Visual Basic

CHAPTER 4

聯立線性方程式與矩陣

(Linear Equations & Matrices)

張 榮 興 博士
豐映科技股份有限公司
E-mail: chang.ronhsin@msa.hinet.net
http://www.resi.com.tw/vb.htm

工程師經常需要面對問題
利用有效率的計算機程式解決大量的計量

質量平衡計算

圖 4.1 是一個化學工廠的系統簡化流程圖。原料流束 1 (或進料) 中，含有 A 成分 400 kg/hr，B 成分 400 kg/hr，及 C 成分 200 kg/hr。這個工廠的主要功能是將部分的 A 及部分的 B 反應，轉化成具有商業價值的產品 C，然後將各成分分離，使產品中 C 成分的含量達到所要求的濃度。

由於三種成分中化合物 C 的沸點最低，因此，當 A、B 及 C 的混合物沸騰時，蒸氣相中會含有較高濃度的 C。相似地，容器中同時含有液態及氣態的混合物，待冷卻後，液體中 A 及 B 的含量將較蒸氣中爲高。根據此一概念，擬定出如圖 4.1 所述之流程，請作初步的質量平衡計算，求出 $S_1, S_2, \cdots\cdots, S_9$ 各流束中，A，B，C 之含量。基本數據如下：

1. 反應器中 B→C 的轉化率爲 0.5，A→C 的轉化率爲 0.3。
2. 流束 S_3 於高溫高壓條件下進入閃蒸塔中，經減壓分離成流束 S_5 及 S_4。在流束 S_5 及 S_4 中，A、B 及 C 之比率分別爲

$$S_{5A}/S_{4A}=1/10 \qquad S_{5B}/S_{4B}=1/6 \qquad S_{5C}/S_{4C}=6/5$$

3. 冷卻分離槽將 S_4 冷卻後，將部分含高濃度 A 及 B 的液體送回反應器，其中 S_6 及 S_8 中 A，B 及 C 之比率分別爲 1.5、2 及 4。
4. 蒸餾塔中，S_7 及 S_9 之分離比率分別爲 1/6、1/4 及 9。

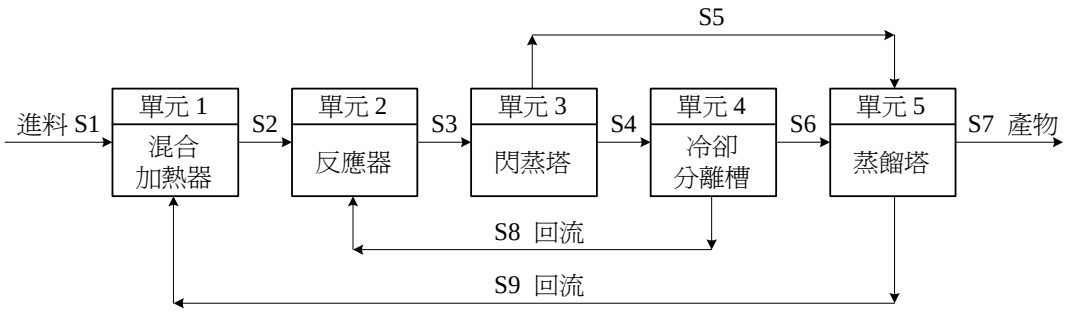


圖 4.1　簡單的化學工廠流程圖

設計問題 D-IV 中，總共有 9 個流體流束，每一流束中都含有 A、B 及 C 三種化合物。除了流束 S_1 中 A、B 及 C 的流量爲已知以外，我們總共有 24 個未知數需要求解，因此，可預測總共需要建立 24 個獨立的方程式。這只是一個相當簡化的工程問題，就需建立如此多的方程式，可推想一個實際化學工廠或其他生產工廠的質量平衡計算將是如何繁重的工作。因此，借助於有效率的計算機程式解決大量的質量及能量平衡計算，乃成爲工程師經常需要面對的問題。

第一節　聯立線性方程式

Visual Basic

化工程序的質量平衡計算，常需求解大量的聯立代數方程式。以設計問題 D-IV 爲例，我們可就各操作單元分別建立方程式：

單元 1 **混合加熱器**　用於將混合物加熱至反應器所要求的入口溫度。在此單元中進料 S_1 與回流 S_9 混合及加熱後，變成流束 S_2，故各成分之質量平衡方程式爲

$$S_{2A} = S_{9A} + 400 \qquad \textbf{(4-1.1)}$$

$$S_{2B} = S_{9B} + 400 \qquad \textbf{(4-1.2)}$$

$$S_{2C} = S_{9C} + 200 \qquad \textbf{(4-1.3)}$$

單元 2 **反應器**　在反應器中，進行反應 A→C 及 B→C，其中 A 的轉變率爲 0.30，B 的轉化率爲 0.50，反應器之進料爲 S_2 及 S_8，因此，

$$S_{3A} = (1-0.3)\cdot(S_{2A} + S_{8A}) \qquad \textbf{(4-1.4)}$$

$$S_{3B} = (1-0.5)\cdot(S_{2B} + S_{8B}) \qquad \textbf{(4-1.5)}$$

$$S_{3C} = \frac{1}{2}\cdot(S_{2B} + S_{8B}) + 0.3(S_{2A} + S_{8A}) + S_{2C} + S_{8C} \qquad \textbf{(4-1.6)}$$

單元 3 **閃蒸塔**　提供熱量，使進入本單元的部分混合物蒸發，由於化合物 C 之沸點較 A 及 B 低，故蒸氣中 C 的含量即增高。根據已知條件

$$S_{5A} = \frac{1}{10}\cdot S_{4A} \qquad \textbf{(4-1.7)}$$

$$S_{5B} = \frac{1}{6} \cdot S_{4B} \quad \textbf{(4-1.8)}$$

$$S_{5C} = \frac{6}{5} \cdot S_{4C} \quad \textbf{(4-1.9)}$$

又由於 $S_3 = S_4 + S_5$，所以可以得到下列平衡式

$$S_{3A} = S_{4A} + S_{5A} \quad \textbf{(4-1.10)}$$

$$S_{3B} = S_{4B} + S_{5B} \quad \textbf{(4-1.11)}$$

$$S_{3C} = S_{4C} + S_{5C} \quad \textbf{(4-1.12)}$$

單元 4 **冷卻分離器**　將進料冷卻，使液相含有較高濃度的 A 及 B，並部分回流至單元 2。由已知條件：

$$S_{6A} = \frac{3}{2} \cdot S_{8A} \quad \textbf{(4-1.13)}$$

$$S_{6B} = 2 \cdot S_{8B} \quad \textbf{(4-1.14)}$$

$$S_{6C} = 4 \cdot S_{8C} \quad \textbf{(4-1.15)}$$

又由於 $S_4 = S_6 + S_8$，因此，可以得到

$$S_{4A} = S_{6A} + S_{8A} \quad \textbf{(4-1.16)}$$

$$S_{4B} = S_{6B} + S_{8B} \quad \textbf{(4-1.17)}$$

$$S_{4C} = S_{6C} + S_{8C} \quad \textbf{(4-1.18)}$$

單元 5 **蒸餾塔**　用於提高產品中所含 C 的濃度，根據已知數據：

$$S_{7A} = \frac{1}{6} \cdot S_{9A} \quad \textbf{(4-1.19)}$$

$$S_{7B} = \frac{1}{4} \cdot S_{9B} \quad \textbf{(4-1.20)}$$

$$S_{7C} = 9 \cdot S_{9C} \quad \textbf{(4-1.21)}$$

又因爲 $S_6 + S_5 = S_7 + S_9$，所以可將各成分之平衡式寫成

$$S_{6A} + S_{5A} = S_{7A} + S_{9A} \quad \textbf{(4-1.22)}$$

$$S_{6B} + S_{5B} = S_{7B} + S_{9B} \quad \textbf{(4-1.23)}$$

$$S_{6C} + S_{5C} = S_{7C} + S_{9C} \quad \textbf{(4-1.24)}$$

我們總共得到 24 個變數及 24 個方程式。在建立以上方程組的時候，我們假設此系統已經是經過長時間的操作，並且已經達到“穩定狀態”(Steady State)，因此，各成分之流量不會隨時間而變。

在以上 24 個方程式中，我們可發覺每一項中只含一個未知數，這類方程式我們即稱爲*線性聯立方程式*。 若將以上 24 組方程式寫成矩陣形式，則可以得到

$$\underline{\underline{A}}\,\underline{X} = \underline{B} \quad \textbf{(4-1.25)}$$

其中 $\underline{\underline{A}}_{24X24}$ 爲係數矩陣， $\underline{X} = [S_{2A}, S_{2B}, S_{2C}, S_{3A}, S_{3B}, S_{3C}, \cdots\cdots, S_{9A}, S_{9B}, S_{9C}]^T$ ， $\underline{B}$ 爲常數向量。

在本章中，我們將以實際問題爲例，說明聯立線性方程式 $\underline{\underline{A}}\,\underline{X} = \underline{B}$ 的各種求解方法。我們將依次介紹最常使用的三種線性聯立方程式解法，『高斯消去法 (Gauss Elimination)』、『高斯佐丹消去法 (Gauss-Jordan Elimination)』，及『高斯賽德迭代法 (Gauss-Seidel Iterative Method)』，最後，再介紹三對角線矩陣方程式 (Tri-diagonal Equations) 的解法。

第二節　高斯消去法

Visual Basic

由於任何方程式乘上常數均不會改變其解答，而且，任何方程式均可利用兩方程式的和或差來取代，因此，我們可利用這種方法找出方程式的解；而這種方程組的求解方法，若利用矩陣的乘法及加法來進行，就稱爲高斯消去法。

高斯消去法的基本運算原則，是利用任何方程式乘上常數都不會改變其解答的原理，將原方程組乘以適當的常數後，作相互加減處理，使係數矩陣的對角線全部變成 1，且使其左下角元素全變成 0，再由最後一個方程式由下往上代入，即可求出方程組的解。

基本上高斯消去法是一種相當規則化的求解線性聯立方程式的方法。由於具有規則性，因此，非常適合寫成計算機程式。但由於計算過程略爲繁複，因此，比較不適合用於手算求解。此外，由於演算過程中需作大量的四則運算 $(+,-,\times,\div)$，因此，方程式數量龐大時，其準確性相對堪虞。高斯消去法的計算時間約與方程數目的三次方

成比例，即解 6 個方程式所需時間約爲 3 個方程式的 8 倍。

以下用簡單的方程組說明高斯消去法的基本運算方式。首先考慮以下的線性聯立方程組：

$$\begin{cases} -3x - y + 11z = 0 \\ 13x - 8y - 3z = 20 \\ -8x + 10y - z = -5 \end{cases} \quad \textbf{(4-2.1)}$$

若將聯立方程組寫成矩陣形式，則係數矩陣及常數向量分別爲

$$\left[\begin{array}{ccc|c} -3 & -1 & 11 & 0 \\ 13 & -8 & -3 & 20 \\ -8 & 10 & -1 & -5 \end{array}\right] \quad \textbf{(4-2.2)}$$

將第一列除以 –3 後寫回第一列，第二列除以 13 再減去第一列，第三列除以 –8 再減去第一列，則可以得到

$$\left[\begin{array}{ccc|c} 1 & \frac{1}{3} & -\frac{11}{3} & 0 \\ 0 & -\frac{37}{39} & \frac{134}{39} & \frac{20}{13} \\ 0 & -\frac{19}{12} & \frac{91}{24} & \frac{5}{8} \end{array}\right] \quad \textbf{(4-2.3)}$$

上式中，第二及第三列的第一個元素均已變成 0。重複這種步驟，將第二列乘以 $-39/37$，使對角線元素變成 1。第三列乘以 $-12/19$ 再減去所得的第二列結果，就可以得到新的矩陣方程式如下所示：

$$\left[\begin{array}{ccc|c} 1 & \frac{1}{3} & -\frac{11}{3} & 0 \\ 0 & 1 & -\frac{134}{37} & -\frac{60}{37} \\ 0 & 0 & 1 & 1 \end{array}\right] \quad \textbf{(4-2.4)}$$

亦即經過高斯消去處理以後，對應的三方程式爲

$$\begin{aligned} x + \frac{1}{3}y - \frac{11}{3}z &= 0 \\ y - \frac{134}{37}z &= -\frac{60}{37} \\ z &= 1 \end{aligned} \quad \textbf{(4-2.5)}$$

由第三方程式得 $z = 1$，代回第二方程式得 $y = 2$，再代回第一方程式，得 $x = 3$。以上的運算過程即稱爲**高斯消去法**。

對計算機運算而言，我們必須設法使捨入誤差減小，以提高運算之準確度。在高斯消去法中，我們可將每一行的最大元素作爲運算樞紐，例如將方程式改寫成

$$\left[\begin{array}{ccc|c} 13 & -8 & 3 & 20 \\ -8 & 10 & -1 & -5 \\ -3 & -1 & 11 & 0 \end{array}\right]$$

即第一次消去後，得到

$$\left[\begin{array}{ccc|c} 1 & -\frac{8}{13} & \frac{3}{13} & \frac{20}{13} \\ 0 & -\frac{33}{52} & -\frac{11}{104} & -\frac{95}{104} \\ 0 & \frac{37}{39} & -\frac{152}{39} & -\frac{20}{13} \end{array}\right]$$

由於 $\left|\frac{37}{39}\right| > \left|-\frac{33}{52}\right|$，故第二次消去前先將第二列及第三列交換，再進行消去操作。利用這種方式可提高準確度 (詳見[1])，此外，亦可防止以“0”作爲樞紐而產生執行錯誤。

例題 4-1　高斯消去法

試利用高斯消去法設計一程式，並求解方程組 (4-2.1)。

解：

TOP-DOWN 設計

高斯消去法目的在於求解方程式 (4-1.25) 所示之聯立線性方程組 $\underline{\underline{A}}\,\underline{X} = \underline{B}$，我們將矩陣 $\underline{\underline{A}}$ 放在 $A(I, J)$ 中，常數向量放在 $Z(I)$ 中。所得到的解 $\underline{X}$ 則爲 $COEF(I)$。

程式設計時，首先要將所要求解的方程式係數輸入，然後，執行高斯消去法，求得方程式的解。高斯消去法程式設計基本步驟爲：

1. 找尋最大元素 BIG；
2. 以最大元素 BIG 爲樞紐進行消去運算：
3. 重複步驟 (1) 及 (2) 直到對角線左下角均爲 0；
4. 由下往上代入求得解答。

符號說明：

A： 係數矩陣，A (N , N)，即 $\underline{\underline{A}}$

COEF： 解答向量，即 $\underline{X}$

NEQN%：方程式數目

Z： 常數向量，即 $\underline{B}$

HOLD： 暫存值

TEMP： 暫存值

BIG： 暫存 A (I , J) 最大值，即樞紐元素

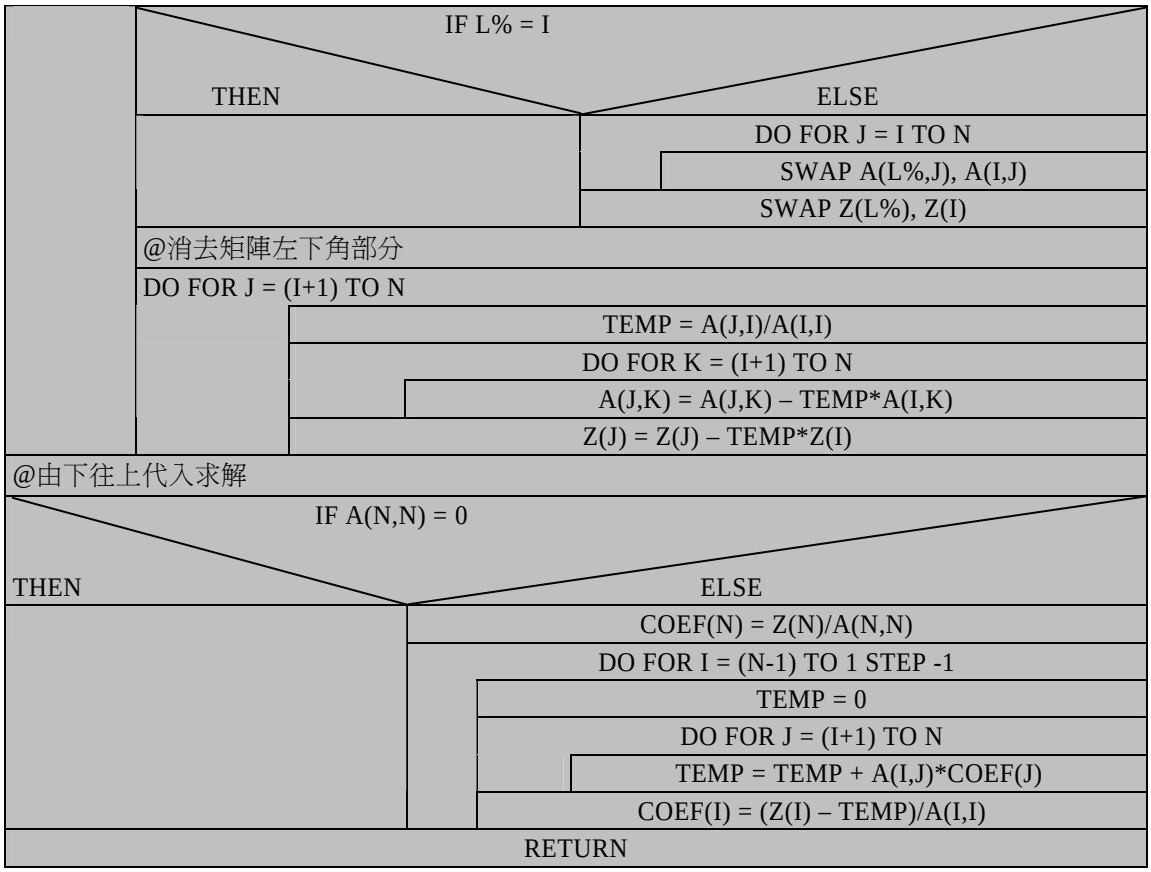
FLAG： 錯誤旗號，0 爲有解，1 表無解

主 程 式

	輸入 A (I，J) 及 Z (I)
	列印 A (I，J) 及 Z (I)
	CALL GAUSS (副程式 GAUSS)
	列印結果 = COEF (I)
執行下一作業	

副程式 **GAUSS**

@清除錯誤旗號		
FLAG = 0		
@高斯消去法		
DO FOR I = 1 TO N-1		
	@找出 BIG 及 BIG 之位置 L%	
	BIG = ｜A(I,I)｜	
	L% = 1	
	DO FOR J = (I+1) TO N	
	IF ｜A(J,I)｜< BIG	
	THEN	ELSE
		BIG = ｜A(J,I)｜
		L% = J
	@以 BIG 爲樞紐，將 BIG 放在 A(I,I)位置	
	IF BIG = 0	
	THEN	ELSE
	FLAG = 1	
	RETUEN	



程式列印：

```
Sub GaussianElimination(Xpos, Ypos)
'       ***********************************
'       * GAUSSIAN ELIMINATION
'       ***********************************
'
'    COEF = SOLUTION VECTOR
'    FLAG = ERROR FLAG
'    NEQN% = NO. OF EQUNS.
'    A       = COEFF. MATRIX
'    Z       = CONSTANT VECTOR
'
Dim A(100, 100), Z(100), Coef(100)
Do
      Cls
      Print "SOLUTION OF SIMULTANEOUS LINEAR EQUATION"
      Print " BY GAUSSIAN ELIMINATION WITH PIVOTING"
      Print
```

```
        Print "No. of Equations N = ";
        NEQN% = Val(InputBox("NO. OF EQUATIONS = ", "", , Xpos, Ypos))
        Print NEQN%
        Print
        MsgBox ("Ready to Enter Data Sets")
        Call EnterEquations(NEQN%, A, Z, Xpos, Ypos)
        Call Gauss(NEQN%, A, Z, Coef, Flag)
        Call EliminationResults(NEQN%, Coef, Flag)
        NewProjYN$ = InputBox("RUN Next Problem <Y/N>?", "", "N", Xpos, Ypos)
Loop While NewProjYN$ <> "N" And NewProjYN$ <> "n"
End Sub


Sub EnterEquations(N, A, Z, Xpos, Ypos)
For I = 1 To N
        Do
                Cls
                Print
                Print "EQUATION #"; I; " :"
                Print
                For J = 1 To N
                        Print "COEFF#"; J; " = ";
                        A(I, J) = Val(InputBox("Coefficient of Equation", "A(I,J)", , Xpos, Ypos))
                        Print A(I, J)
                Next J
                Print "Z("; I; ") =";
                Z(I) = Val(InputBox("Enter Z(I) =", "Z(I)", , Xpos, Ypos))
                Print Z(I)
                CorrectYN$ = InputBox("Coefficients Correct <Y/N>?", "", "Y", Xpos, Ypos)
        Loop While CorrectYN$ = "N" Or CorrectYN$ = "n"
Next I
End Sub


Sub EliminationResults(N, Coef, Flag)
If Flag <> 1 Then
        Cls
        Print
        Print "** SOLUTION    of "; N; " Equations are:"
        Print
        For I = 1 To N
                Print "X("; I; ") = "; Coef(I)
        Next I
        Print
End If
End Sub
```

```
Sub Gauss(N, A, Z, Coef, Flag)
'
'     ********************
'          SUBROUTINE GAUSS
'     ********************
'
' N          = No. of Simultaneous Equations
' A          = Coefficient Matrix
' Z          = Constant Vector
' Coef     = Solution Vector
' Flag     = Solution and/or Error Flag
'                    Flag = 1 for matrix singular
'
Flag = 0
'
'-----START
'
For I = 1 To N - 1
       BIG = Abs(A(I, I))
       L% = I: Rem L%= BIG ELEM.
       IP1 = I + 1
'
'-----CHECK FOR BIG ELEMENT
'
For J = IP1 To N
              If (Abs(A(J, I)) > BIG) Then
                     BIG = Abs(A(J, I))
                     L% = J
              End If
       Next J
       If (BIG = 0!) Then
              Flag = 1
              Exit For
       End If
'
'-----PIVOT
'
If (L% <> I) Then
              For J = 1 To N
                     HOLD = A(L%, J)
                     A(L%, J) = A(I, J)
                     A(I, J) = HOLD
              Next J
              HOLD = Z(L%)
              Z(L%) = Z(I)
              Z(I) = HOLD
       End If
'
```

```
'-----ELIMINATION
'
        For J = IP1 To N
            TEMP = A(J, I) / A(I, I)
            For K = IP1 To N
                A(J, K) = A(J, K) - TEMP * A(I, K)
            Next K
                Z(J) = Z(J) - TEMP * Z(I)
        Next J
Next I
'
'-----BACK SUBSTITUTION
'
If Flag = 0 Then
        If (A(N, N) = 0) Then
            Flag = 1
        Else
            Coef(N) = Z(N) / A(N, N)
            For I = N - 1 To 1 Step -1
                TEMP = 0
                For J = I + 1 To N
                    TEMP = TEMP + A(I, J) * Coef(J)
                Next J
                Coef(I) = (Z(I) - TEMP) / A(I, I)
            Next I
        End If
End If
'
'-----RETURN TO USER'S PROGRAM
'
If (Flag = 1) Then
        Print "ERROR: Matrix singular !!"
End If
End Sub


Private Sub Start_Click()
Xpos = 8500
Ypos = 6000
Call GaussianElimination(Xpos, Ypos)
End Sub
```

副程式使用說明：

1. 主要副程式 *Sub Gauss(N, A, Z, Coef, Flag)* 的使用方法如下：

(1) 宣告方程式個數 N。

(2) 輸入方程式係數矩陣 A(I, J) 及 Z(I)。

(3) 執行副程式 *Call Gauss(N, A, Z, Coef, Flag)*。

(4) 得到結果爲 Coef(I)。

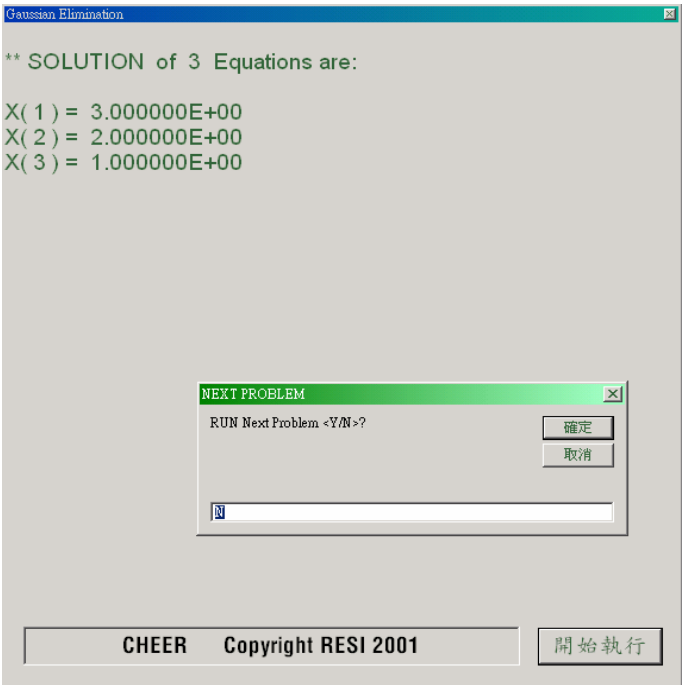
(5) 所求得結果的正確與否，利用 Flag 表示，若 Flag=1 表示方程式無解。

測試數據及結果：

依下列方程式輸入方程式係數

$$\begin{cases} -3x - y + 11z = 0 \\ 13x - 8y - 3z = 20 \\ -8x + 10y - z = -5 \end{cases}$$

所得結果如下圖所示。



第三節　高斯佐丹法

Visual Basic

高斯佐丹法基本上與高斯消去法相當類似，但是高斯佐丹法也較高斯消去法略爲複雜，高斯消去法是將主對角線以下的元素消去成爲零，而將主對角線上各元素變成1。高斯佐丹法求解時，則是將對角線以外的元素仿照高斯消去法的原理完全消去，剩下對角線元素，並將對角線元素完全變成 1。因此，高斯佐丹法除了能解出方程式的解以外，同時更進一步可以求得常數矩陣的反矩陣，因此，使用價值相對較高。

但由於高斯佐丹法和高斯消去法運算方式相似，因此，與高斯消去法具有相同的優點及缺點。執行時間約與矩陣階次 n 的三次方成正比，且由於運用大量的乘、除及加法運算，因此，準確度較差。

假設一線性聯立方程式以矩陣符號表示爲

$$\underline{\underline{A}}\,\underline{\underline{X}} = \underline{\underline{B}} \qquad \textbf{(4-3.1)}$$

將矩陣 $\underline{\underline{A}}_{nxn}$ 利用消去法將非對角線元素完全消去，且使主對角線元素變成 1，即等於將 $\underline{\underline{A}}_{nxn}$ 變成 $\underline{\underline{I}}_{nxn}$ (單位矩陣)。由於

$$\underline{\underline{A}}^{-1}\,\underline{\underline{A}} = \underline{\underline{I}} \qquad \textbf{(4-3.2)}$$

故將方程式 (4-3.1) 兩邊各乘以反矩陣 $\underline{\underline{A}}^{-1}$，得到

$$\underline{\underline{I}}\,\underline{\underline{X}} = \underline{\underline{A}}^{-1}\,\underline{\underline{B}} \qquad \textbf{(4-3.3)}$$

在化學工程程序質能平衡計算時，我們常需假設數種進料組成或流量，分別求解以後，以求得最適設計條件。如設計問題 D-IV，進料 S_1 的組成或流量改變時，方程式 (4-1.1) 至 (4-1.24) 的係數矩陣並不受影響，只有常數矩陣改變。由於利用高斯佐丹法求解 (4-3.1) 時，可以同時求得 $\underline{\underline{A}}^{-1}_{nxn}$ ，因此，常數矩陣 $\underline{\underline{B}}_{nxm}$ 中若同時存放 m 組不同的數據，則在高斯佐丹消去後，利用 $\underline{\underline{A}}^{-1}_{nxn}$ 與 $\underline{\underline{B}}_{nxm}$ 的乘積，即可同時求得 m 組不同條件下的解答 $\underline{\underline{X}}_{nxm}$ 。

程式規劃時，可將反矩陣 $\underline{\underline{A}}^{-1}_{nxn}$ 與矩陣 $\underline{\underline{A}}_{nxn}$ 放在同一存放位置，所得到的方程式解 $\underline{\underline{X}}_{nxm}$ 及常數矩陣 $\underline{\underline{B}}_{nxm}$ 亦可放在同一位置，亦即呼叫高斯佐丹法副程式前放置係數矩陣及常數矩陣的 $\underline{\underline{A}}_{nxn}$ 及 $\underline{\underline{B}}_{nxm}$ ，在執行高斯佐丹消去法後，則存放 $\underline{\underline{A}}^{-1}_{nxn}$ 及 $\underline{\underline{X}}_{nxm}$ 。

例題 4-2　高斯佐丹消去法

利用高斯佐丹消去法設計一程式，解方程式 (4-2.1)。

解：

TOP-DOWN 設計

高斯 - 佐丹消去法可用於求解 $\underline{\underline{A}}\,\underline{\underline{X}} = \underline{\underline{B}}$，或求矩陣 $\underline{\underline{A}}$ 之反矩陣 $\underline{\underline{A}}^{-1}$。將矩陣 $\underline{\underline{A}}_{nxn}$ 放在 A (I , J) 中；常數矩陣 $\underline{\underline{B}}_{nxm}$ 放在 Z (I , J) 中，J = 1, 2, ……, NVEC (注意 NVEC 即前述的 m)，表示 NEVC 種不同條件下之常數向量。所得的解 $\underline{X}_{nxm}$ 放回 Z (I , J) 中，反矩陣則放在 A (I , J) 中，再回至原呼叫程式。

高斯 - 佐丹消去法程式設計基本步驟與高斯消去法相當類似：

1. 先找尋行及列的最大元素 BIG；
2. 以 BIG 爲樞紐進行消去運算；
3. 重複步驟(1)及(2)至非對角線元素均爲 0；
4. 求得 $\underline{\underline{A}}^{-1}_{nxn}$ 或解答。

主　程　式

```
輸入 NVEC
輸入 A (I，J) 及 Z (I，L)
列印 A (I，J) 及 Z (I，L)
CALL Gauss Jordan
列印結果 Z (I，J)
執行下一作業
```

GaussJordan 副程式

```
@清除錯誤旗幟、PIVOT 旗幟及判別式起始值
    FLAG = 0
    Do for I = 1 to NEQN%
        Index(I,3) = 0
    Det = 0
@執行較大元素轉置作業
    Do for I = 1 to NEQN%
```

```
@確定 BIG 位置
BIG = 0
Do for J = 1 to NEQN%
    Do while Index(J,3) <> 1
        Do for K = 1 to NEQN%
            IF Index(K,3)>1
            Then                                  Else
            FLAG = 1                              IF Index(K,3) = 1 or BIG> | Ajk |
                                                  Then                    Else
            Exit Sub                              Irow% = J
                                                  Icol% = K
                                                  BIG =    | Ajk |
@設定 PIVOT 旗幟
    Index(Icol%,3)=Index(Icol%,3)+1
@調整 PIVOT 位置
    Index(I,1) = Irow%
    Index(I,2) = Icol%
@將 PIVOT 元素移動到主對角線
  Do while Irow% <> Icol%
      DET = - DET
        Do for L = 1 to NEQN%
          Swap A(Irow%,L), A(Icol%,L)
      Do while NVEC >= 1
            Do for L = 1 to NEQN%
              Swap Z(Irow%,L), Z(Icol%,L)
@將最大元素提出，使其變成 1
  PIVOT = A(Icol%, Icol%)
  DET = DET * PIVOT
  Do for L = 1 to NEQN%
      A(Icol%, L) = A(Icol%,L)/PIVOT
    Do while NVEC>=1
      Do for L = 1 to NEQN%
          Z(Icol%,L) = Z(Icol%,L)/PIVOT
@消去對角線元素
Do for L = 1 to NEQN%
      Do while L<>Icol%
          Temp = A(L,Icol%)
          A(L,Icol%) = 0
          Do for LL = 1 to NEQN%
              A(L,LL) = A(L,LL)-A(Icol%,LL)*Temp
          Do while NVEC>=1
              Do for LL = 1 to NEQN%
```

```
                                        Z(L,LL)=Z(L,LL)-Z(Icol%,LL)*Temp
@行間交換及排列
    Do for I = 1 to NEQN%
        L = NEQN% - I + 1
        Do while Index(L, 1) <> Index(L,2)
            Irow% = Index(L,1)
            Icol% = Index(L,2)
            Do for K = 1 to NEQN%
                Swap    A(K,Irow%), A(K,Icol%)
@設定旗號並返回主程式
```

符號說明：

A：　係數矩陣，A (N , N)，執行高斯佐丹法前爲 $\underline{\underline{A}}$，執行後爲 $\underline{\underline{A}}^{-1}$

BIG：　暫存 A (I , J) 之最大元素，樞紐元素。

DET：　矩陣 $\underline{\underline{A}}$ 之行列式值

FLAG：　錯誤旗幟，0 表正確，1 表錯誤 (無解)

HOLD：　暫存值

INDEX：　工作矩陣，

INDEX (I , 1) 存放列轉軸元素位置；

INDEX (I , 2) 存放行轉軸元素位置；

INDEX (I , 3) 存放 BIG 旗幟，0 表未找到 BIG，1 表已找到

NEQN%：方程式數目

NVEC：　常數向量組數。NVEC = 0 時求 $\underline{\underline{A}}^{-1}$

PIVOT：　對角線樞紐元素

TEMP：　暫存值

Z：　常數向量及方程組之解

程式列印：

```
Sub GaussJordanElimination(Xpos, Ypos)
'       ********************************************
'       * GAUSSIAN JORDAN ELIMINATION
'       ********************************************
'
'    FLAG    = ERROR FLAG
'    NEQN    = NO. OF EQUNS.
'    A       = COEFF. MATRIX
```

```
'     Z         = CONSTANT VECTOR / RETURN SOLUTION VECTOR
'     INDEX = WORK MATRIX
'     DET      = DETERMINANT
'     INV%    = PRINT INVERSE FLAG
'     NVEC    = NO. OF CONSTANT VECTOR
'     NVEC    = 0 FOR MATRIX INVERSION
Dim A(100, 100), Z(100, 5), Index(100, 3)
Do
      Cls
      Print "SOLUTION OF SIMULTANEOUS LINEAR EQUATION"
      Print " BY GAUSS-JORDAN ELIMINATION WITH PIVOTING"
      Print
      Print " No. of Equations N = ";
      NEQN% = Val(InputBox("NO. OF EQUATIONS = ", "", , Xpos, Ypos))
      Print NEQN%
      Print
      Print "No. of Constant Vector (0 for matrix inversion) Nvec = ";
      Nvec = Val(InputBox("NO. OF CONSTANT VECTOR =", Nvec, 1, Xpos, Ypos))
      Print Nvec
      MsgBox ("Ready to Enter Equation Parameters")
      Call EnterEquations(NEQN%, Nvec, A, Z, Xpos, Ypos)
      Call GaussJordan(NEQN%, Nvec, A, Z, Index, Flag)
      Call EliminationResults(NEQN%, Nvec, Z, Flag)
      NewProjYN$ = InputBox("RUN Next Problem <Y/N>?", "", "N", Xpos, Ypos)
Loop While NewProjYN$ <> "N" And NewProjYN$ <> "n"
End Sub


Sub EnterEquations(N, Nvec, A, Z, Xpos, Ypos)
For I = 1 To N
      Do
            Cls
            Print
            Print "EQUATION #"; I; " :"
            Print
            For J = 1 To N
                  Print "COEFF#"; J; " = ";
                  A(I, J) = Val(InputBox("Enter Coefficient ", "A(I,J)", , Xpos, Ypos))
                  Print A(I, J)
            Next J
            If Nvec > 0 Then
                  For J = 1 To Nvec
                        Print "Constant C("; I; ","; J; ") =";
                        Z(I, J) = Val(InputBox("Enter Z(I,J) =", "Z(I,J)", , Xpos, Ypos))
                        Print Z(I, J)
                  Next J
            End If
            CorrectYN$ = InputBox("Coeff Correct <Y/N>?", "", "Y", Xpos, Ypos)
```

```
        Loop While CorrectYN$ = "N" Or CorrectYN$ = "n"
Next I
End Sub


Sub EliminationResults(NEQN%, Nvec, Z, Flag)
If Flag <> 1 And Nvec > 0 Then
        Cls
        Print
        Print "** SOLUTION    of "; NEQN%; " Equations are:"
        Print
        For J = 1 To Nvec
                Print "SET #"; J
                For I = 1 To NEQN%
                        Print "             "; Z(I, J)
                Next I
                Print
        Next J
End If
End Sub


Sub GaussJordan(NEQN%, Nvec, A, Z, Index, Flag)
'
'     GAUSS-JORDAN SUBROUTINE
'
' NEQN%          = No. of Equations
' Nvec           = No. of Constant Vectors
' A              = Matrix, Equation Coefficients
' Z              = Multi-row Constant Vectors / Return Solution Vector
' Index          = Work Matrix
' Flag           = Error Flag, 1 = matrix singular
'
Flag = 0:    ' CLEAR ERROR
If Nvec = 0 Then Inv% = 1
For I = 1 To NEQN%
        Index(I, 3) = 0
Next I
DET = 0
For I = 1 To NEQN%
'
'    -- SEARCH FOR PIVOT ELEMENT
        BIG = 0
        For J = 1 To NEQN%
                If (Index(J, 3) <> 1) Then
                    For K = 1 To NEQN%
                            If (Index(K, 3) > 1) Then
                                Flag = 1
```

```
                                Print "Flag = 0 @ Index("; K; ",3) >1"
                                Exit For
                                Exit For
                                Exit For
                            ElseIf (Index(K, 3) < 1) And (BIG < Abs(A(J, K))) Then
                                Irow% = J
                                Icol% = K
                                BIG = Abs(A(J, K))
                            End If
                        Next K
                    End If
                Next J
                Index(Icol%, 3) = Index(Icol%, 3) + 1
                Index(I, 1) = Irow%
                Index(I, 2) = Icol%
        '
        '   -- PUT PIVOT ON DIAGONAL
                If (Irow% <> Icol%) Then
                    DET = -DET
                    For L = 1 To NEQN%
                        Temp = A(Irow%, L)
                        A(Irow%, L) = A(Icol%, L)
                        A(Icol%, L) = Temp
                    Next L
                    If Nvec >= 1 Then
                        For L = 1 To Nvec
                            Temp = Z(Irow%, L)
                            Z(Irow%, L) = Z(Icol%, L)
                            Z(Icol%, L) = Temp
                        Next L
                    End If
                End If
        '
        ' --- (PIVOT ROW) / (PIVOT COLUMN)
                Pivot = A(Icol%, Icol%)
                DET = DET * Pivot
                A(Icol%, Icol%) = 1!
                For L = 1 To NEQN%
                    A(Icol%, L) = A(Icol%, L) / Pivot
                Next L
                If (Nvec >= 1) Then
                    For L = 1 To Nvec
                        Z(Icol%, L) = Z(Icol%, L) / Pivot
                    Next L
                End If
        '
        '   -- REDUCE NONPIVOT ROWS
                For L = 1 To NEQN%
```

```
            If (L <> Icol%) Then
                Temp = A(L, Icol%)
                A(L, Icol%) = 0!
                For LL = 1 To NEQN%
                    A(L, LL) = A(L, LL) - A(Icol%, LL) * Temp
                Next LL
                If (Nvec >= 1) Then
                    For LL = 1 To Nvec
                        Z(L, LL) = Z(L, LL) - Z(Icol%, LL) * Temp
                    Next LL
                End If
            End If
        Next L
    Next I
    '
    '    -- INTERLCHANGE COLUMNS
    If Flag <> 1 Then
        For I = 1 To NEQN%
            L = NEQN% - I + 1
            If (Index(L, 1) <> Index(L, 2)) Then
                Irow% = Index(L, 1)
                Icol% = Index(L, 2)
                For K = 1 To NEQN%
                    Temp = A(K, Irow%)
                    A(K, Irow%) = A(K, Icol%)
                    A(K, Icol%) = Temp
                Next K
            End If
        Next I
        For K = 1 To NEQN%
            If (Index(K, 3) <> 1) Then Flag = 1
        Next K
        Flag = 0
        If (Inv% = 1) Then
            If (DET = 0) Then
                Flag = 1
            Else
                Print
                Print "** Matrix Inverse : "
                Print
                For I = 1 To NEQN%
                    For J = 1 To NEQN%
                        Print A(I, J);
                    Next J
                    Print
                Next I
                Print
                Print "Determinant = "; DET
```

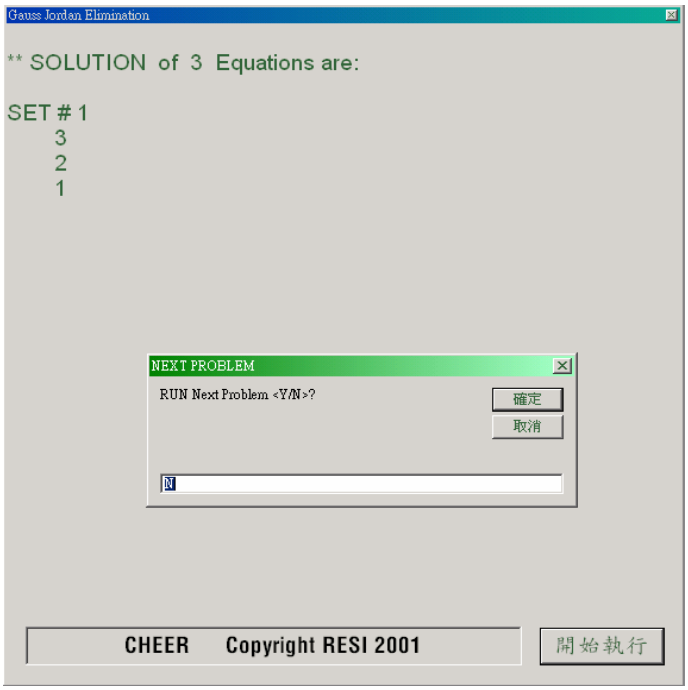
```
                    Print
                End If
            End If
        End If
    '
    '    -- RETURN TO USER'S PROGRAM
    If (Flag = 1) Then Print "ERROR: Matrix singular !!"
    End Sub
```

測試數據及結果：

依下列方程式輸入方程式係數

$$\begin{cases} -3x - y + 11z = 0 \\ 13x - 8y - 3z = 20 \\ -8x + 10y - z = -5 \end{cases}$$

所得結果如下圖所示。



第四節　高斯賽德迭代法

Visual Basic

前面兩節所述高斯消去法及高斯佐丹消去法由於使用大量的乘、除及加法運算，因此，矩陣方程式較大時，運算所產生的誤差將變的愈爲明顯，而使這兩種方法所得結果變成無意義。遇到這種情況，可考慮使用高斯賽德迭代法 (Gauss-Seidel Iterative method)。高斯賽德迭代法是利用依序迭代求解聯立線性方程組的方法。其基本做法是給定一最初近似值，然後利用方程式進行重複迭代，直到結果接近真正解答爲止。由於這種方法每一次近似計算只跟前一近似值有關，因此，捨入誤差 (Round-off error) 不會因連續運算而累積。此外，由於採用迭代趨近法，因此，方程式非線性時亦可採用同一策略進行求解。

考慮方程式 (4-2.1) 所示之聯立線性方程組

$$\begin{cases} -3x - y + 11z = 0 \\ 13x - 8y - 3z = 20 \\ -8x + 10y - z = -5 \end{cases} \quad \textbf{(4-2.1)}$$

我們可利用第一式求得 x 的迭代表示式：

$$x = \frac{y - 11z}{-3} \quad \textbf{(4-4.1)}$$

式中 x 爲前一迭代所得 y 及 z 值的函數，例如，最初近似值設 x，y，z 均爲 0，則第一次迭代所得 x 值亦爲 0。同理利用第二式求得 y 的迭代近似表示式爲：

$$y = \frac{-13x + 3z + 20}{-8} \quad \textbf{(4-4.2)}$$

此時，x 值仍爲 0，z 值亦爲 0，代入方程式中得到 $y = -2.5$。利用第三式求解 z，得

$$z = \frac{8x - 10y - 5}{-1} \quad \textbf{(4-4.3)}$$

此時，x 值爲 0，y 值的第一次迭代值爲 −2.5，故得到 z 的第一次迭代值爲 $z = -20$，利用此一策略所得迭代結果如下：

迭代次數	$x=\frac{y-11z}{-3}$	$y=\frac{-13x+3z+20}{-8}$	$z=\frac{8x-10y-5}{-1}$
0	0.00	0.00	0.00
1	0.00	–2.50	–20.00
2	–72.50	–112.81	–543.13
3	–1,953.85	–2,973.84	–14,102.58
4	–50,718.17	–77,131.06	–365,560.26
5	–1,314,677.25	–1,999,267.94	–9,475,256.39
6	–34,076,184.11	–51,820,580.54	–245,596,327.47
7	–883,246,340.53	–1,343,176,683.06	–6,365,796,101.38

很顯然地，迭代結果並不收斂而且快速發散。可以推想，應該是處理策略上不正確所導致。觀察方程式 (4-4.1)，若 *z* 與真實的解有誤差，則由方程式可知，*x* 的誤差將與 *z* 的誤差擴增 11 倍。同理，由方程式 (4-4.2) 可知 *y* 的誤差將比 *x* 的誤差擴大 13 倍。*z* 的誤差將比 *y* 的誤差擴大 10 倍。因此，會因迭代計算使得誤差快速成長。

根據這種觀察，我們若能將方程式的最大係數作爲除數，則應該能使誤差逐漸減小。以下我們以同一組方程式爲例，實際了解策略上的改變對迭代收斂性的影響。首先將方程式 (4-2.1) 的次序調動，使最大元素均在主對角線上：

$$\begin{cases} 13x-8y-3z=20 \\ -8x+10y-z=-5 \\ -3x-y+11z=0 \end{cases} \quad \textbf{(4-4.4)}$$

仿照前面的做法，則對應的迭代方程式可變成

$$x=\frac{20+8y+3z}{13} \quad \textbf{(4-4.5)}$$

$$y=\frac{8x+z-5}{10} \quad \textbf{(4-4.6)}$$

$$z=\frac{3x+y}{11} \quad \textbf{(4-4.7)}$$

觀察這些方程式，*由於除數均爲該方程式中之最大數，其他變數的誤差值經演算後應該都會被乘以一個比 1 小的數字，使得誤差逐漸縮小。*

利用方程式 (4-4.5)、(4-4.6) 及 (4-4.7)，仍由 *x*，*y*，*z* 均爲 0 開始迭代，所得迭代結果爲：

迭代次數	$x=\frac{20+8y+3z}{13}$	$y=\frac{8x+z-5}{10}$	$z=\frac{3x+y}{11}$
0	0.000000	0.000000	0.000000
1	1.538462	0.730769	0.486014
2	2.100323	1.228860	0.684530
3	2.452651	1.530574	0.808048
4	2.666826	1.714265	0.883158
5	2.797200	1.826076	0.928880
21	2.999928	1.999938	0.999975
22	2.999956	1.999962	0.999985
23	2.999973	1.999977	0.999991

由以上結果可發現迭代值快速趨近 (3，2，1)。上述兩種迭代所得結果完全不同，唯一的差別是後者將係數矩陣的最大元素均排列在主對角線上。因此，利用高斯賽德迭代法設計程式時，必須將係數矩陣的最大元素均排列在主對角線上。

此外，設計程式時，亦可引入緩衝係數 (λ)，使迭代的收斂速率增快。例如，上述迭代法第 J 次所得 X 值爲 $X(J)$，次一迭代所得爲 X_1，則利用緩衝法，所得新的 $X(J+1)$ 值爲

$$X(J+1)=\lambda * X_1+(1-\lambda)*X(J) \qquad \textbf{(4-4.8)}$$

當 $\lambda=1$ 時，即與前述未作緩衝處理的方法相同。一般而言，緩衝係數 λ 值範圍介於 0 至 2 間，亦即 $0<\lambda<2$。

例題 4-3　高斯賽德迭代法

利用高斯賽德迭代法設計一求解線性聯立方程式的計算機程式，並求解方程式 (4-2.1)。

解：

TOP-DOWN 設計

高斯賽德迭代法可用於求解聯立線性方程式 $\underline{\underline{A}}\,\underline{X}=\underline{B}$。程式設計時，將係數矩陣 $\underline{\underline{A}}$ 放在 A (I , J)，$\underline{B}$ 放在 Z (I)，迭代所得結果放在 COEF (I)。

高斯賽德迭代法程式基本設計主要分成兩大步驟：

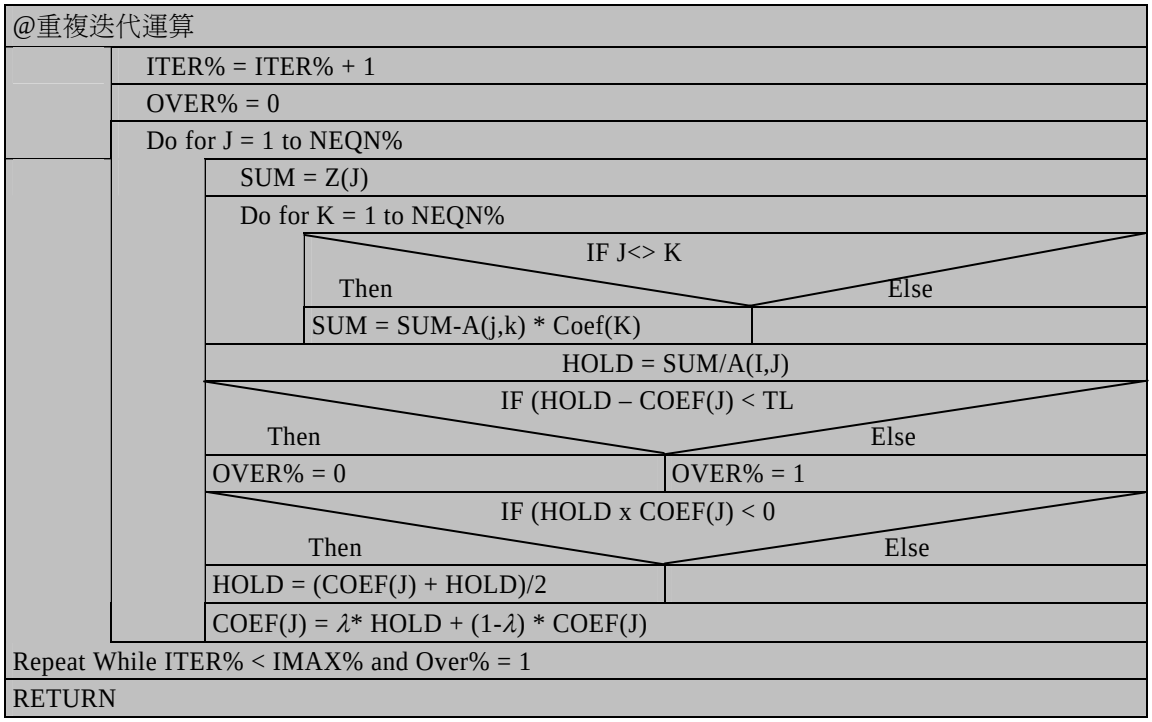
1. 找尋各行列最大元素，將它放在主對角線上；
2. 進行迭代運算至滿足收斂條件爲止。

主 程 式

	輸入方程式數目 NEQN%
	輸入 A(I , J)及 Z (I)
	輸入收斂誤差容忍度 Tolerance
	輸入緩衝係數 λ
	輸入最大迭代次數 IMAX%
	列印 A 及 z
	CALL GaussSeider (SUBROUTINE Gauss-Seider)
	列印結果 COEF (I)
執行下一作業	

副程式 **GaussSeider**

@清除錯誤旗幟		
FLAG = 0		
@尋找 BIG 並將它放置在主對角線上		
Do for I = 1 to NEQN% -1		
	BIG = $\lvert A_{ij} \rvert$	
	L=I	
	Do for J = (I+1) to NEQN%	
	IF $\lvert A_{ij} \rvert$ < BIG	
	Then	Else
		BIG = $\lvert A_{ij} \rvert$
		L=J
	IF BIG = 0	
	Then	Else
	FLAG = 1	IF L = 1
		Then Else
	Exit Sub	Do for J = 1 to NEQN%
		Swap A(L,J),A(I,J)
		Swap Z(L),Z(I)
@檢查主對角線元素是否爲 0		
IF A(NEQN%,NEQN%) = 0		
Then		Else
FLAG = 1		
Exit Sub		
@歸零，準備開始迭代		
Do for I = 1 to NEQN%		
	COEF(I) = 0	
ITER% = 0		



符號說明：

A：　係數矩陣，A (N , N)

BIG：　暫存 A (I , J) 元素之最大值

COEF：　解答向量

FLAG：　錯誤旗幟，1 表示無解

HOLD：　暫存值

ITER%：　迭代次數

LAMBDA：緩衝係數，λ

NEQN%：　方程式數目

OVER%：　收斂旗幟，1 表示收斂，0 表已達收斂條件

Tolerance：　收斂容忍條件，絕對誤差

程式列印：

```
Sub GaussSeiderElimination(Xpos, Ypos)
'       ******************************************
'         GAUSS - SEIDER ITERATION
'       ******************************************
```

```
'     A                 = COEFF. MATRIX
'     COEF              = SOLUTION VECTOR
'     FLAG              = ERROR FLAG
'     LAMBDA            = RELAXATION FACTOR
'     NEQN              = NO. OF EQUNS.
'     Tolerance         = ABS. ERROR TOLERANCE
'     Z                 = CONSTANT VECTOR

Dim A(100, 100), Z(100), Coef(100)
Do
      Cls
      Print "SOLUTION OF SIMULTANEOUS LINEAR EQUATION"
      Print " BY GAUSS-JORDAN ELIMINATION WITH PIVOTING"
      Print

      Print "No. of Equations N = ";
      NEQN% = Val(InputBox("NO. OF EQUATIONS = ",, , Xpos, Ypos))
      Print NEQN%
      Print "Error Tolerance = ";
      Tolerance = Val(InputBox("Error Tolerance = ", "", 0.00001, Xpos, Ypos))
      Print Tolerance

      Print "Relaxation Factor = ";
      Lambda = Val(InputBox("Relaxation Factor [0< ; >2] = ",, 0.5, Xpos, Ypos))
      Print Lambda

      Print "Maximum Iteration Number = ";
      Imax% = Val(InputBox("MAX. ITERATION NUMBER = ",, 10, Xpos, Ypos))
      Print Imax%
      MsgBox ("Ready to Enter Equation Parameters")

      Call EnterEquations(NEQN%, A, Z, Xpos, Ypos)
      Call GaussSeider(NEQN%, A, Z, Coef, Flag%, Imax%, Lambda, Tolerance)
      Call EliminationResults(NEQN%, Coef, Flag%)

      NewProjYN$ = InputBox("RUN Next Problem <Y/N>?", "", "N", Xpos, Ypos)
Loop While NewProjYN$ <> "N" And NewProjYN$ <> "n"
End Sub


Sub EnterEquations(N, A, Z, Xpos, Ypos)

For I = 1 To N
      Do
            Cls
            Print
            Print "EQUATION #"; I; " :"
            Print
            For J = 1 To N
                  Print "COEFF#"; J; " = ";
```

```
                A(I, J) = Val(InputBox("Enter Coefficient ", "A(I,J)", , Xpos, Ypos))
                Print A(I, J)
            Next J
            Print "Constant = ";
            Z(I) = Val(InputBox("Enter Z(I) =", "Z(I)", , Xpos, Ypos))
            Print Z(I)
            CorrectYN$ = InputBox("Coeff Correct <Y/N>?", "", "Y", Xpos, Ypos)
        Loop While CorrectYN$ = "N" Or CorrectYN$ = "n"
Next I
End Sub


Sub EliminationResults(NEQN%, Coef, Flag%)

If Flag% <> 1 Then
        Print
        Print "** SOLUTION     of "; NEQN%; " Equations are:"
        For I = 1 To NEQN%
            Print Format(Coef(I), "    0.000000E+00");
        Next I
        Print
End If
End Sub


Sub GaussSeider(NEQN%, A, Z, Coef, Flag, Imax%, Lambda, Tolerance)
'
' Clear Error Flag and Registers

Flag = 0
For I = 1 To NEQN% - 1
        BIG = Abs(A(I, I))
        L = I
        For J = I + 1 To NEQN%
            If (Abs(A(J, I)) > BIG) Then
                BIG = Abs(A(J, I))
                L = J
            End If
        Next J
        If BIG = 0 Then
            Flag = 1
            Exit For
        ElseIf L <> I Then
            For J = 1 To NEQN%
                HOLD = A(L, J)
                A(L, J) = A(I, J)
                A(I, J) = HOLD
            Next J
            HOLD = Z(L)
            Z(L) = Z(I)
```

```
            Z(I) = HOLD
        End If
    Next I
    '
    ' INITIALIZE
    Cls
    If (A(NEQN%, NEQN%) = 0) Then
        Flag = 1
    Else
        For I = 1 To NEQN%
            Coef(I) = 0
        Next I
    '
    '   -- START INTERATION
        Print
        Print "ITERATION"
        Iter% = 0

        Do
            Iter% = Iter% + 1
            If (Int(Iter% / 50) * 50 = Iter%) Then Print
            OVER% = 0
            For J = 1 To NEQN%
                Sum = Z(J)
                For K = 1 To NEQN%
                    If (J <> K) Then Sum = Sum - A(J, K) * Coef(K)
                Next K
                HOLD = Sum / A(J, J)
                If (Abs(HOLD - Coef(J)) < Tolerance) Then
                    OVER% = 0
                Else
                    OVER% = 1
                End If
                If (HOLD * Coef(J) < 0) Then HOLD = (Coef(J) + HOLD) / 2
                Coef(J) = Lambda * HOLD + (1 - Lambda) * Coef(J)
            Next J
        Loop While (Iter% <= Imax%) And (OVER% = 1)

        If (OVER% <> 0) Then Flag = 1
    End If
    '
    '     -- CHECK FLAG AND RETURN
    If Flag = 1 Then Print " ** No solution or solution is not convergent!!"
    '
    '-- CHEER by Ron Hsin Chang, Copyright 2001
    End Sub
```

副程式使用方法：

1. 副程式 GaussSeider (NEQN%, A, Z, Coef, Flag, Imax%, Lambda, Tolerance) 的使用方法如下：

(1) 輸入方程式數目 NEQN%、係數矩陣 A 及 Z。

(2) 輸入誤差容忍度 Tolerance、緩衝係數 Lambda、及最大迭代數 Imax%。

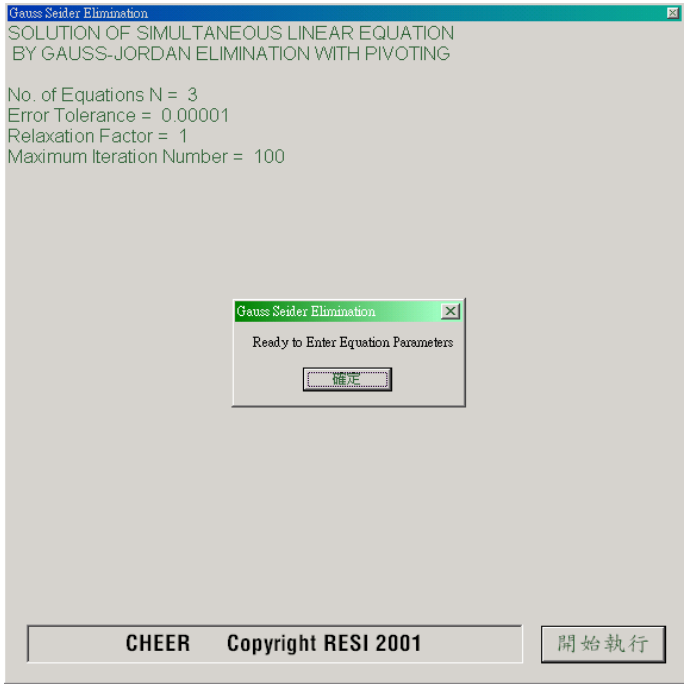
(3) 呼叫副程式 Call GaussSeider (NEQN%, A, Z, Coef, Flag, Imax%, Lambda, Tolerance)。

(4) 解答狀態旗幟爲 Flag，結果爲 Coef。

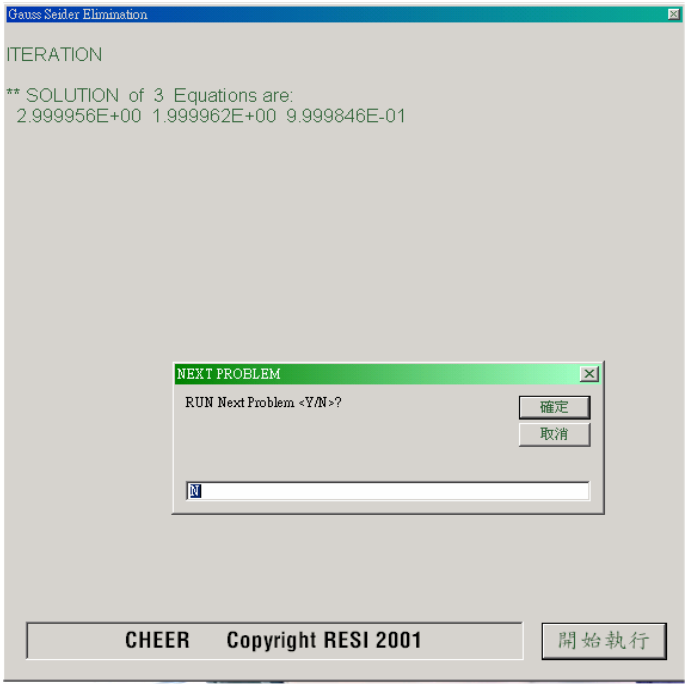
測試數據及結果：

輸入方程組如下：

$$\begin{cases} -3x - y + 11z = 0 \\ 13x - 8y - 3z = 20 \\ -8x + 10y - z = -5 \end{cases}$$



所得結果如下：



第五節　聯立線性方程式之解法比較

Visual Basic

高斯消去法及高斯佐丹消去法需涉及大量的乘除及加法運算，因此，當係數矩陣階數較高時，其捨入誤差可能變得相當可觀。高斯賽德迭代法由於利用重複迭代逼近正確解，因此，沒有捨入誤差累積的問題，但有時無法收斂且矩陣階數低時，迭代法通常較爲耗時。比較三種方法的算數運算次數分別爲：

高斯消去法 (全部過程)	高斯佐丹法 (全部過程)	高斯賽德法 (每次迭代)
$\frac{2}{3}n^3$	n^3	$2\,n^2$

因此，以 n = 100 爲例，利用高斯消去法約需作 68,1550 次算數運算，利用高斯賽德迭代法每次迭代約需運算 19,900 次，因此，若迭代次數超過 34 次 (約爲 n / 3)，則後者可能略爲費時。但後者縱使略爲費時，由於其捨入誤差小，故仍甚具價值。

例題 4-4　質能平衡計算

利用本章所介紹的聯立線性方程式解法，求解設計問題 D-IV。

輸入數據：

設計問題 D-IV，如第一節所述，可寫成 24 個線性聯立方程式。以矩陣方式表示成 $\underline{\underline{A}}\,\underline{\underline{X}} = \underline{\underline{Z}}$，則係數矩陣及常數向量分別如圖 4.2 所示。

測試結果：

以高斯消去法及高斯佐丹消去法求解 D-IV，所的結果完全一樣，但後者所使用時間約爲前者的五倍。

```
** SOLUTION
8.200000E+01        6.000000E+02        2.977778E+02
7.700000E+02        3.500000E+02        1.075556E+03
7.000000E+02        3.000000E+02        4.888889E+02
7.000000E+01        5.000000E+01        5.866667E+02
4.200000E+02        2.000000E+02        3.9111111E+02
7.000000E+01        5.000000E+01        8.8000000E+02
2.800000E+02        1.000000E+01        9.7777778E+01
4.200000E+02        2.000000E+02        9.7777778E+01
```

直接利用圖 4.2 所示矩陣輸入高斯 - 賽德迭代法程式，無法獲得收斂結果。其原因是本章所附程式只檢查主對角線左下角之最大元素，而由圖 4.2 可知，右上角仍留有相當大數值的元素，故使計算值發散。解決方法有三：

1. 搬動方程式，使右上角元素均較主對角線元素小；
2. 若右側元素較大，則將該方程式除以該元素，使主對角線右側值變成 1，例如第 7 個方程式由 $-(S4A) + 10\,(S5A) = 0$ 改寫成 $-0.1(S4A) + (S5A) = 0$。
3. 改寫高斯-賽德迭代副程式，使它具有完全搜尋 BIG 元素之能力。

利用第二種方法，將第 7，8，9，13，14，15 及 21 個方程式處理後，輸入高斯 - 賽德迭迭代程式即可在 39 次迭代得到誤差爲 10^{-4} 的結果。使用時間約爲高斯消去法的 5 倍。

** SOLUTION

8.19998E+01	6.00000E+02	2.97777E+02
7.69997E+02	3.50000E+02	1.07555E+03
6.99998E+02	3.00000E+02	4.88888E+02
6.99997E+01	5.00000E+01	5.86666E+02
4.19999E+02	2.00000E+02	3.91110E+02
6.99997E+01	5.00000E+01	8.79999E+02
2.79999E+02	1.00000E+01	9.77776E+01
4.19999E+02	2.00000E+02	9.77776E+01

S2			S3			S4			S5			S6			S7			S8			S9			Z
A	B	C	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C	
1																					−1			400
	1																					−1		400
		1																					−1	200
0.7			−1															0.7						0
	0.5			−1															0.5					0
0.3	0.5	1			−1													0.3	0.5	1				0
						−1			10															0
							−1			6														0
								−6			5													0
			1			−1			−1															0
				1			−1			−1														0
					1			−1			−1													0
												2						−3						0
													1						−2					0
														1						−4				0
						1						−1						−1						0
							1						−1						−1					0
								1						−1						−1				0
															6						−1			0
																4						−1		0
																	1						−9	0
									1			1			−1						−1			0
										1			1			−1						−1		0
											1			1			−1						−1	0

圖 **4.2**　設計問題 D-IV 的係數及常數矩陣

第六節　三對角線矩陣方程式

Visual Basic

三對角線矩陣方程式 (Tridiagonal matrix equation) 或雅可必矩陣 (Jacobi matrix) 方程式，在各種工程及化工應用上極爲常見，例如蒸餾、吸收、反應器 (CSTR) 設計等，或利用有限差分法解邊界值問題時均可能出現。

三對角線矩陣方程式由於其係數矩陣除了三個對角線外，其他元素均爲零，因此，利用高斯消去法求解並不划算。如果將三對角線矩陣方程式視爲一般矩陣方程式進行求解，則需使用 N^2 個記憶位置，且其運算也需耗費很長的時間。以下介紹一種效率較高的處理方式。

典型的三對角線矩陣方程式，如：

$$\underline{\underline{A}}\,\underline{X} = \underline{Z} \tag{4-6.1}$$

$$\begin{bmatrix} \alpha_1 & \gamma_1 & 0 & 0 & 0 & 0 \\ \beta_2 & \alpha_2 & \gamma_2 & 0 & 0 & 0 \\ 0 & \beta_3 & \alpha_3 & \gamma_3 & 0 & 0 \\ 0 & 0 & \beta_4 & \alpha_4 & \gamma_4 & 0 \\ 0 & 0 & 0 & \beta_5 & \alpha_5 & \gamma_5 \\ 0 & 0 & 0 & 0 & \beta_6 & \alpha_6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Z_4 \\ Z_5 \\ Z_6 \end{bmatrix} \tag{4-6.2}$$

係數矩陣中非零元素只有 *(3 N - 2) = 16* 個，如果我們能妥善地利用此性質，則可使求解過程顯著簡化，並提高執行效率。

將係數矩陣上下分解 (Lower-Upper decomposition) 成兩個雙對角線矩陣 (Bi-diagonal form) 之乘積：

$$\underline{\underline{A}}\,\underline{X} \equiv \underline{\underline{L}}\,\underline{\underline{U}}\,\underline{X} = \underline{Z} \tag{4-6.3}$$

$$\begin{bmatrix} a_1 & 0 & 0 & 0 & 0 & 0 \\ b_2 & a_2 & 0 & 0 & 0 & 0 \\ 0 & b_3 & a_3 & 0 & 0 & 0 \\ 0 & 0 & b_4 & a_4 & 0 & 0 \\ 0 & 0 & 0 & b_5 & a_5 & 0 \\ 0 & 0 & 0 & 0 & b_6 & a_6 \end{bmatrix} \begin{bmatrix} 1 & c_1 & 0 & 0 & 0 & 0 \\ 0 & 1 & c_2 & 0 & 0 & 0 \\ 0 & 0 & 1 & c_3 & 0 & 0 \\ 0 & 0 & 0 & 1 & c_4 & 0 \\ 0 & 0 & 0 & 0 & 1 & c_5 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Z_4 \\ Z_5 \\ Z_6 \end{bmatrix} \tag{4-6.4}$$

由於 $\underline{\underline{A}} \equiv \underline{\underline{L}}\,\underline{\underline{U}}$ ，因此，比較對應的各項，可以得到：

$$
\begin{aligned}
&a_1 = \alpha_1 \\
&b_2 = \beta_2 \\
&b_3 = \beta_3 \\
&a_1 c_1 = \gamma_1 \\
&b_2 c_1 + a_2 = \alpha_2 \\
&b_3 c_2 + a_3 = \alpha_3 \qquad \cdots\cdots \\
&a_2 c_2 = \gamma_2 \\
&a_3 c_3 = \gamma_3
\end{aligned}
$$

或寫成

$$
\begin{cases}
a_1 = \alpha_1 & \\
c_1 = \gamma_1 / \alpha_1 & \\
b_i = \beta_i & i = 2,3,\cdots,n \\
a_i = \alpha_i - \beta_i c_{i-1} & i = 2,3,\cdots,n \\
c_i = \gamma_i / a_i & i = 2,3,\cdots,n-1
\end{cases}
\qquad \textbf{(4-6.5)}
$$

由於 $\underline{\underline{L}}\,\underline{\underline{U}}\,\underline{X} = \underline{Z}$ ，令 $\underline{\underline{U}}\,\underline{X} = \underline{F}$ ，則可將原方程式寫成：

$$
\underline{\underline{L}}\,\underline{F} = \underline{Z} \qquad \textbf{(4-6.6)}
$$

或

$$
\begin{bmatrix}
a_1 & 0 & 0 & 0 & 0 & 0 \\
b_2 & a_2 & 0 & 0 & 0 & 0 \\
0 & b_3 & a_3 & 0 & 0 & 0 \\
0 & 0 & b_4 & a_4 & 0 & 0 \\
0 & 0 & 0 & b_5 & a_5 & 0 \\
0 & 0 & 0 & 0 & b_6 & a_6
\end{bmatrix}
\begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \end{bmatrix}
=
\begin{bmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Z_4 \\ Z_5 \\ Z_6 \end{bmatrix}
\qquad \textbf{(4-6.7)}
$$

由上往下代入，解此方程式，得

$$
\begin{cases}
F_1 = Z_1 / a_1 & \\
F_i = (Z_i - b_i F_{i-1}) / a_i & i = 2,3,\cdots\cdots,n
\end{cases}
\qquad \textbf{(4-6.8)}
$$

求得向量 $\underline{F}$ 以後，在利用 $\underline{\underline{U}}\,\underline{X} = \underline{F}$ 求解向量 $\underline{X}$ ，由於

$$\begin{bmatrix} 1 & c_1 & 0 & 0 & 0 & 0 \\ 0 & 1 & c_2 & 0 & 0 & 0 \\ 0 & 0 & 1 & c_3 & 0 & 0 \\ 0 & 0 & 0 & 1 & c_4 & 0 \\ 0 & 0 & 0 & 0 & 1 & c_5 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \end{bmatrix} \qquad \textbf{(4-6.9)}$$

由最下方往上代入 (與高斯消去法很相似的做法)，即可解出 X 爲：

$$\begin{cases} x_n = F_n \\ x_j = F_j - C_j\, x_{j+1} \qquad j = n-1, n-2, \cdots\cdots, 1 \end{cases} \qquad \textbf{(4-6.10)}$$

例題 4-5　三對角線矩陣方程式

試利用本節所介紹方法，解以下聯立方程組

$$\begin{cases} 2x_1 - x_2 = 1 \\ -x_1 + 2x_2 - x_3 = 1 \\ \quad -x_2 + 2x_3 - x_4 = 1 \\ \quad\quad -x_3 + 2x_4 - x_5 = 1 \\ \quad\quad\quad -x_4 + 2x_5 - x_6 = 1 \\ \quad\quad\quad\quad -x_5 + 2x_6 = 1 \end{cases} \qquad \textbf{(4-6.11)}$$

解：

TOP-DOWN 設計：

將三對角線矩陣方程式三對角線值分別存放在一維陣列中，

$$\begin{aligned} \underline{A} &= [\alpha_1 \;\; \alpha_2 \;\; \cdots\cdots \;\; \alpha_n]^T \\ \underline{B} &= [\beta_1 \;\; \beta_2 \;\; \cdots\cdots \;\; \beta_n]^T \qquad \beta_1 = 0 \\ \underline{C} &= [\gamma_1 \;\; \gamma_2 \;\; \cdots\cdots \;\; \gamma_n]^T \qquad \gamma_n = 0 \end{aligned} \qquad \textbf{(4-6.12)}$$

呼叫 JACOBI 副程式時，先依 (4-6.5) 將 A，B，C 元素改寫成 a, b 及 c，再利用 (4-6.8) 及 (4-6.10) 求解 $\underline{X}$。

主 程 式

	輸入 A, B, C, Z
	執行副程式 Call Jacobi
	列印結果
執行下一作業?	

Jacobi 副程式

@方程式(4-6.5)	
C(1) = C(1)/A(1)	
Z(1) = Z(1)/A(1)	
Do for I = 2 t0 N	
	A(I) = A(I) –B(I) * C(I-1)
	C(I) =C(I)/A(I)
	Z(I) = (Z(I) – B(I)*Z(I-1))/A(I)
Do for I = (N-1) to 1 Step –1	
	Z(I) = Z(I) – C(I) * Z(I+1)
RETURN	

符號說明：

A： 存放 α_i 或 a_i

B： 存放 β_i 或 b_i

C： 存放 γ_i 或 c_i

Z： 存放 Z_i 或 x_i

程式列印：

```
Sub TridiagonalMatrixEquation(Xpos, Ypos)
'       ****************************************
'         TRIDIAGONAL MATRIX EQN.
'       ****************************************
'    A,B,C  = COEFFICIENT VECTOR
'    Z              = CONSTANT VECTOR
'                     /RETURN : SOLUTION
'
Dim A(50), B(50), C(50), Z(50)
'
'      INPUT A, B, C & Z
'
AB$ = "                                "
```

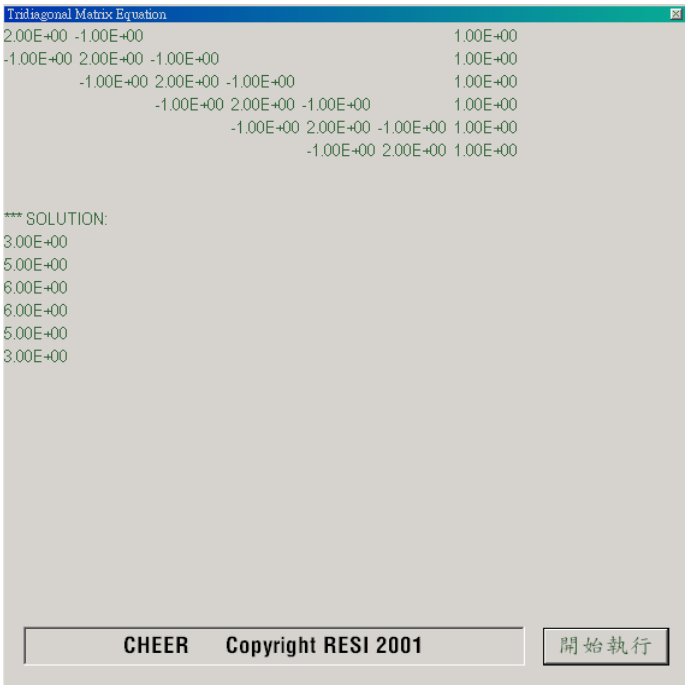
```
NEQN% = 6
For I = 1 To NEQN%
        A(I) = 2
        B(I) = -1
        C(I) = -1
        Z(I) = 1
Next I
B(1) = 0
C(NEQN%) = 0
'
'           ECHO
'
Cls
Print Format(A(1), "0.00E+00    ");
Print Format(C(1), "0.00E+00    ");
For I = 3 To NEQN%
        Print AB$;
Next I
Print Format(Z(1), "0.00E+00    ")
For I = 2 To NEQN% - 1
        If I > 2 Then
                For J = 3 To I
                Print AB$;
                Next J
        End If
        Print Format(B(I), "0.00E+00    ");
        Print Format(A(I), "0.00E+00    ");
        Print Format(C(I), "0.00E+00    ");
        If I <> NEQN% - 1 Then
                For J = (I + 1) To NEQN% - 1
                        Print AB$;
                Next J
        End If
        Print Format(Z(I), "0.00E+00    ")
Next I
For I = 1 To NEQN% - 2
        Print AB$;
Next I
Print Format(B(NEQN%), "0.00E+00    ");
Print Format(A(NEQN%), "0.00E+00    ");
Print Format(Z(NEQN%), "0.00E+00    ")
Print
Print
'
'         SOLVE & PRINT RESULT
'
Call Tridiagonal(NEQN%, A, B, C, Z)
Print "*** SOLUTION:"
```

```
For I = 1 To NEQN%
        Print Format(Z(I), "0.00E+00    ")
        If (Int(I / 10) = I / 10) Then Print
Next I
Print
End Sub


Sub Tridiagonal(NEQN%, A, B, C, Z)
'
'         SUBROUTINE JACOBI
'
' NEQN%            = No. of Equations
' A, B, C          = Tridiagonal Coefficients of Equations
' Z                = Constant Vector / Return ==> Solution Vector
'
C(1) = C(1) / A(1)
Z(1) = Z(1) / A(1)
'
'    -- LU DECOMPOSITION

For I = 2 To NEQN%
        A(I) = A(I) - B(I) * C(I - 1)
        C(I) = C(I) / A(I)
        Z(I) = (Z(I) - B(I) * Z(I - 1)) / A(I)
Next I
'
'    -- BACK SUBSTITUTION

For I = NEQN% - 1 To 1 Step -1
        Z(I) = Z(I) - C(I) * Z(I + 1)
Next I
'
'-- CHEER by Ron Hsin Chang, Copyright 2001
'
End Sub
```

副程式使用方法：

1. 副程式 **Sub Tridiagonal(NEQN%, A, B, C, Z)** 的使用方法：
   (1) 輸入方程式數目 (NEQN%)、三組對角線元素值 (A ， B ， C) 及常數陣列值 (Z)。
   (2) 呼叫副程式 Call Tridiagonal(NEQN%, A, B, C, Z)。
   (3) 計算結果會儲存在 Z 陣列，並傳回。

測試數據及結果：



結果討論：

三對角線矩陣方程式最常出現於利用有限差分法解微分方程的問題，故其程式規劃必須加以熟悉，其應用詳見第九章及第十一章。另外，在化工領域的蒸餾、吸收及反應器設計上的應用上，亦時常利用此技巧。

利用此程式解三對角線矩陣方程式的優點有二：(1) 較高斯消去法節省記憶空間；(2) 速度遠較高斯消去法快。利用上下分解法技巧，亦可提高他種帶狀對角線方程式的解題速度，詳見 [4，5]。

第七節　利用 Excel 解聯立方程式

Visual Basic

本章所介紹的各種線性聯立方程式解法，雖然簡單易用，但程式編寫仍有一定的複雜度。近年來，由於 Microsoft Excel 的普遍使用，工程人員亦可善用 Microsoft Excel 內建的反矩陣運算函數 Minverse(Array) 及矩陣乘法運算函數 Mmult (Array 1,

Array 2)，做爲求解線性聯立方程式或求反矩陣的工具。回顧本章第三節，我們可將線性聯立方程式以矩陣符號表示爲

$$\underline{\underline{A}}\,\underline{\underline{X}} = \underline{\underline{B}} \qquad \textbf{(4-7.1)}$$

其解答爲

$$\underline{\underline{X}} = \underline{\underline{A}}^{-1}\,\underline{\underline{B}} \qquad \textbf{(4-7.2)}$$

亦即，要求解聯立方程式 (4-7.2)，只要先求出反矩陣 $\underline{\underline{A}}^{-1}$，再利用矩陣乘法，即可求出解答。在 Excel 中，有兩個簡單易用的矩陣操作函數，(1) 反矩陣運算函數 *Minverse(Array)*，及 (2) 矩陣乘法運算函數 *Mmult(Array 1, Array 2)*；正好可以作這兩種運算。

以方程式 (4-7.2) 爲例，定義矩陣 *A* 以後，利用 *Minverse(A)* 即可得到反矩陣值 A^{-1}；然後，定義陣列 B，再利用 *Mmult*(A^{-1}, *B*) 即可得到解答 *X*。

例題 4-6 利用 Excel 解聯立方程式

利用 Excel 解下列聯立方程式

$$\begin{cases} -3x - y + 11z = 0 \\ 13x - 8y - 3z = 20 \\ -8x + 10y - z = -5 \end{cases}$$

解：利用 Excel 解上列聯立方程式，如下表所示：

	A	B	C	D	E	F	G	H
1	矩陣 *A*				矩陣 *B*			
2		–3	–1	11		0		
3		13	–8	–3		20		
4		–8	10	–1		–5		
5								
6	反矩陣 A^{-1}				解答 $X = A^{-1}B$			
7		0.06609	0.18957	0.15826		3		
8		0.06435	0.15826	0.23304		2		
9		0.11478	0.06609	0.06435		1		

程式編寫方法：

1. 首先在 B2..D4 輸入係數矩陣 A。

2. 在 F2..F4 輸入常數陣列 B。

3. 圈選 B7..D9，輸入=Minverse(B2..D4)。

4. 然後同時按 <Ctrl> <Shift> <Enter>，則 B7..D9 將出現矩陣 A 的反矩陣值。

5. 圈選 F7..F9，輸入=Mmult(B7..D9,F2..F4)。

6. 然後同時按 <Ctrl> <Shift> <Enter>，則 F7..F9 將出現答案陣列 X 值。

注意：在 Excel 應用上，需使用 <Ctrl> <Shift> <Enter> 代表函數輸入。

第八節　矩陣特徵值

Visual Basic

對於一個方矩陣 $\underline{\underline{A}}$，若存在一個非爲零的陣列 $\underline{u}$，使得該矩陣 $\underline{\underline{A}}$ 與陣列 $\underline{u}$ 的乘積等於陣列 $\underline{u}$ 與一特定常數 λ 的乘績，則稱該特定常數 λ 爲矩陣的特徵值 (Eigenvalue)，$\underline{u}$ 即稱爲矩陣的特徵陣列 (Eigenvector)。

以數學符號表示，可以寫成

$$\underline{\underline{A}}\underline{u} = \lambda \underline{u} \qquad \textbf{(4-8.1)}$$

或

$$(\underline{\underline{A}} - \lambda \underline{\underline{I}})\underline{u} = 0$$

$$\begin{bmatrix} a_{11}-\lambda & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22}-\lambda & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33}-\lambda & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn}-\lambda \end{bmatrix} \underline{u} = 0 \qquad \textbf{(4-8.2)}$$

利用方程式 (4-8.2) 可以建立矩陣的特徵方程式爲 $\phi(\lambda) = \det(\underline{\underline{A}} - \lambda \underline{\underline{I}}) = 0$。

例題 4-7　矩陣之特徵值

試求矩陣 $\underline{\underline{A}}$ 之特徵值。

$$\underline{\underline{A}} = \begin{bmatrix} 17 & -1 & -27 & -6 \\ 6 & -14 & -54 & -24 \\ 1 & 1 & -29 & -4 \\ -9 & -19 & 51 & 6 \end{bmatrix}$$

解： 利用方程式 (4-8.2) 展開，得到矩陣 A 的特徵方程式爲

$$\det(\underline{\underline{A}} - \lambda \underline{\underline{I}}) = \det \begin{bmatrix} 17-\lambda & -1 & -27 & -6 \\ 6 & -14-\lambda & -54 & -24 \\ 1 & 1 & -29-\lambda & -4 \\ -9 & -19 & 51 & 6-\lambda \end{bmatrix}$$

$$\lambda^4 + 20\lambda^3 - 700\lambda^2 - 8000\lambda + 120000 = 0$$

解此特徵方程式，得到矩陣的特徵值爲 −30 , −20 , 10 及 20。

魯逖髯瑟 L-U 分解法

方矩陣的特徵值在解微分方程式問題時，常需使用，是一種很重要的數學技巧。但是利用以上所介紹的方法解方矩陣的特徵值，當矩陣元素較多時，將變得極爲複雜，而不切實際。魯逖髯瑟 (Rutishauser, [6]) 提出的 L-U 分解法，是求解方矩陣特徵值一個有效率的方法。

首先假設方矩陣 $\underline{\underline{A}} = \underline{\underline{A}}_1$ ，同時假設可以將方矩陣 $\underline{\underline{A}}$ 改寫成上三角形矩陣 $\underline{\underline{R}}$ 及下三角形矩陣 $\underline{\underline{L}}$ 的乘積

$$\underline{\underline{A}}_k = \underline{\underline{L}}_k \underline{\underline{R}}_k \qquad \underline{\underline{A}}_{k+1} = \underline{\underline{R}}_k \underline{\underline{L}}_k \qquad \textbf{(4-8.3)}$$

其中 $\underline{\underline{L}}_k$ 爲對角線元素均爲 1，且在對角線以上的元素均爲零的下三角形方陣，$\underline{\underline{R}}_k$ 爲對角線以下的元素均爲零的上三角形方陣。

由於 $\underline{\underline{A}}_1 = \underline{\underline{L}}_1 \underline{\underline{R}}_1$ ，可以得到 $\underline{\underline{L}}_1 = \underline{\underline{A}}_1 \underline{\underline{R}}_1^{-1}$ ，及 $\underline{\underline{R}}_1 = \underline{\underline{L}}_1^{-1} \underline{\underline{A}}_1$ 。而又由於 $\underline{\underline{A}}_2 = \underline{\underline{R}}_1 \underline{\underline{L}}_1$ ，若將 $\underline{\underline{L}}_1$ 及 $\underline{\underline{R}}_1$ 的表示式分別代入 $\underline{\underline{A}}_2$ 中，可以得到

$$\underline{\underline{A}}_2 = \underline{\underline{L}}_1^{-1} \underline{\underline{A}}_1 \underline{\underline{L}}_1$$
$$\underline{\underline{A}}_2 = \underline{\underline{R}}_1 \underline{\underline{A}}_1 \underline{\underline{R}}_1^{-1}$$

連續重複執行此步驟，可以得到 $\underline{\underline{A}}_{k+1}$ 的表示式爲

$$\underline{\underline{A}}_{k+1} = \underline{\underline{L}}_k^{-1}\underline{\underline{L}}_{k-1}^{-1}\cdots\underline{\underline{L}}_2^{-1}\underline{\underline{L}}_1^{-1}\underline{\underline{A}}\underline{\underline{L}}_1\underline{\underline{L}}_2\cdots\underline{\underline{L}}_{k-1}\underline{\underline{L}}_k = \left(\underline{\underline{L}}_k^*\right)^{-1}\underline{\underline{A}}\left(\underline{\underline{L}}_k^*\right) \quad \textbf{(4-8.4)}$$

$$\underline{\underline{A}}_{k+1} = \underline{\underline{R}}_k\underline{\underline{R}}_{k-1}\cdots\underline{\underline{R}}_2\underline{\underline{R}}_1\underline{\underline{A}}\underline{\underline{R}}_1^{-1}\underline{\underline{R}}_2^{-1}\cdots\underline{\underline{R}}_{k-1}^{-1}\underline{\underline{R}}_k^{-1} = \left(\underline{\underline{R}}_k^*\right)\underline{\underline{A}}\left(\underline{\underline{R}}_k^*\right)^{-1} \quad \textbf{(4-8.5)}$$

$\underline{\underline{A}}_{k+1}$ 與方矩陣 $\underline{\underline{A}}$ 相似，且有相同的特徵值及特徵陣列。假設 $\underline{\underline{L}}_k^*$ 爲有限下三角形方陣，當 k 值趨近於無窮大時，$\underline{\underline{L}}_k^*$ 會趨近於定值。亦即

$$\underset{k\to\infty}{Lim}\,\underline{\underline{L}}_k^* = \underline{\underline{L}}_1\underline{\underline{L}}_2\cdots\underline{\underline{L}}_{k-1}\underline{\underline{L}}_k = L^* \quad \textbf{(4-8.6)}$$

k 值增加 1 時，$\underline{\underline{L}}_k^*$ 仍會相同，亦即 $\underset{k\to\infty}{Lim}\,\underline{\underline{L}}_k = \underline{\underline{I}}$。代回方程式 (4-8.3)，得到

$$\underset{k\to\infty}{Lim}\underline{\underline{A}}_k = \underset{k\to\infty}{Lim}\,\underline{\underline{R}}_k = \underline{\underline{R}} \quad \textbf{(4-8.7)}$$

將方程式 (4-8.7) 代入方程式 (4-8.4)，得到

$$\underline{\underline{A}} = \left(\underline{\underline{L}}^*\right)\underline{\underline{R}}\left(\underline{\underline{L}}^*\right)^{-1} \quad \textbf{(4-8.8)}$$

其中 $\underline{\underline{R}}$ 與 $\underline{\underline{A}}$ 有相同的特徵值，而且若 $\underline{\underline{A}}$ 的特徵值爲實數，則其特徵值即爲 $\underline{\underline{R}}$ 的對角線元素。

又由於由方程式 (4-8.3) 知道 $\underline{\underline{L}}_k\underline{\underline{R}}_k = \underline{\underline{R}}_{k-1}\underline{\underline{L}}_{k-1}$，因此

$$\begin{aligned}
\underline{\underline{L}}_k^*\underline{\underline{R}}_k^* &= (\underline{\underline{L}}_1\underline{\underline{L}}_2\cdots\underline{\underline{L}}_{k-2})\underline{\underline{L}}_{k-1}\underline{\underline{L}}_k\underline{\underline{R}}_k\underline{\underline{R}}_{k-1}(\underline{\underline{R}}_{k-2}\cdots\underline{\underline{R}}_2\underline{\underline{R}}_1)\\
&= (\underline{\underline{L}}_1\underline{\underline{L}}_2\cdots\underline{\underline{L}}_{k-2})\underline{\underline{L}}_{k-1}\underline{\underline{R}}_{k-1}\underline{\underline{L}}_{k-1}\underline{\underline{R}}_{k-1}(\underline{\underline{R}}_{k-2}\cdots\underline{\underline{R}}_2\underline{\underline{R}}_1)\\
&= (\underline{\underline{L}}_1\underline{\underline{L}}_2\cdots\underline{\underline{L}}_{k-2})\underline{\underline{R}}_{k-2}\underline{\underline{L}}_{k-2}\underline{\underline{R}}_{k-2}\underline{\underline{L}}_{k-2}(\underline{\underline{R}}_{k-2}\cdots\underline{\underline{R}}_2\underline{\underline{R}}_1)\\
&= \cdots\cdots\\
&= (\underline{\underline{L}}_1\underline{\underline{R}}_1)^k \equiv \underline{\underline{A}}^k
\end{aligned} \quad \textbf{(4-8.9)}$$

令 $\underline{\underline{B}} = \underline{\underline{A}}^k \equiv \underline{\underline{C}}\underline{\underline{D}}, \underline{\underline{B}} = (b_{ij})$；$\underline{\underline{C}}$ 爲對角線元素均爲 1，且在對角線以上的元素均爲零的下三角形方陣，$\underline{\underline{C}}=(c_{ij})$；$\underline{\underline{D}}$ 爲對角線以下的元素均爲零的上三角形方陣，$\underline{\underline{D}}=(d_{ij})$。則利用矩陣運算方式，可以知道 $b_{ij} = \sum_{k=1}^{N} c_{ik}d_{kj}$，又由 $\underline{\underline{C}}$ 及 $\underline{\underline{D}}$ 的特性知道，若 $i < j \le N$ 則 $c_{ij} = 0$；$c_{ii} = 1$；且若 $i > j$ 則 $d_{ij} = 0$。因此，整理後，可以得到

$$\begin{aligned}
b_{ij} &= \sum_{k=1}^{i-1} c_{ik}d_{kj} + d_{ij} \quad , \quad i \le j\\
b_{ij} &= \sum_{k=1}^{j-1} c_{ik}d_{kj} + c_{ij}d_{jj} \quad , \quad i > j
\end{aligned} \quad \textbf{(4-8.10)}$$

由於 $\underline{\underline{B}} = \underline{\underline{A}}^k$ 可以利用 A 重複乘積計算之，c_{ij} 及 d_{ij} 可以用下列方式計算之。

$$\begin{aligned} d_{ij} &= b_{ij} - \sum_{k=1}^{i-1} c_{ik} d_{kj} \quad , \quad i \le j \\ c_{ij} &= \frac{1}{d_{jj}} \left[b_{ij} - \sum_{k=1}^{j-1} c_{jk} d_{kj} \right] \quad , \quad i > j \end{aligned} \tag{4-8.11}$$

魯逖髻瑟的 L-U 分解法計算策略

1. 將原始矩陣 $\underline{\underline{A}}$ 放置在作業矩陣 $\underline{\underline{B}}$ 中。
2. 將矩陣 $\underline{\underline{A}}_k$ 分解成下三角形矩陣 $\underline{\underline{L}}_k$ 及上三角形矩陣 $\underline{\underline{R}}_k$ 乘積，$\underline{\underline{A}}_k = \underline{\underline{L}}_k \underline{\underline{R}}_k$。其中

$$\begin{aligned} R_{ij} &= A_{ij} - \sum_{k=1}^{i-1} L_{ik} R_{kj} \quad , \quad i \le j \\ L_{ij} &= \frac{1}{R_{jj}} \left[A_{ij} - \sum_{k=1}^{j-1} L_{jk} R_{kj} \right] \quad , \quad i > j \end{aligned}$$

3. 將第 2 步驟所得到的下三角形矩陣 $\underline{\underline{L}}_k$ 及上三角形矩陣 $\underline{\underline{R}}_k$，以相反次序再組合乘積，得到 $\underline{\underline{A}}_{k+1}$。

$$\left.\begin{aligned} A_{ij} &= \sum_{k=1}^{N} R_{ik} L_{kj} = R_{ii} L_{ij} + \sum_{k=i+1}^{N} R_{ik} L_{kj} \qquad j = 1,2,\cdots,i-1 \\ A_{ij} &= R_{ij} + \sum_{k=j+1}^{N} R_{ik} L_{kj} \qquad\qquad\qquad\quad j = i, i+1,\cdots,N \end{aligned}\right\} i = 1,2,\cdots,N$$

4. 由步驟 2 及步驟 3 所得到的下三角形矩陣 $\underline{\underline{L}}_k$，計算下三角形矩陣的乘積矩陣 $\underline{\underline{L}}_k^* \underline{\underline{L}}_{k+1} \rightarrow \underline{\underline{L}}_{k+1}^*$。
5. 重複步驟 2 至步驟 4，直到下三角形矩陣 $\underline{\underline{L}}_k$ 的元素和小於某一很小的界限值 Eps(2)，視爲計算已收斂。
6. 矩陣 $\underline{\underline{A}}$ 的特徵值 (Eigenvalue) 即爲上三角形矩陣 $\underline{\underline{R}}_k$ 的對角線元素。
7. 利用類似高斯消去法最後求解的方式，計算所得到轉化矩陣 $\underline{\underline{B}}$ 的特徵陣列 (Eigenvector)。

$$\left.\begin{aligned} X_{jj} &= 1 \\ X_{ij} &= \frac{A_{ij} + \sum_{k=i+1}^{j-1} A_{ik} X_{kj}}{A_{jj} - A_{ii}} \quad i = j-1, j-2, \cdots, 1 \end{aligned}\right\} j = 1,2,\cdots,N$$

8. 最後再計算原矩陣的特徵陣列。

$$\left.\begin{aligned} U_{ij} &= X_{ij} + \sum_{k=1}^{i-1} L_{ik} X_{kj} \quad j = i, i+1, \cdots, N \\ U_{ij} &= L_{ij} + \sum_{k=1}^{j-1} L_{ik} X_{kj} \quad j = 1,2,\cdots,N \end{aligned}\right\} i = 1,2,\cdots,N$$

例題 4-8　矩陣之特徵值

試利用魯逖髯瑟 (Rutishauser , [6]) 的 L-U 分解法求矩陣 $\underline{\underline{A}}$ 之特徵值。

$$\underline{\underline{A}} = \begin{bmatrix} 10 & 9 & 7 & 5 \\ 9 & 10 & 8 & 6 \\ 7 & 8 & 10 & 7 \\ 5 & 6 & 7 & 5 \end{bmatrix}$$

解： 計算程式邏輯詳見以上所說明之**魯逖髯瑟的 L-U 分解法計算策略**，或見程式內所附說明。

程式列印：

```
Sub EigenMatrix(Xpos, Ypos)
Dim A(20, 20), B(20, 20), U(20, 20) As Double
Dim Eps(4) As Double
Dim FileName As String
Dim EigenIndex As Integer

Cls
FileNo = FreeFile
FileName = InputBox("Enter File Name for Common Data", "", "EigenCommon.dat", Xpos, Ypos)

Open FileName For Input As FileNo
    Input #FileNo, ITmax
    Print "ITmax = "; ITmax
    For I = 1 To 4
        Input #FileNo, Eps(I)
```

```
            Print "EPS("; I; ") = ";
            Print Format(Eps(I), "0.0000E+00")
        Next I
        Input #FileNo, Freq
        Print "Sweep = "; Freq
        Input #FileNo, Sweep
        Print "Sweep = "; Sweep
Close #FileNo

FileNo = FreeFile
FileName = InputBox("Enter File Name for Matrix Data ", "", "Eigen02.dat", Xpos, Ypos)

Open FileName For Input As FileNo
        Input #FileNo, N
        Input #FileNo, EigenIndex
        Input #FileNo, Eps(1)
        Print " N = "; N
        Print " Index = "; EigenIndex
        Print " Absolute Error Criteria = "; Eps(1)
        Print
        Print "Starting matrix is:"
        For I = 1 To N
            For J = 1 To N
                Input #FileNo, A(I, J)
                Print Format(A(I, J), "        0.000E+00");
            Next J
            Print
        Next I
Close #FileNo

Index = EigenIndex
Call Rutishauser(N, A, B, U, Freq, ITmax, Eps, EigenIndex, Sweep, LogicTag1, LogicTag2, Iter,
Stripd)
Print "Tridiagonal Matrix ID = "; Stripd
MsgBox ("Ready to Proceed")

Cls
Print
Print " No. of Iteration = "; Iter
Print " LogicTag1 = "; LogicTag1
Print " LogicTag2 = "; LogicTag2
Print
Print " The transformed matrix is"
For I = 1 To N
        For J = 1 To N
            Print Format(B(I, J), "    0.0000E+00    ");
        Next J
        Print
```

```
Next I
Print
Print " Eigenvalues are:"
For I = 1 To N
    Print Format(B(I, I), "    0.0000E+00    ");
Next I
Print
If Index < 0 Then
    Print "Eigenvectors NOT required"
    Exit Sub
End If
If (LogicTag1) Then
    Print " Eigenvectors not computed because "
    Print " one or more Sub-diagonal elements too large"
    Exit Sub
ElseIf (LogicTag2) Then
    Print " Eigenvectors not computed because "
    Print " pair of Eigenvalues too close"
    Exit Sub
End If
MsgBox ("Ready to Proceed")

Cls
Print
If Index >= 0 Then
    Print " Eigenvectors are:"
    For I = 1 To N
        For J = 1 To N
            Print Format(U(I, J), "    0.000E+00    ");
        Next J
        Print
    Next I
    Print
End If
If Index > 0 Then
    Print "Eigenrows are :"
    For J = 1 To N
        For I = 1 To N
            Print Format(U(I, J), "    0.000E+00    ");
        Next I
        Print
    Next J
    Print
End If

End Sub
```

```
Sub Rutishauser(N, A, B, U, Freq, ITmax, Eps, EigenIndex, Sweep, LogicTag1, LogicTag2, Iter, Stripd)
'
'     EigenSystem Finds the Eigenvalues, Eigenvectors and Eigenrows
'     of an NxN Square Matrix A By Rutishauser's Lower Upper Decomposition
'
'     N                The Size of the Actual Matrix
'     A                Array containing the NXN starting matrix A
'     B                Array containing final transformed matrix.
'                      Eigenvalues are the diagonal element of B.
'
'     EigenIndex    is an Indicator
'     EigenIndex    = -1    Gives Eignevalues only.
'     EigenIndex    = 0     Gives Eigenvalues and Eigenvectors.
'     EigenIndex    = 1     Gives Eigenvalues, Eigenvectors, and Eigenrows.
'
'     EPS(1)          Tolerance used in convergence testing. Typical 1.0E-12
'     EPS(2)          Tolerance of differences between any two Eigenvalues. Typical 1.E-4
'     EPS(3)          Tolerance used in check SUBSUM < EPS(3)
                           for having Eigenvalues. Typical 1.E-6
'     EPS(4)          Tolerance for the sweeping procedure. Typical 1.E-2
'
'     FREQ             Number of LR steps elapsing between successive "Sweeps", if any.
'     ITER             Number of LR steps actually performed by the program.
'     ITMAX            Maximum number of LR steps to be performed.
'
'     STRIPD           ="TRUE" if A is tridiagonal.    ="False" if A is not tridiagonal.
'     SWEEP             ="TRUE" if sweeping procedure is to be applied.    ="False" if not applied.
'
'     U                N X N Matrix whose column contain the eigenvectors of    A.
'
Dim X(20, 20), V(20), Sum, SubSum, SumSq, ULen, U20 As Double
Dim Begin(20), Finish(20) As Integer

'
'     Check Error Criteria and EigenIndex
'
Call CheckEPS(Eps, Freq, ITmax)
Call MachineErrorCheck(U20, Eps(1))
Call TridiagonalCheck(N, A, Stripd)
If N = 0 Then Exit Sub
NM1 = N - 1

For I = 1 To N
      For J = 1 To N
            X(I, J) = 0
            B(I, J) = 0
      Next J
```

```
Next I
L = 0
LogicTag1 = False
LogicTag2 = False
'
'     Determine the vectors begin and finish
'
If (Stripd) Then
        Begin(1) = 1
        Begin(N) = NM1
        Finish(1) = 2
        Finish(N) = N
        If (N > 2) Then
                For J = 2 To NM1
                        Begin(J) = J - 1
                        Finish(J) = J + 1
                Next J
        End If
Else
        For J = 1 To N
                Begin(J) = 1
                Finish(J) = N
        Next J
End If

For I = 1 To N
        JLow = Begin(I)
        JHigh = Finish(I)
        For J = JLow To JHigh
                B(I, J) = A(I, J)
        Next J
Next I
'
'     Start LR transformation
'     Iteration until convergence satisfactory
'     or iterations exceed ITmax
'
Iter = 0
Do While Iter <= ITmax
        Iter = Iter + 1
        For J = 1 To N
                ILow = Begin(J)
                For I = ILow To J
                        Sum = 0
                        IM1 = I - 1
                        KLow = Begin(I)
                        If (KLow <= IM1) Then
                                For K = KLow To IM1
```

```
                        Sum = Sum + B(I, K) * B(K, J)
                    Next K
                End If
                B(I, J) = B(I, J) - Sum
            Next I
            JP1 = J + 1
            IHigh = Finish(J)
            If (JP1 > IHigh) Then
                Exit For
            Else
                For I = JP1 To IHigh
                    Sum = 0
                    KLow = Begin(I)
                    JM1 = J - 1
                    If (KLow <= JM1) Then
                        For K = KLow To JM1
                            Sum = Sum + B(I, K) * B(K, J)
                        Next K
                    End If
                    B(I, J) = (B(I, J) - Sum) / B(J, J)
                Next I
            End If
        Next J
'
'   The accumulated product of the successive lower
'   triangular decomposition matrices ia computed
'
        If (EigenIndex >= 0) Then
            For I = 2 To N
                IM1 = I - 1
                For J = 1 To IM1
                    X(I, J) = B(I, J) + X(I, J)
                    KLow = J + 1
                    If (KLow <= IM1) Then
                        For K = KLow To IM1
                            X(I, J) = X(I, J) + X(I, K) * B(K, J)
                        Next K
                    End If
                Next J
            Next I
        End If
'
'   Factors are combined in reverse order
'
        For I = 1 To N
            JLow = Begin(I)
            IM1 = I - 1
            If (JLow <= IM1) Then
```

```
                    For J = JLow To IM1
                        B(I, J) = B(I, I) * B(I, J)
                        IP1 = I + 1
                        KHigh = Finish(I)
                        If (IP1 <= KHigh) Then
                            For K = IP1 To KHigh
                                B(I, J) = B(I, J) + B(I, K) * B(K, J)
                            Next K
                        End If
                    Next J
                End If
                JHigh = Finish(I)
                For J = I To JHigh
                    JP1 = J + 1
                    KHigh = Finish(J)
                    If (JP1 <= KHigh) Then
                        For K = JP1 To KHigh
                            B(I, J) = B(I, J) + B(I, K) * B(K, J)
                        Next K
                    End If
                Next J
            Next I
            For I = 1 To N
                JLow = Begin(I)
                JHigh = Finish(I)
                For J = JLow To JHigh
                    If (Abs(B(I, J)) < 0.0000000001) Then B(I, J) = 0
                Next J
            Next I

            L = L + 1
        '
        '   The sum of the absolute values of the
        '   sub-diagonal elements is computed
        '
            SubSum = 0
            For I = 2 To N
                SubSum = SubSum + Abs(B(I, I - 1))
            Next I
        '
        '   Determine column vectors for sweeping procedure
        '
            If (L = Freq And SubSum < Eps(4) And Sweep) Then
                For J = 1 To NM1
        '
        '           Reject cases for which diag. elements too close
        '
                    LoopFlag = 0
```

```
                    For I = 1 To N
                        If (Abs(B(J, J) - B(I, I)) < Eps(2) And J <> I) Then
                            LoopFlag = 1
                            Exit For
                        End If
                    Next I
                    If (LoopFlag = 0) Then
                        JP1 = J + 1
                        For IT = JP1 To N
                            I = N + JP1 - IT
                            V(I) = B(I, J)
                            IP1 = I + 1
                            If (I <> N) Then
                                For K = IP1 To N
                                    V(I) = V(I) + B(I, K) * V(K)
                                Next K
                            End If
                            V(I) = V(I) / (B(J, J) - B(I, I))
                        Next IT
'
'                       Modify lower triangular product matrix
'
                        For IT = JP1 To N
                            I = N + JP1 - IT
                            X(I, J) = X(I, J) + V(I)
                            IM1 = I - 1
                            If (JP1 <= IM1) Then
                                For K = JP1 To IM1
                                    X(I, J) = X(I, J) + X(I, K) * V(K)
                                Next K
                            End If
                        Next IT
'
'                       Postmultiply B with sweeping matrix
'
                        For I = 1 To N
                            For K = JP1 To N
                                B(I, J) = B(I, J) + B(I, K) * V(K)
                            Next K
                        Next I
'
'                       Premultiply B with inverse of sweeping matrix
'
                        For I = JP1 To N
                            For K = 1 To N
                                B(I, K) = B(I, K) - V(I) * B(J, K)
                            Next K
                        Next I
```

```
                End If
            Next J

            If (Stripd) Then
                For J = 1 To N
                    Begin(J) = 1
                    Finish(J) = N
                Next J
            End If
        End If
'
'   Check for convergence
'
        If (L = Freq Or Iter = ITmax Or SubSum < Eps(1)) Then L = 0
        If (SubSum < Eps(1)) Then Exit Do
Loop
'
'   Check to see if Eigenvectors are required or if any two
'   Eigenvalues are closer together than EPS(2)
'
If (EigenIndex < 0) Then Exit Sub
If (SubSum > Eps(3)) Then
        LogicTag1 = True
        Exit Sub
End If
For I = 1 To NM1
        IP1 = I + 1
        For J = IP1 To N
            If (Abs(B(I, I) - B(J, J)) < Eps(2)) Then
                LogicTag2 = True
                Exit Sub
            End If
        Next J
Next I
'
'   Compute Eigenvectors of transformed matrix
'
For J = 1 To N
        X(J, J) = 1
        If (J > 1) Then
            JM1 = J - 1
            For IT = 1 To JM1
                I = J - IT
                Sum = B(I, J)
                IP1 = I + 1
                If (IP1 <= JM1) Then
                    For K = IP1 To JM1
                        Sum = Sum + B(I, K) * X(K, J)
```

```
                    Next K
                End If
                X(I, J) = Sum / (B(J, J) - B(I, I))
            Next IT
        End If
    Next J
    '
    '    Compute Eigenvectors of original matrix
    '
    For I = 1 To N
        IM1 = I - 1
        If (I > 1) Then
            For J = 1 To IM1
                U(I, J) = X(I, J)
                JM1 = J - 1
                If (J > 1) Then
                    For K = 1 To JM1
                        U(I, J) = U(I, J) + X(I, K) * X(K, J)
                    Next K
                End If
            Next J
        End If
        For J = I To N
            U(I, J) = X(I, J)
            If (I > 1) Then
                For K = 1 To IM1
                    U(I, J) = U(I, J) + X(I, K) * X(K, J)
                Next K
            End If
        Next J
    Next I
    '
    '    Normalize the Eigenvectors
    '
    For J = 1 To N
        SumSq = 0
        For I = 1 To N
            SumSq = SumSq + U(I, J) * U(I, J)
        Next I
        ULen = Sqr(SumSq)
        For I = 1 To N
            U(I, J) = U(I, J) / ULen
        Next I
    Next J
    End Sub


    Sub TridiagonalCheck(N, A, Stripd)
```

```
'
'     Check whether the matrix is a tridiagonal matrix
'
Stripd = True
NM2 = N - 2
For I = 1 To NM2
      For J = I + 2 To N
            If (A(I, J) <> 0) Then
                  Stripd = False
                  Exit Sub
            End If
      Next J
Next I
For I = 3 To N
      IM2 = I - 2
      For J = 1 To IM2
            If (A(I, J) <> 0) Then
                  Stripd = False
                  Exit Sub
            End If
      Next J
Next I
End Sub


Sub CheckEPS(Eps, Freq, ITmax)
If Eps(1) = 0 Then Eps(1) = 0.000000000001
If Eps(2) = 0 Then Eps(2) = 0.0001
If Eps(3) = 0 Then Eps(3) = 0.000001
If Eps(4) = 0 Then Eps(4) = 0.01
If Freq < 5 Then Freq = 5
If ITmax < 20 Then ITmax = 50
End Sub


Sub MachineErrorCheck(U20, AbsErr)
'
'     Calculate Machine Round Off Error
'
ReMin = 0.000000000001
EpsMachine = 1

Do
      EpsMachine = EpsMachine / 2
      EpsP1 = EpsMachine + 1
Loop While EpsP1 > 1
U20 = 20 * EpsMachine
Rer = 2 * EpsMachine + ReMin
```

```
'
'    Check for proper error tolerance
If AbsErr < Rer Then
        AbsErr = Rer
End If
End Sub
```

副程式使用方法說明：

副程式 Sub Rutishauser (N, A, B, U, Freq, ITmax, Eps, EigenIndex, Sweep, LogicTag1, LogicTag2, Iter, Stripd) 的使用方法

1. 用途：計算矩陣 A 的特徵值及特徵陣列。

2. 呼叫方法 Call Rutishauser(N, A, B, U, Freq, ITmax, Eps, EigenIndex, Sweep, LogicTag1, LogicTag2, Iter, Stripd)

N	矩陣 A 的維次
A	NXN 矩陣
B	轉換後的矩陣，其對角線元素爲矩陣 A 的特徵值
EigenIndex	運算指標
	EigenIndex = -1　只計算特徵值
	EigenIndex = 0　計算特徵值及特徵行陣列.
	EigenIndex = 1　計算特徵值及特徵行陣列及特徵列陣列
EPS(1)	收斂誤差限值，通常使用 1.0E-12
EPS(2)	兩特徵值間的誤差限值，通常使用 1.E-4
EPS(3)	檢驗是否獲得特徵值 SUBSUM < EPS(3)的檢驗限值， 通常使用 1.E-6
EPS(4)	LR 處理過程的整理限值，通常使用 1.E-2
FREQ	若要求作強制整理，兩次 "Sweeps" 間 LR 處理的次數
ITER	程式中實際執行的 LR 次數
ITMAX	執行 LR 處理的最高次數限制
STRIPD	若矩陣 A 爲三對角線矩陣，則爲 "TRUE" 若矩陣 A 不是三對角線矩陣，則爲 ="False"
SWEEP	若使用 sweep 程序，強制整理，則爲 "TRUE" 否則爲 "False"
U	含有矩陣 A 的特徵行陣列 eigenvectors 之 N X N 矩陣

3. 此副程式內部會呼叫以下三個副程式：

Sub TridiagonalCheck(N, A, Stripd)

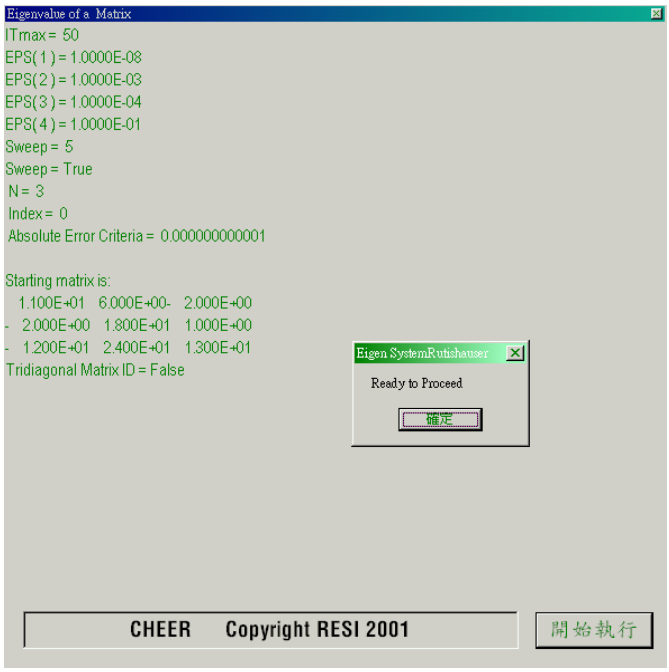
Sub MachineErrorCheck(U20, AbsErr)

Sub CheckEPS(Eps, Freq, ITmax)

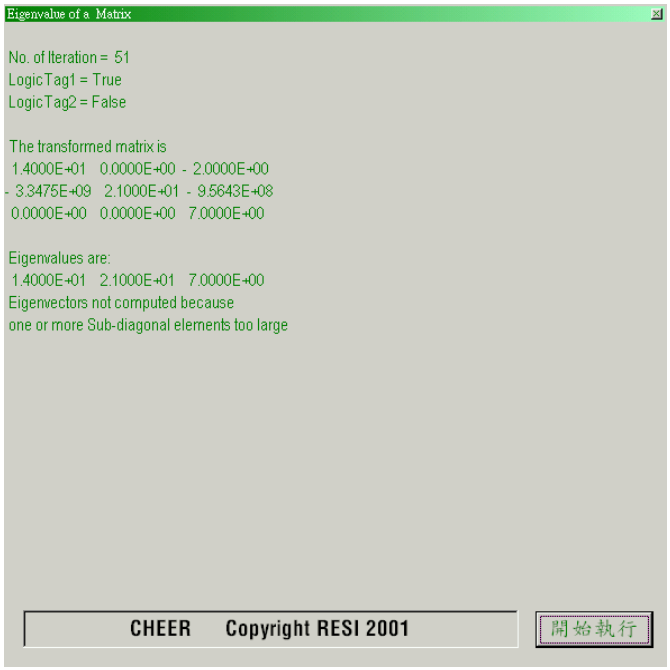
輸入數據與測試結果：

第一組

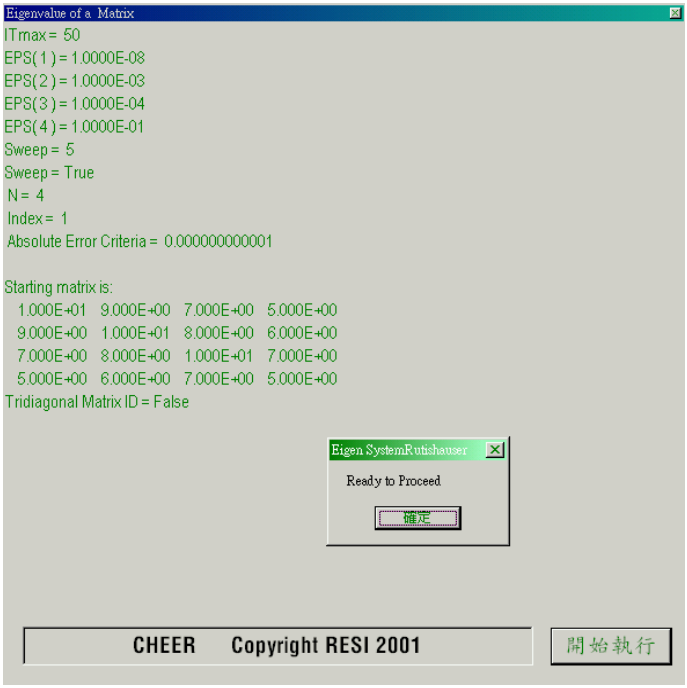
原始輸入數據

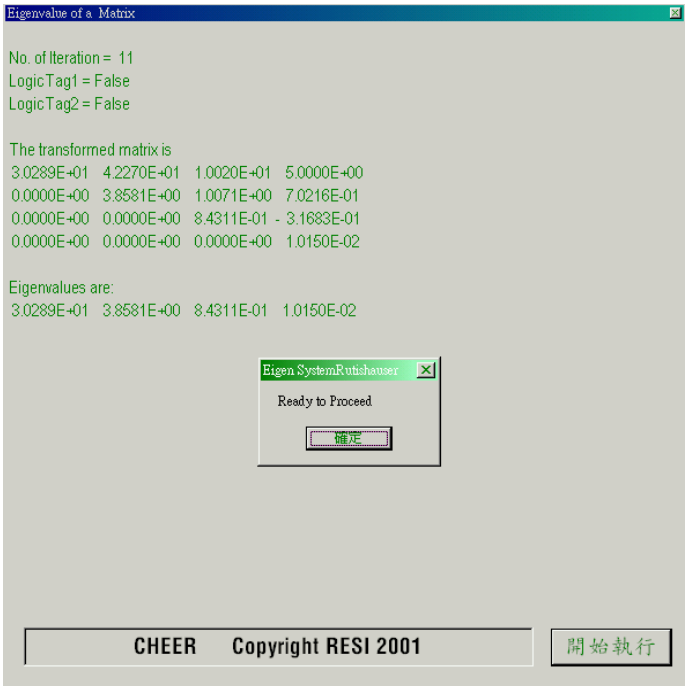


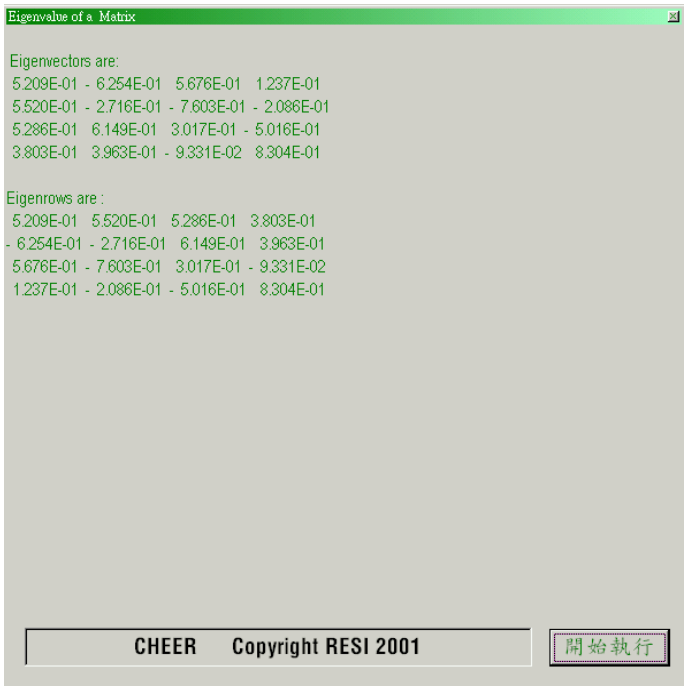
所得到的結果


Eigenvalue of a Matrix
No. of Iteration = 51
LogicTag1 = True
LogicTag2 = False
The transformed matrix is
1.4000E+01 0.0000E+00 - 2.0000E+00
- 3.3475E+09 2.1000E+01 - 9.5643E+08
0.0000E+00 0.0000E+00 7.0000E+00
Eigenvalues are:
1.4000E+01 2.1000E+01 7.0000E+00
Eigenvectors not computed because
one or more Sub-diagonal elements too large
CHEER Copyright RESI 2001
開始執行

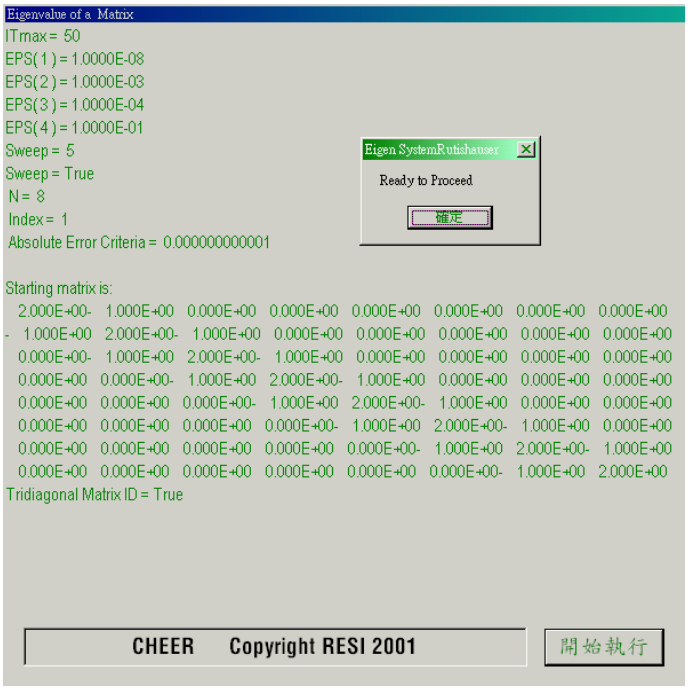
第二組

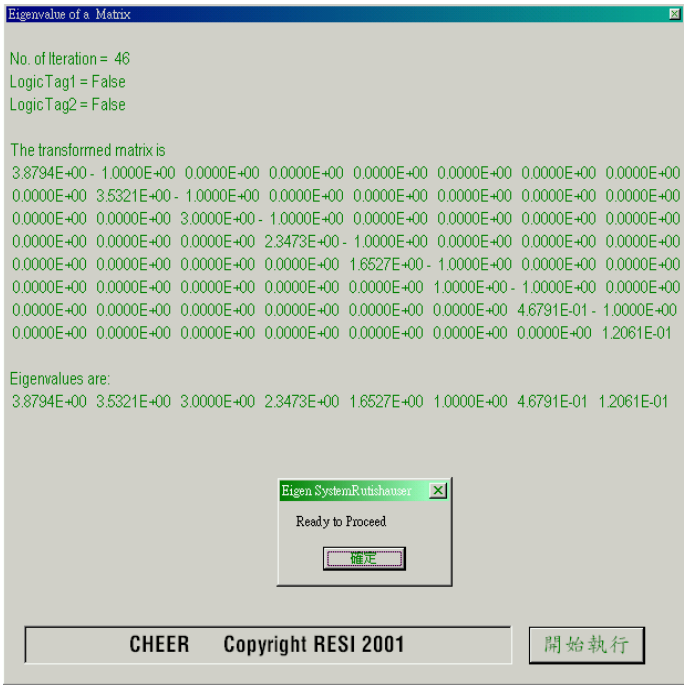

Eigenvalue of a Matrix
ITmax = 50
EPS(1) = 1.0000E-08
EPS(2) = 1.0000E-03
EPS(3) = 1.0000E-04
EPS(4) = 1.0000E-01
Sweep = 5
Sweep = True
N = 4
Index = 1
Absolute Error Criteria = 0.000000000001
Starting matrix is:
1.000E+01 9.000E+00 7.000E+00 5.000E+00
9.000E+00 1.000E+01 8.000E+00 6.000E+00
7.000E+00 8.000E+00 1.000E+01 7.000E+00
5.000E+00 6.000E+00 7.000E+00 5.000E+00
Tridiagonal Matrix ID = False
Eigen SystemRutishauser
Ready to Proceed
確定
CHEER Copyright RESI 2001
開始執行


Eigenvalue of a Matrix
No. of Iteration = 11
LogicTag1 = False
LogicTag2 = False
The transformed matrix is
3.0289E+01 4.2270E+01 1.0020E+01 5.0000E+00
0.0000E+00 3.8581E+00 1.0071E+00 7.0216E-01
0.0000E+00 0.0000E+00 8.4311E-01 - 3.1683E-01
0.0000E+00 0.0000E+00 0.0000E+00 1.0150E-02
Eigenvalues are:
3.0289E+01 3.8581E+00 8.4311E-01 1.0150E-02
Eigen SystemRutishauser
Ready to Proceed
確定
CHEER Copyright RESI 2001
開始執行


Eigenvalue of a Matrix
Eigenvectors are:
5.209E-01 - 6.254E-01 5.676E-01 1.237E-01
5.520E-01 - 2.716E-01 - 7.603E-01 - 2.086E-01
5.286E-01 6.149E-01 3.017E-01 - 5.016E-01
3.803E-01 3.963E-01 - 9.331E-02 8.304E-01
Eigenrows are :
5.209E-01 5.520E-01 5.286E-01 3.803E-01
- 6.254E-01 - 2.716E-01 6.149E-01 3.963E-01
5.676E-01 - 7.603E-01 3.017E-01 - 9.331E-02
1.237E-01 - 2.086E-01 - 5.016E-01 8.304E-01
CHEER Copyright RESI 2001
開始執行

第三組


Eigenvalue of a Matrix
ITmax= 50
EPS(1)= 1.0000E-08
EPS(2)= 1.0000E-03
EPS(3)= 1.0000E-04
EPS(4)= 1.0000E-01
Sweep= 5
Sweep= True
N= 8
Index= 1
Absolute Error Criteria= 0.000000000001
Eigen SystemRutishauser
Ready to Proceed
確定
Starting matrix is:
2.000E+00- 1.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00
- 1.000E+00 2.000E+00- 1.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00
0.000E+00- 1.000E+00 2.000E+00- 1.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00
0.000E+00 0.000E+00- 1.000E+00 2.000E+00- 1.000E+00 0.000E+00 0.000E+00 0.000E+00
0.000E+00 0.000E+00 0.000E+00- 1.000E+00 2.000E+00- 1.000E+00 0.000E+00 0.000E+00
0.000E+00 0.000E+00 0.000E+00 0.000E+00- 1.000E+00 2.000E+00- 1.000E+00 0.000E+00
0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00- 1.000E+00 2.000E+00- 1.000E+00
0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00- 1.000E+00 2.000E+00
Tridiagonal Matrix ID= True
CHEER Copyright RESI 2001
開始執行



結果討論：

1. 魯逖髯瑟 (Rutishauser, [6]) 的 L-U 分解法基本上只適用於特徵值爲實數的情況。
2. 將特徵值可能爲複數的情況加以考慮，且爲了簡化，將 A，B，及 X 整合成同一矩陣，並取消三項對角線矩陣的簡化處理程序。所得程式如下所示。
3. 矩陣特徵值應用，在第十一章偏微分方程式部分將再作進一步說明。

```
Sub EigenMatrix(Xpos, Ypos)
Dim A(150, 50), NComplex(50) As Double
Dim FileName As String
Dim EigenIndex As Integer

Cls
FileNo = FreeFile
FileName = InputBox("Enter File Name for Data Input", "FILE NAME", "Eigen01.dat", Xpos, Ypos)

Open FileName For Input As FileNo
      Input #FileNo, N
      Input #FileNo, EigenIndex
```

```
        Input #FileNo, AbsErr
        Print " N = "; N
        Print " Index = "; EigenIndex
        Print " Absolute Error Criteria = "; AbsErr
        Print
        Print "Starting matrix is:"
        For I = 1 To N
            For J = 1 To N
                Input #FileNo, A(I, J)
                Print Format(A(I, J), "        0.000E+00");
            Next J
            Print
        Next I
    Close #FileNo
    MsgBox ("Ready to Proceed")
    Index = EigenIndex
    Cls
    Call EigenSystem(N, EigenIndex, AbsErr, NComplex, A)
    Print
    Print " No. of Iteration = "; EigenIndex
    Print
    Print " Eigenvalues are:"
    For I = 1 To N
        If NComplex(I) = 0 Then
            Print Format(A(I, I), "    0.0000E+00    ");
        Else
            Print "Complex Eigenvalues"
        End If
    Next I
    Print
    Print

    N1 = 2 * N
    N2 = 3 * N

    If Index >= 0 Then
        Print " Eigenvectors are:"
        For I = N + 1 To N1
            For J = 1 To N
                Print Format(A(I, J), "    0.000E+00    ");
            Next J
            Print
        Next I
        Print
    End If

    If Index > 0 Then
        Print "Eigenrows are :"
```

```
        For I = N1 + 1 To N2
            For J = 1 To N
                Print Format(A(I, J), "    0.000E+00    ");
            Next J
            Print
        Next I
        Print
End If

End Sub


Sub EigenSystem(N, EigenIndex, AbsErr, NComplex, A)
'
'    EigenSystem Finds the Eigenvalues, Eigenvectors and Eigenrows
'    of an NxN Square Matrix A
'
'    N is the Size of the Actual Matrix
'
'    EigenIndex is an Indicator
'    EigenIndex    = -1   Gives Eignevalues only.
'                         The results are stored in the first column of A.
'                         Real and imaginary parts of eventual complex eigenvalues
'                         being in adjacent positions.
'    EigenIndex    = 0    Gives Eigenvalues and Eigenvectors.
'                         The Eigenvalues are on the main diagonal of A.
'                         Complex parts as a 2*2 block with the real parts (identical)
'                         on the diagonal, and the imaginary parts (identical, but in the
'                         opposite sign) as adjacent off-diagonal elements.
'                         The matrix of eigenvectors, Q, is stored in row N+1 - 2N
'    EigenIndex    = 1    Gives Eigenvalues, Eigenvectors, and Eigenrows.
'                         Eigenvalues and eigenvectors being stored as for EigenIndex =0.
'                         The eigenrows, Qinv, are stored in row 2N+1 - 3N
'
'    On exit, EigenIndex gives the number of iterations in the QR-step index.
'
'    AbsErr is the tolerance, and a suggested value is 1.E-8
'
'    NComplex is an indicator array for the Eigenvalues.
'    NComplex(I)=0      indicates that the I'th eigenvalue is real.
'    NComplex(I)=1      and NComplex(I+1)=2      indicate a complex pair of eigenvalues, with real
'                       part in A(I+1) and a complex part in A(I+2) for EigenIndex = -1.
'
'    The diagonalized matrix, D, the matrix of eigenvectors, Q, and
'    the matrix of eigenrows, Qinv, have the property, that
'          A = Q * D * Qinv
'
'    Check Error Criteria and EigenIndex
```

```
'
If (AbsErr <= 0) Then AbsErr = 0.000000000001

If (EigenIndex < -1) Then
        EigenIndex = -1
ElseIf EigenIndex > 1 Then
        EigenIndex = 1
End If
'
'       Define Matrix Size and Reset
'
If (EigenIndex <= 0) Then
        NS = N
Else
        NS = 2 * N
End If

NTotal = 2 * N
Index = EigenIndex

For I = 1 To N
        If (EigenIndex >= 0) Then
                For J = 1 To N
                        A(J + N, I) = 0
                Next J
                A(I + N, I) = 1
        End If
        NComplex(I) = 0
Next I
'
Call ArrangeMatrix(N, A, EigenIndex)
Call QRutish(N, AbsErr, A, EigenIndex)

If (Index > 0) Then
        For I = 1 To N
                For J = 1 To N
                        A(I + 2 * N, J) = A(J + N, I)
                Next J
        Next I
End If

Call LRAlgorithm(N, Index, AbsErr, NComplex, A)

If (Index >= 0) Then
        Call EigenVector(N, NTotal, NS, NComplex, A)
End If

End Sub
```

```
Sub ArrangeMatrix(N, A, EigenIndex)

Dim SumA2, SX, SK As Double

If (N <= 2) Then Exit Sub
NMinus2 = N - 2
NTotal = N + N
If (EigenIndex < 0) Then NTotal = N

For I = 1 To NMinus2
    IPlus1 = I + 1
    IPlus2 = I + 2
    SumA2 = 0

    For J = IPlus2 To N
        SumA2 = SumA2 + A(J, I) * A(J, I)
    Next J

    If (SumA2 <> 0) Then
        SumA2 = Sqr(SumA2 + A(IPlus1, I) * A(IPlus1, I))
        If (A(IPlus1, I) < 0) Then SumA2 = -SumA2
        A(IPlus1, I) = A(IPlus1, I) + SumA2
        SK = SumA2 * A(IPlus1, I)
        For J = IPlus1 To N
            SX = 0
            For K = IPlus1 To N
                SX = SX + A(K, I) * A(K, J)
            Next K
            If (SX <> 0) Then
                SX = SX / SK
                For K = IPlus1 To N
                    A(K, J) = A(K, J) - SX * A(K, I)
                Next K
            End If
        Next J

        For J = 1 To NTotal
            SX = 0
            For K = IPlus1 To N
                SX = SX + A(K, I) * A(J, K)
            Next K
            If (SX <> 0) Then
                SX = SX / SK
                For K = IPlus1 To N
                    A(J, K) = A(J, K) - SX * A(K, I)
                Next K
```

```
                End If
            Next J

            A(IPlus1, I) = -SumA2
            For J = IPlus2 To N
                A(J, I) = 0
            Next J
        End If
    Next I

    End Sub


    Sub QRutish(N, AbsErr, A, Index)

    Dim X, Y, Z, SX, SY As Double
    MT = 0
    If (Index < 0) Then MT = 1
    Index = 0
    NTotal = 2 * N
    SX = 0
    SY = 0
    I = N + 1
    Do
        I = I - 1

        If (I <= 2) Then Exit Sub
        Do
            IFLAG = 0
            M = 1
            Index = Index + 1
            II = I
            For JA = 2 To II
                J = I + 2 - JA
                K = J - 1
                X = Abs(A(J, K)) / (Abs(A(J, J)) + Abs(A(K, K)) + AbsErr)
                If (X <= AbsErr) Then
                    M = J
                    A(J, K) = 0
                    If (M <= (I - 2)) Then Exit For
                    If (M = (I - 1)) Then I = I - 1
                    IFLAG = 1
                    Exit For
                End If
            Next JA

            If IFLAG = 0 Then
                If (Index <> 1) Then
```

```
            SX = A(I, I) + A(I - 1, I - 1)
            SY = A(I, I) * A(I - 1, I - 1) - A(I, I - 1) * A(I - 1, I)
            If (SX = 0 And SY = 0) Then SX = A(I, I - 1)
        End If
        X = A(M, M) / A(M + 1, M) * (A(M, M) - SX) + A(M, M + 1) + SY / A(M + 1, M)
        Y = A(M, M) + A(M + 1, M + 1) - SX
        Z = A(M + 2, M + 1)
        IL = I - 1
        NL1 = N + MT * (I - N)
        NL2 = 1 + (M - 1) * MT
        NL3 = NTotal + (I - NTotal) * MT
        For J = M To IL
            SumSquare = Sqr(X * X + Y * Y + Z * Z)
            If (X < 0) Then SumSquare = -SumSquare
            X = X + SumSquare
            Y = Y / X
            Z = Z / X
            X = X / SumSquare
            For K = J To NL1
                SX = A(J, K) + Y * A(J + 1, K)
                If (J < IL) Then SX = SX + Z * A(J + 2, K)
                SX = SX * X
                A(J, K) = A(J, K) - SX
                If (J < IL) Then A(J + 2, K) = A(J + 2, K) - Z * SX
                A(J + 1, K) = A(J + 1, K) - Y * SX
            Next K
            For K = NL2 To NL3
                JFLAG = 0
                If ((K > N) Or ((K < J + 4) And (K <= I))) Then
                    SX = A(K, J) + Y * A(K, J + 1)
                    If (J < IL) Then SX = SX + Z * A(K, J + 2)
                    SX = SX * X
                    A(K, J) = A(K, J) - SX
                    A(K, J + 1) = A(K, J + 1) - Y * SX
                    If (J < IL) Then A(K, J + 2) = A(K, J + 2) - Z * SX
                End If
            Next K
            If (J > M) Then
                A(J, J - 1) = -SumSquare
                A(J + 1, J - 1) = 0
                If (J < I - 1) Then A(J + 2, J - 1) = 0
            End If
            X = A(J + 1, J)
            Y = 0
            Z = 0
            If (J < IL) Then Y = A(J + 2, J)
            If (J < I - 2) Then Z = A(J + 3, J)
        Next J
```

```
            Else
                Exit Do
            End If
            If (Index >= 10 * N) Then Exit Sub
        Loop While (Index < 10 * N)
    Loop While IFLAG = 1

    End Sub


    Sub LRAlgorithm(N, Index, AbsErr, NComplex, A)
    Dim Q, X, Y, Z, R As Double

    NS = 2 * N
    NTotal = NS
    If (Index <= 0) Then NS = N
    I = 0
    Do
        ILoop = 0
        I = I + 1

        If (I - N > 0) Then Exit Sub
        If (I < N) Then
            X = Abs(A(I + 1, I)) / (Abs(A(I, I)) + Abs(A(I + 1, I + 1)) + AbsErr)

            If (X > AbsErr) Then
                Y = A(I, I) + A(I + 1, I + 1)
                Z = A(I, I) * A(I + 1, I + 1) - A(I, I + 1) * A(I + 1, I)
                X = Y * Y - 4 * Z

                If (X >= 0) Then
                    R = (Abs(Y) + Sqr(X)) / 2
                    If (Y < 0) Then R = -R
                    If (Index < 0) Then
                        A(I, 1) = R
                        A(I + 1, 1) = Z / R
                        ILoop = 1
                    Else
                        Q = A(I, I) - R
                        S = A(I, I) - Z / R
                        If (Abs(S) > Abs(Q)) Then Q = S
                        Q = A(I + 1, I) / Q
                        C = 1 / Sqr(Q * Q + 1)
                        S = Q * C
                    End If
                Else
                    R = (A(I + 1, I + 1) - A(I, I)) / (A(I + 1, I) + A(I, I + 1))
                    If (Index < 0) Then
```

```
                A(I, 1) = Y / 2
                A(I + 1, 1) = Sqr(-X) / 2
                NComplex(I) = 1
                NComplex(I + 1) = 2
                ILoop = 1
            Else
                Q = 1 / Sqr(1 + R * R)
                C = Sqr((Q + 1) / 2)
                S = R * Q / C / 2
            End If
        End If

        If ILoop = 0 Then
            For J = 1 To NS
                K1 = I
                K2 = J
                If (J > N) Then
                    K1 = I + 2 * N
                    K2 = J - N
                End If
                Q = A(K1, K2)
                A(K1, K2) = Q * C + S * A(K1 + 1, K2)
                A(K1 + 1, K2) = A(K1 + 1, K2) * C - S * Q
            Next J

            For J = 1 To NTotal
                If ((J < I + 2) Or (J > N)) Then
                    Q = A(J, I)
                    A(J, I) = A(J, I) * C + A(J, I + 1) * S
                    A(J, I + 1) = C * A(J, I + 1) - Q * S
                End If
            Next J

            If (X < 0) Then
                NComplex(I) = 1
                NComplex(I + 1) = 2
                R = Sqr(Abs(A(I + 1, I) / A(I, I + 1)))
                For J = 1 To NTotal
                    A(J, I + 1) = A(J, I + 1) * R
                    If (J <= N) Then
                        A(I + 1, J) = A(I + 1, J) / R
                    ElseIf (J <= NS) Then
                        A(I + 2 * N + 1, J - N) = A(I + 2 * N + 1, J - N) / R
                    End If
                Next J
                ILoop = 1
            Else
                A(I + 1, I) = 0
```

```
                    End If
                End If
            Else
                A(I + 1, I) = 0
                ILoop = 0
            End If
        End If

        If (ILoop = 1) Then
            I = I + 1
        ElseIf (Index < 0) Then
            A(I, 1) = A(I, I)
        End If

    Loop
    End Sub


    Sub EigenVector(N, NVEC, NROW, NComplex, A)
    Dim B, D, DB, Det, R1, R2, RI1, RI2, X(2, 2) As Double
    IX = N
    NE = N + NROW
    Do While (IX > 0)
        C = A(IX, IX)
        D = 0
        I1 = 2
        If (NComplex(IX) > 0) Then
            I1 = 1
            D = A(IX - 1, IX)
        End If
        JX = IX + I1 - 3
        Do While (JX > 0)
            AC = A(JX, JX) - C
            J1 = 2
            B = 0

            If (NComplex(JX) > 0) Then
                J1 = 1
                B = A(JX - 1, JX)
            End If

            For J = 1 To 2
                For I = 1 To 2
                    X(I, J) = 0
                Next I
            Next J

            For I = I1 To 2
```

```
                For J = J1 To 2
                    X(J, I) = A(JX + J - 2, IX + I - 2)
                    A(JX + J - 2, IX + I - 2) = 0
                Next J
            Next I

            U = (X(1, 1) + X(2, 2)) / 2
            S = X(1, 1) - U
            V = (X(1, 2) - X(2, 1)) / 2
            T = X(1, 2) - V

            DB = B - D
            Det = AC * AC + DB * DB
            R1 = (U * AC + V * DB) / Det
            RI1 = (V * AC - U * DB) / Det

            DB = D + B
            Det = AC * AC + DB * DB
            R2 = (S * AC - T * DB) / Det
            RI2 = (T * AC + S * DB) / Det

            X(1, 1) = R1 + R2
            X(2, 2) = R1 - R2
            X(1, 2) = RI2 - RI1
            X(2, 1) = RI2 + RI1

            For K = 1 To NE
                If (K > N Or K < JX + J1 - 2) Then
                    For I = I1 To 2
                        For J = J1 To 2
                            If (K <= 2 * N) Then
                                A(K, IX + I - 2) = A(K, IX + I - 2) - X(I, J) * A(K, JX + J - 2)
                            Else
                                A(JX + J + 2 * N - 2, K - 2 * N) = A(JX + J + 2 * N - 2, K - 2 * N)
                                    + X(I, J) * A(IX + I + 2 * N - 2, K - 2 * N)
                            End If
                        Next J
                    Next I
                End If
            Next K

            JX = JX + J1 - 3
        Loop
        IX = IX + I1 - 3
    Loop
    End Sub
```

參考文獻

1. Stark, P. A. “Introduction to Numerical Methods”, New York: Macmillan, (1970).

2. Dorn, W. S., and D. McCracken, “Numerical Methods with FORTRAN IV Case Studies” John Wiley, (1972).

3. Fox, L., “An Introduction to Numerical Linear Algebra” Oxford Univ. Press, (1964).

4. Finlayson, B. A., “Nonlinear Analysis in Chemical Engineering”, (1980)

5. Villadsen, J., and M. L. Michelsen, “Solution of Differential Equation Models by Polynomial Approximation”, Prentice Hall. (1978)

6. Rutishauser, H, “Solution of Eigenvalue Problems with the L-R Transformation”, National Bureau of Standards, Appl. Math. Series, Vol. 49, pp. 47-81, (1958)

習　題

1. 解下列聯立方程式

$$\begin{aligned} x+y+z+w&=10\\ 2x+3y+z+5w&=31\\ -x+y5z+3w&=-2\\ 3x+y+7z2w&=18 \end{aligned}$$

2. 利用高斯佐丹法解以下三組聯立方程式，注意其中只有常數項不同，故可利用及修改本章所附程式一次輸入，求得三方程組的解。

$$\begin{cases} x+y+z=3\\ x-y-2z=-2\\ 2x+y-z=2 \end{cases}$$

$$\begin{cases} x+y+z=6\\ x-y-2z=-1\\ 2x+y-z=7 \end{cases}$$

$$\begin{cases} x+y+z=1 \\ x-y-2z=0 \\ 2x+y-z=0 \end{cases}$$

3. 試修改本章中所列的高斯賽得迭代程式，使程式可保留原輸入的矩陣。並修改程式，使該程式具有更改緩衝係數 λ，並重新執行的能力。

(提示：將原矩陣保留在 A1 (I , J) 及 Z1 (I) 中)

4. 試求下列矩陣的特徵值及特徵陣列。

(a) $\begin{bmatrix} 6 & 4 & 4 & 1 \\ 4 & 6 & 1 & 4 \\ 4 & 1 & 6 & 4 \\ 1 & 4 & 4 & 6 \end{bmatrix}$ (b) $\begin{bmatrix} 4 & -5 & 0 & 3 \\ 0 & 4 & -3 & -5 \\ 5 & -3 & 4 & 0 \\ 3 & 0 & 5 & 4 \end{bmatrix}$

(c) $\begin{bmatrix} -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 \end{bmatrix}$

5. 利用 25% 過量空氣使丙烷 (C_3H_8) 燃燒。燃燒反應化學式爲

$$C_3H_8 + 5O_2 \rightarrow 3CO_2 + 4H_2O$$

其中，過量空氣是指超過讓丙烷完全燃燒所需的空氣量。燃燒產物包括 CO_2，H_2O，N_2，O_2 及未完全燃燒的 C_3H_8，都直接由煙道排放。則每排放 100 莫耳煙道氣，需要用掉多少空氣？(假設丙烷的燃燒比率爲 0.98)

6. 圖 4.3 爲氨轉化器簡化流程圖，進料 *(F_1)* 中含有 *75.16% H_2 , 24.57% N_2* 及 *0.27% Ar*。進料 *(F_1)* 與回流氣體 *(F_2)* 混合後進入反應器；進入反應器的氣體中含有 *79.52% H_2*；而離開氨分離塔的氣體中，完全不含氨，且其中含 *80.01% H_2*。假設氨產品中完全不含溶解的氣體。

(a) 試建立所需方程式。

(b) 以每 100 莫耳進料爲計算基準，則回流量及排放量分別爲多少？

(c) 每通過轉化器一次，氫氣的轉化率爲多少？

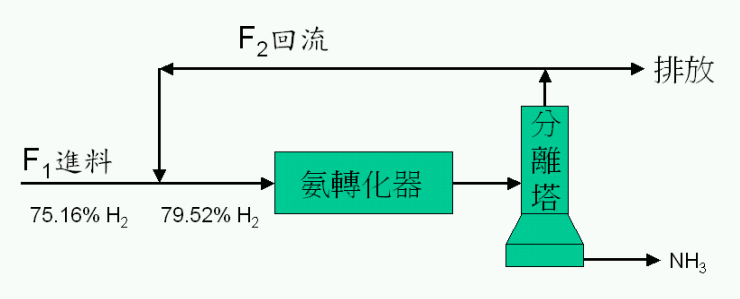


圖 4.3　氨轉化器簡化流程

7. 同分異構化反應器 (Isomerizer) 爲一種可將同分異構物重整的催化反應器。經過此反應器，反應物及產物的莫耳數不會改變。圖 4.4 所示流程圖，爲一用於製造對二甲苯之程序。流程圖中，各成分組成均以莫耳百分比表示，其中 A，B，C 及 D 分別爲

A　乙苯
B　鄰二甲苯
C　間二甲苯
D　對二甲苯

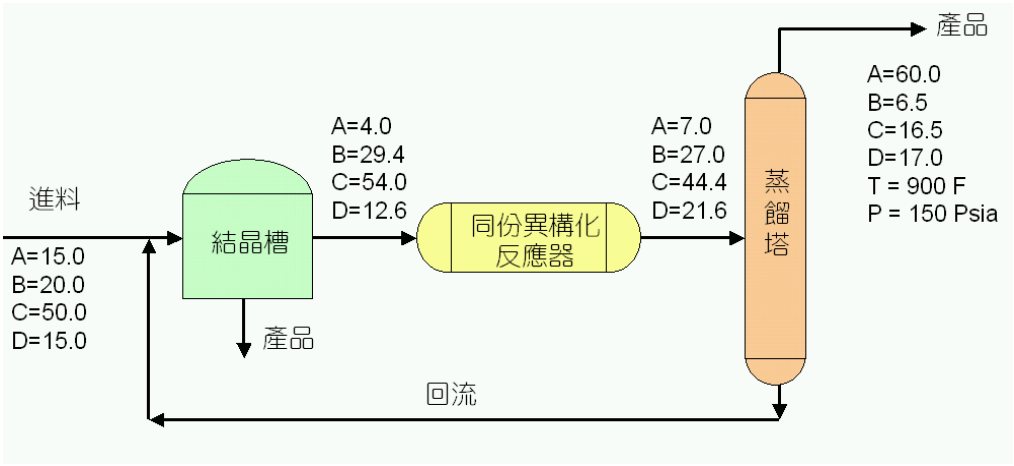


圖 4.4　同份異構化反應程序

進入蒸餾塔後，百分之八十的乙苯均可由塔頂蒸餾出。進入此程序的原料與結晶槽所得產品的莫耳數比爲 1.63。試依以下步驟進行此程序之質量結算：

(a) 對各操作單元及結點 (流束交點) 作質量平衡。

(b) 利用高斯消去法解所得方程組。

(c) 求回流比 (即蒸餾塔底部產物與其進料之莫耳數比)。

(c) 求結晶槽所得產品的組成。

8. 圖 4.5 爲丙烷催化脫氫生成丙烯的流程圖。其中回流組成及流量未知。反應器、吸收塔及蒸餾塔的部分數據如下：

每一百莫耳丙烷進料所得反應器產物組成

成分	莫耳%
氫	24.4
甲烷	3.2
乙烯	0.3
乙烷	5.3
丙烯	21.3
丙烷	44.5
總計	100.0

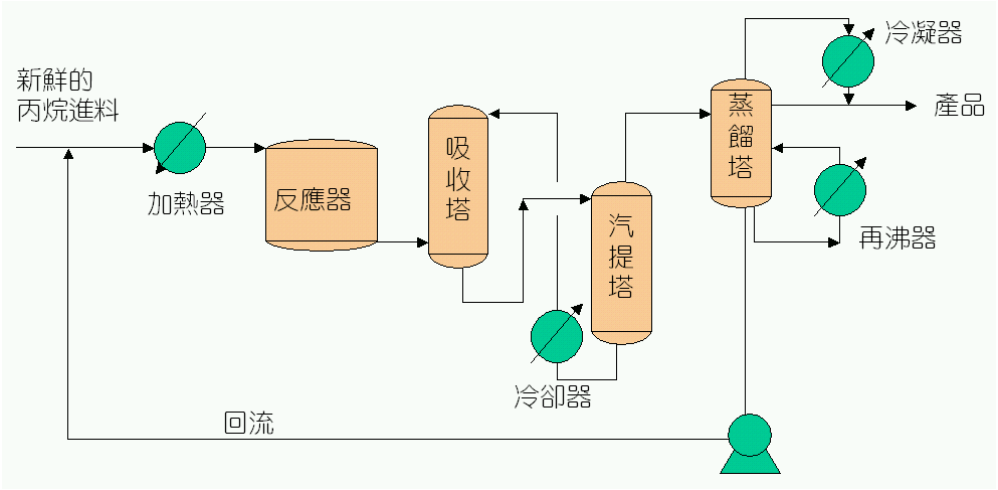


圖 4.5　丙烷催化脫氫反應程序

假設反應器的產品中所有的氫均來自丙烷進料，且反應器中的質量損失是由於催化劑上產生碳的緣故。同時假設反應器產物中的氫，在吸收塔中均不會被吸收。

(a) 試將流程圖簡化，畫成方塊圖 (如設計問題 D-IV)。

(b) 建立完整的質量平衡方程式。

(c) 利用高斯賽德迭代法解所得的方程組。

(d) 以進料 100 莫耳爲計算基準，求反應器進料、產物、吸收塔塔底流束，蒸餾塔塔底流束，及回流流量。

(e) 你能求出回流的組成嗎？

每一百莫耳進料所得吸收塔性能

成分	吸收塔進料莫耳數	未吸收部分比率
氫	25.4	1.00
甲烷	3.2	0.95
乙烯	0.3	0.70
乙烷	5.3	0.60
丙烯	21.3	0.01
丙烷	44.5	0.003
總計	100.0	

蒸餾塔性能

成　分	進料中由塔頂產物 回收的比率
甲　烷	1.00
乙　烷	1.00
乙　烯	1.00
丙　烯	0.96
丙　烷	0.002