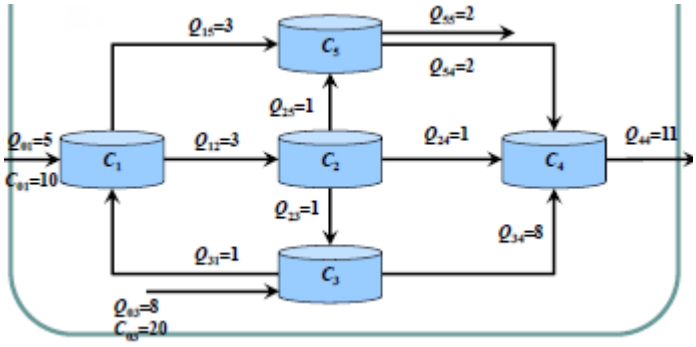
第三章

问题回顾 利用质量守恒求一个耦合反应系统中的稳态浓度。下图中 Q 为体积流率（m3/min），C 为某保守性物质（在化学反应中不增加也不减少的物质）浓度（mg/m3），各反应器内状态为均匀的。



问题分析 根据题目可知，对于任意一个反应容器i，其进出的体积流率 Q 都是相等的。为了求稳定状态下，保守性物质 C 的浓度，可以利用每个反应容器i中的保守性物料 C 的守恒关系列写方程。

$$\mathrm{m}_{出} = \mathrm{m}_{入}$$

即：

$$c_i Q_{i0} + c_i \sum_{j=1}^{5} Q_{ij} = \sum_{j=1}^{5} c_j Q_{ji} + Q_{0i} c_{0i}$$

c_i是未知量。下面对 c_1 的方程形式进行整理和用向量的形式表示上面式子。

$$[c_1 \quad c_2 \quad c_3 \quad c_4 \quad c_5] \begin{bmatrix} -Q_{10} - Q_{12} - Q_{13} - Q_{14} - Q_{15} \\ Q_{21} \\ Q_{31} \\ Q_{41} \\ Q_{51} \end{bmatrix} + Q_{0i} c_{0i} = 0$$

再化简：

$$[c_1 \quad c_2 \quad c_3 \quad c_4 \quad c_5] \left(\begin{bmatrix} 0 \\ Q_{21} \\ Q_{31} \\ Q_{41} \\ Q_{51} \end{bmatrix} + \begin{bmatrix} -Q_{1\,流出} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right) + Q_{0i} c_{0i} = 0$$

将 $\bar{c}$=$[c_1 \quad c_2 \quad c_3 \quad c_4 \quad c_5]$ 写成统一的矩阵形式即：

$$\bar{c} \left(\overline{\overline{\mathrm{Q}_{入}}} + \overline{\overline{\mathrm{Q}_{出}}} \right) + \bar{c}_0 \overline{\overline{Q_{源}}} = \bar{0}$$

其中

$$\overline{\overline{Q_{入}}} = \begin{bmatrix} 0 & Q_{12} & Q_{13} & Q_{14} & Q_{15} \\ Q_{21} & 0 & Q_{23} & Q_{24} & Q_{25} \\ Q_{31} & Q_{32} & 0 & Q_{34} & Q_{35} \\ Q_{41} & Q_{42} & Q_{43} & 0 & Q_{45} \\ Q_{51} & Q_{52} & Q_{53} & Q_{54} & 0 \end{bmatrix}; \quad \overline{\overline{Q_{出}}} = diag \left(\begin{bmatrix} -Q_{1\,流出} \\ -Q_{2\,流出} \\ -Q_{3\,流出} \\ -Q_{4\,流出} \\ -Q_{5\,流出} \end{bmatrix} \right)$$

只要接触上述关于 c 的方程就可以得到稳定分布的解。下面约定，在解上述方程的时候，将方程化为类似于书本中的标准形式：

$$\begin{bmatrix} -6 & 0 & 1 & 0 & 0 \\ 3 & -3 & 0 & 0 & 0 \\ 0 & 1 & -9 & 0 & 0 \\ 0 & 1 & 8 & -11 & 2 \\ 3 & 1 & 0 & 0 & -4 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix} = \begin{bmatrix} -50 \\ 0 \\ -160 \\ 0 \\ 0 \end{bmatrix}$$

并且为了保持符号的一致性，下面讨论算法时，运用以下经典符号形式：

$$Ax = b$$

目录

交换列主元的高斯消去法

由于本问题的特点，对角线上的元素（代表流出量）肯定是列上的最大的元素，所以必定不需要进行主元调换。但是为了不失一般性，后面的讨论以及程序都带上了交换列主元的功能。

```
% this program is developed to demonstrate the Gauss method on solving
% the linear equations
n=length(A);
AB = [A b];
% modification about exchanging the pivot element
for i=1:n-1
    [M,index] = max(abs(AB(i:n,i)));
    index=index+i-1;       % get the pivotal element
    if AB(index,i) == 0
        error('singular matrix!!!')
    end
    if index ~= i
        temp = AB(index,:);
        AB(index,:) = AB(i,:);
        AB(i,:) = temp;
    end
% Down subtraction
    for j=i+1:n
        if AB(j,i)~=0
% Gauss elimination
            AB(j,i:n+1) = AB(j,i:n+1) - AB(j,i)/AB(i,i) *AB(i,i:n+1);
        end
    end
end
% Back up
x(n,1)=AB(n,n+1)/AB(n,n);
i=n-1;
while(i>=1)
    x(i)=(AB(i,n+1) - AB(i,i+1:n)*x(i+1:n))/AB(i,i);
    i = i-1;
end
```

得到最后的结果是：$\bar{x} = [11.5094 \quad 11.5094 \quad 19.0566 \quad 16.9983 \quad 11.5094]^T$

以上经典的高斯方法没有采取行缩放技术。这就导致了一种潜在的风险，对于某些条件数极大的矩阵 A 的求解会出现舍入误差不可忽略的现象。如下面这个课本中的算例

$$\begin{bmatrix} 2 & 100000 \\ 1 & 1 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 100000 \\ 2 \end{bmatrix}$$

其中矩阵 A 的条件数为：

$$\mathrm{cond}\left(\begin{bmatrix}2 & 100000\\ 1 & 1\end{bmatrix}\right) = 100000$$

但是该算例在本程序来说运行结果($x\approx [1,1]^{\mathrm{T}}$)看来完全正确，究其原因，是由于 matlab 中采用的是 double 型的双精度变量，当继续增大上述 A 矩阵的条件数时，会出现问题。如当：

$$\mathrm{A} = \begin{bmatrix}2 & 10^{17}\\ 1 & 1\end{bmatrix}\ \mathrm{b} = [\,10^{17}, 2]^{\mathrm{T}}$$

求解出现问题，得到的答案变为：

$$\mathrm{x} = [0\,, 1]^{\mathrm{T}}$$

意即 double 精度已经不够。此时就要求我们增大数据类型的精度，或者在程序中加入缩放环节，使得行中最大的元素为 1；但是考虑到实际工程中，很少有比10^{17}还要大的物理量，权衡代码执行效率和程序的通用性，本例高斯消去程序中没有采取以上措施。面对实际问题时，我们可以根据实际问题物理量的范围采用合适精度的高斯消去方法，提高解得精确度。

LU 分解法

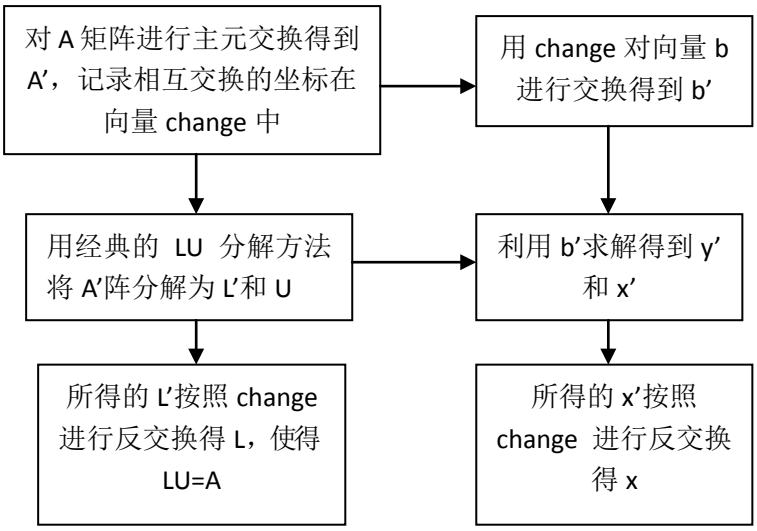
这里的LU分解是高斯消去方法的一种拓展。A=L*U，L一般为下三角阵，对角线是 1，U一般为上三角阵。而原先的方程变为一个方程组Ay=b，Ux=y。通过间接求解y得到x。在求解过程中发现，并不是所有的方程都能满足L为下三角阵，或者U为上三角阵。这并不难理解，Doolittle分解 [1] 中，下三角阵由高斯消去法中的消去系数 a_{ji}/a_{ii} 组成，如果a_{ii}正好为 0，那么就必须进行主元交换了。

比如随机取一个对角线上全为 0 的矩阵：

$$\mathrm{A} = \begin{bmatrix}0 & 0.67874\\ 0.93399 & 0\end{bmatrix}$$

就无法进行传统意义上的 LU 分解。一种可行的办法是先按照前面高斯消去中的交换列主元的策略使得，可以用传统的方法 LU 分解，然后根据前面的交换记录，将 L 重新交换回原来的形式。这种方法能够保证 LU=A 依然成立。同时，为了求解 Ax=b，应该同时按照主元交换记录交换得到 b’，求解方程 L’Ux’=b’，得到 x’后，按照主元交换记录重新得到 x。框图如下所示。

[1] 见课本



图表 1 LU 分解流程

LU 分解的 MATLAB 脚本程序如下

```
% this program is developed to demonstrate the LU factorizing method on
% solving the linear equations

n=length(A);
change=1:n;
% modification about exchanging the pivot element
for i=1:n-1
    [M,index] = max(abs(A(i:n,i)));
    index=index+i-1;       % get the pivotal element
    if A(index,i) == 0
        error('singular matrix!!!')
    end
   %once exchange happens record their index, and change back on matrix
L
   %or U;
    if index ~= i
        % record the index
        t = change(i);
        change(i) = change(index);
        change(index) = t;
        % change the rows
        temp = A(index,:);
        A(index,:) = A(i,:);
        A(i,:) = temp;
    end
```

```
    % get L U in Doolittle principle
    for j=i+1:n
        if A(j,i)~=0
            A(j,i+1:n) = A(j,i+1:n) - A(j,i)/A(i,i) *A(i,i+1:n);
            A(j,i) = A(j,i)/A(i,i);
        end
    end
end
L = tril(A,-1)+eye(n);
L(1:n,:) = L(change,:);
  % change back the arrows in switching the pivotal element part
U = triu(A);
y = zeros(n,1);
 % get d for L*y=b, U*x=y;
y(1)=b(1);
i=2;
b=b(change); % change b for solving the equation
while(i<=n)
    y(i)=(b(i) - A(i,1:i-1)*y(1:i-1));
    i = i+1;
end
x = zeros(n,1);
x(n) =y(n)/A(n,n);
i=n-1;
while(i>=1)
    x(i)=(y(i) - A(i,i+1:n)*x(i+1:n))/A(i,i);
    i = i-1;
end
change(change)=1:n;  % change back
x=x(change);
```

对于本题的算例，分解的结果如下：

$$L=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -0.5 & 1 & 0 & 0 & 0 \\ 0 & -0.33333 & 1 & 0 & 0 \\ 0 & -0.33333 & -0.92453 & 1 & 0 \\ -0.5 & -0.33333 & -0.075472 & 0 & 1 \end{bmatrix};\ U=\begin{bmatrix} -6 & 0 & 1 & 0 & 0 \\ 0 & -3 & 0.5 & 0 & 0 \\ 0 & 0 & -8.8333 & 0 & 0 \\ 0 & 0 & 0 & -11 & 2 \\ 0 & 0 & 0 & 0 & -4 \end{bmatrix}$$

中间计算过程中，

$$y=[-50 \quad -25 \quad -168.3333 \quad -163.9623 \quad -46.0377]^{T}$$

$$x=[11.5094 \quad 11.5094 \quad 19.0566 \quad 16.9983 \quad 11.5094]^{T}$$

特别低，我们再看看对于开头提出的对角为 0 的矩阵（新例子）

$$A=\begin{bmatrix} 0 & 0.046171 & 0.69483 \\ 0.031833 & 0 & 0.3171 \\ 0.27692 & 0.82346 & 0 \end{bmatrix}$$

分解的结果是：

$$\mathrm{L}=\begin{bmatrix}0 & -0.48777 & 1\\ 0.11495 & 1 & 0\\ 1 & 0 & 0\end{bmatrix};\quad \mathrm{U}=\begin{bmatrix}0.27692 & 0.82346 & 0\\ 0 & -0.094658 & 0.3171\\ 0 & 0 & 0.8495\end{bmatrix}$$

可以看到 L 不再是下三角阵了，但是依然有下面式子成立。

$$\begin{bmatrix}0 & -0.48777 & 1\\ 0.11495 & 1 & 0\\ 1 & 0 & 0\end{bmatrix}\begin{bmatrix}0.27692 & 0.82346 & 0\\ 0 & -0.094658 & 0.3171\\ 0 & 0 & 0.8495\end{bmatrix}=\mathrm{A}$$

LU 分解相对于高斯消去的优点在本例中就是，改变反应系统的输入（C 的输入量），可以不用重新分解 A 进行计算，复杂性为 O(n^2)。但是如果要调节管路之间的流量Q_{ij}，就改变了 A 矩阵，那么要重新进行 LU 分解。

迭代方法

迭代方法就是利用原来的线性方程 Ax=b 构造一个迭代的线性递推式$x^{k+1}=G\,x^{k}+f$ 。一般的构造方法是将 A 裂解为 M-N 两个矩阵，然后可得$G=M^{-1}N;f=M^{-1}b$。

编程过程中，一般可以观察迭代解是否收敛到可以忽略的误差范围内，作为迭代的停止条件。

下面按照经典的几种迭代方法进行分类讨论。需要说明的是迭代法同样存在 LU 分解中类似的主对角线上有 0 元素的操作问题，求解过程中和 LU 分解类似，只需要进行主元交换就可以解决，所以后面的讨论默认 A 的对角线不为 0。

Jacobi 方法

Jacobi 方法采用 M=A 的对角线上元素组成的向量，在 MATLAB 的矩阵语言中就是：M=diag(diag(A))，一般该矩阵记做是 D。由于 Jacobi 方法很容易以用非矩阵语言实现，也就是说，从方程 $Dx^{k+1}=N\,x^{k}+b$ 中能够容易解出 x^{k+1}，因此文中的 jacobi 将按照 MATLAB 矩阵化的 m 语言给出。

```
% this program is developed to realize the Jacobi method
D = diag(diag(A));
N = D-A;
n = length(A);

G = D^-1*N;
f = D^-1*b;
i_max = 1e3; % max iteration times
x0 = zeros(n,1);
dtol=1e-5;
for i=1:i_max
    x = G*x0 + f;
    if norm(x-x0)/norm(x) < dtol
```

```
        break;
    else
        x0 = x;
    end
end
```

其中的 tril()和 triu()函数时求矩阵的下、上三角矩阵。本例的求解通过 11 次迭代得到了预期的 1e-5 的相对精度。

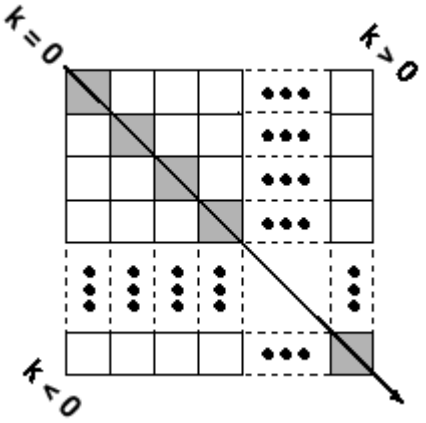
$$x = [11.5094 \quad 11.5094 \quad 19.0566 \quad 16.9983 \quad 11.5094]^T$$

G-S 方法

高斯赛达尔方法的迭代方程构造形式为：M=D-L，N=U，其中L定义为A的k=-1对角线（如下图）以下元素相反数构成的下三角阵，是与A同维方阵。

在MATLAB中用`L=-tril(A,-1)` 表示。同理，`U = -triu(A,1)`。

$$G = (D-L)^{-1}U;\ f = (D-L)^{-1}b$$



由于 G-S 方法是一种特殊的超松弛方法（w=1），下面仅用矩阵的方式在 MATLAB 脚本中实现，非矩阵方式在后面 SOR 方法中说明：

```
% this program is developed to realize the G-S method, on
% iteratively solving the linear equations.
D = diag(diag(A));
L = -tril(A,-1);
U = -triu(A,1);
n = length(A);

G=(D-L)^-1*U;
f=(D-L)^-1*b;
i_max = 1e3; % max iteration times
x0 = zeros(n,1);
dtol=1e-5;
for i=1:i_max
    x = G*x0 + f;
    if norm(x-x0)/norm(x) < dtol
        break;
    else
        x0 = x;
```

```
    end
end
```

经过 5 次迭代得到收敛解（估计相对精度小于 1e-5）

$$x = [11.5094 \quad 11.5094 \quad 19.0566 \quad 16.9983 \quad 11.5094]^T$$

该方法的收敛性取决于迭代矩阵 G 的性质。对于误差向量 $r=x^{k+1} - x^*$ 有如下方程

$$r^{k+1} = G * r^k$$

两边取范数有

$$\frac{\|r^{k+1}\|}{\|r^{k+1}\|} = \|G\| = q$$

显然 q 越接近于 0，该方法收敛越快。事实上可以用 G 的最大特征值（$\lambda_{max}(G)$）来衡量该矩阵的收敛速度，有关收敛速度还会在后面讨论。

超松弛 SOR 法

如前所述 G-S 方法以及 Jacobi 方法都存在收敛性的问题，由于 G 矩阵在方程确定后就已经固定不能改变，如何使那些原本用 G-S 方法不收敛的问题变为收敛，或者如何加快 G-S 方法的收敛性便成为一个有意义的问题，这即是 SOR 方法的初衷，通过一个可调控的参量 w 控制收敛性。

用矩阵形式容易分析收敛性，按照递推公式的形式可以用非 MATLAB 这样矩阵化的语言编码实现，并且编码简单。

下面分别用两种方式编码实现：

1、 非矩阵形式

```
% this program is developed to realize the modified G-S method, SOR, on
% interatively solving the linear equations.
D = diag(diag(A));
L = -tril(A,-1);
U = -triu(A,1);
n = length(A);
w=1.5;
G=(D-L)^-1*U;
f=(D-L)^-1*b;
i_max = 1e3; % max iteration times
x0 = zeros(n,1);
dtol=1e-5;
x = x0;
for i=1:i_max
    for j=1:n
        x(j) = x0(j) + w/A(j,j)*(b(j) - A(j,1:j-1)*x(1:j-1)...
            -A(j,j:n)*x0(j:n));
    end
    if norm(x-x0)/norm(x) < dtol
        break;
```

```
    else
        x0 = x;
    end
end
```

2、矩阵形式（略去与矩阵形式相同的初始化部分）

```
% SOR- version 2, matrix style.
w = 1.00277;        % sor factor
G = inv((D- w*L)) *((1-w)*D + w* U );
f = w*b;
for i = 1:i_max
    x = G*x0 + f;
    if norm(x-x0)/norm(x0) < dtol
       break;
    else
        x0 = x;
    end
end
```

收敛速度

如前面所述，迭代方法的收敛速度可以用谱半径或者迭代矩阵的最大特征根来衡量，针对本文题的算例进行计算可以得到以下表格。经过调试，发现当 w=1.00277 谱半径最小，但事实上此时的 SOR 方法已经十分接近于 G-S 方法。

$\frac{\lvert x-x_0\rvert}{\lvert x_0\rvert}$ ≤1e-5	G-S	SOR, w=1.00277	Jacobi
$\lambda_{max}(G)$	0.0185	0.0055	0.02646
迭代次数	5	5	11

虽然本例子中 G-S 方法和 SOR 方法收敛速度极快。但这并不对所有矩阵都奏效的。下面进行了一个数值小实验试验，可以窥视这一现象。

用 MATLAB 中 rand(n)函数随机生成了一个 4*4 的矩阵，每个元素服从[0,1]上的均匀分布，所以每个元素的大小几乎相等，如下所示。

$$A = \begin{bmatrix} 0.58906 & 0.99579 & 0.77784 & 0.92182 \\ 0.43626 & 0.36496 & 0.44018 & 0.22332 \\ 0.54974 & 0.96617 & 0.6305 & 0.37449 \\ 0.35297 & 0.024741 & 0.31402 & 0.30814 \end{bmatrix}$$

对 A 改变其对角线上元素的大小（diag()为 MATLAB 中取对角线的函数）

$$A' = A + \mathrm{diag}(\mathrm{diag}(A) * k);\ \ k > 0$$

发现：

k	0.1	0.2	0.3	0.4	0.8	0.9	…
SOR 方法收敛 w 的大概取值区间	空	空	空	空	空	[0.3,1.9]	[0.1,1.9]
A’的条件数	1.36e+2	4.51e+1	2.81e+1	2.32e+1	6.89e+1	8.06e+1	逐渐降低，有波动

重复上述随机矩阵的取法，都有类似的结果。可以看出条件数并不是决定解在 SOR 中的收敛性的因素，与之相比更加重要的是对角占优性。此处对角线上的元素为同行中元素的 2 倍左右时（并不是严格对角占优），SOR 或者其他两种迭代方法就有很好的表现。

事实上在本例中对角线上的元素为流出总流量的相反数，由每一个罐子的流量守恒，可以得到求解矩阵的严格对角占优的性质，这就保证了本问题（无论参数如何变化）在求解过程中的快速收敛的性质，回顾工程上的例子，曾经学过的电路方程就有类似的性质，如果用节点电压法，或者网孔电流法列写方程，也有类似的严格对角占优性质。我想，这也是迭代方法在实际应用中颇为广泛的原因。

小结

本文先对一个实际稳态分布问题列写守恒方程，从直接法和迭代法两类方法出发，考虑了实际问题的求解。并对不同方法的使用性以及性能加以了一定的比较，得出了一些有意义的结论。特别还讨论了 SOR 法的使用特点，对今后的学习有指导意义。