变化的线性方程组  |
| 追赶法            | 高效，适合三对角矩阵                | $O(n)$   | 三对角矩阵，如有限差分法中的离散方程 |
| Gauss-Seidel 法 | 迭代法，适合对角占优矩阵或稀疏矩阵，需满足收敛条件 | $O(kn)$  | 稀疏矩阵、对角占优矩阵        |

通过本次实验，我不仅掌握了多种线性方程组求解方法，还对它们的优缺点、适用场景有了更深刻的认识。在实际应用中，应根据问题特点选择合适的求解方法，以兼顾效率与稳定性。