第七章 實例測試與程式碼解說

撰寫 SMS(Sharp Matrix Solver)精銳矩陣計算器時，一面撰寫程式碼且同時一面測試，兩者同時進行，自我測試程式碼是一件重要的工作。

但有關矩陣微分方程式，總體而言就是空間維度(Space Dimension)上有 m 個自由度(Degree Of Fredom)，且狀態維度(State Dimension)上也是有 r 個自由度(即 r-Order 階度)，狀態維度是時間 t 的函數，詳細可參見維基百科 State-Space Representation。

有關數值驗證的實例並不多見，題材本身就是一個大問題，一般微分方程、線性代數、或是數值矩陣計算的書籍，其題材大部分是一個空間自由度(m=1)而狀態自由度(r>=2)的問題，但 m>=2 且 r>=2 的題材較少，尤其非線性的題才(即系統矩陣 A = A(t))更少，且一般題材都是無需特別解，但如果有外力時(f = f(t))上述一般解(General Solution)尚需包含齊次解(Homogeneous Solution)和特別解(Particular Solution)，更增加問題的複雜性。

目前找到四個較具代表性的測試題才 :
比對測試的結果與課本提供的數據，完全相同，也表示類別庫程式碼的可靠性，而且四個測試的程式碼都很類似，主要輸入已知的初始條件，或是邊界條件，就可得求得結果，讀者甚至可直接使用以下的四個題材作測試驗證。

第 1 個測試 :
J. L. Humar，"Dynamics of Structures" 第 502-504 頁。
第 2 個測試 :
Ray W. Clough & Joseph Penzien，"Dynamics of Structures" 第 202-203 頁。
第 3 個測試 :
William E. Boyce / Richard C. DiPrima, "Elementary Differential Equations and Boundary Value Problems 10th ED." 第 413 頁-416 頁。
第 4 個測試 :
Dennis G. Zill，"Differential Equations with Boundary-Value Problems 9th ED." 第 322 頁。

7.1　第 1 個測試

矩陣*常微分方程式*(ODE) : M * y(t)'' + C * y(t)' + K * y(t) = 0

M、K、C 矩陣和初始值已知，求解 y(t)''、y(t)'、y(t)的*數值*響應值。

$$M = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \quad K = \begin{pmatrix} 3 & -1 \\ -1 & 1 \end{pmatrix} \quad C = \begin{pmatrix} 0.4 & -0.5 \\ -0.05 & 0.2 \end{pmatrix}$$

$$\begin{pmatrix} \dot{y}(0) \\ y(0) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \mid \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

“|”是兩個向量(矩陣)的垂直合併的運算子。故上述是 4X1 的矩陣。

以下是課本所示的響應數學式 :

$$\dot{y}_0(t) = e^{-0.08334t}\{-0.00558 * \cos(0.70221t) - 0.71164 * \sin(0.70221t)\} + e^{-0.11677t}\{0.00559 * \cos(1.40933t) - 0.000066 * \sin(1.40933t)\}$$

$$\dot{y}_1(t) = e^{-0.08334t}\{-0.00558 * \cos(0.70221t) - 1.42474 * \sin(0.70221t)\} + e^{-0.11677t}\{0.00559 * \cos(1.40933t) + 0.000193 * \sin(1.40933t)\}$$

$$y_0(t) = e^{-0.08334t}\{-1.00028 * \cos(0.70221t) + 0.11076 * \sin(0.70221t)\} + e^{-0.11677t}\{-0.00028 * \cos(1.40933t) + 0.00395 * \sin(1.40933t)\}$$

$$y_1(t) = e^{-0.08334t}\{1.99983 * \cos(0.70221t) + 0.24528 * \sin(0.70221t)\} + e^{-0.11677t}\{0.00019 * \cos(1.40933t) - 0.00395 * \sin(1.40933t)\}$$

$$\ddot{y}_0(t) = N / A$$
$$\ddot{y}_1(t) = N / A$$

由上式 ODE(m=2, r=2, t)，矩陣微分方程式和初始條件，如下所示 :

$$M * \ddot{y} + C * \dot{y} + K * y = 0$$

$$\begin{pmatrix} \dot{y}(0) \\ y(0) \end{pmatrix} = \left(\dot{y}(0) \mid y(0)\right) = (0, 0, 1, 2)t$$

M, C, 和 K 是 mxm(即 2x2)的實數矩陣，由上述解析數學式知道需要繁雜的計算，才可導出複雜的空間和狀態***響應數學式***，但若是 m>2 或是 r>2，幾乎不容易由手算求得此複雜的響應數學式，但使用 SMS 應用程式，很容易求得***隨時間變化的數值(Numerical Number)***。使用解析法求解的響應數學式與 SMS 矩陣求解器應用程式，經檢測驗證兩者計算的結果，完全相同，故使用計算機程式求解是方便有效的方法，本演算法是百分百精準的數值求解法--由系統矩陣 A 求得，複數的特徵值和特徵向量矩陣，進而求得齊次解，是獨特的求解法，相關議題似乎尚未有人提及且程式碼實作。

7.2 C#程式碼碼與解說

引用 SMS 類別庫(using Matrix_0)，ConsoleApp6K 程式碼和相關的說明如下 :
(本控制台應用程式使用 Microsoft DotNet Framework 4.X)

```
1    using System;
2    using Matrix_0;
3
4    namespace ConsoleApp6K
5    {
6       internal class Program
7      {
8          static void Main(string[] args)
9          {
10
11   double[,] y0Start = { { 0 }, { 0 }, { 1 }, { 2 } };
12
13   double[,] M = { { 2, 0 }, { 0, 1 } };
14   double[,] K = { { 3, -1 }, { -1, 1 } };
15   double[,] C = { { 0.4, -0.05 }, { -0.05, 0.2 } };
16   MKCMatrix MKC = new MKCMatrix(M, K, C);
17   ReMatrix A = new ReMatrix(MKC.Matrix);
18
19   EIG eig = new EIG(A);
20   CxMatrix D = eig.CxMatrixD;
```

```
21    CxMatrix V = eig.CxVector;
22    CxMatrix Q = eig.CxMatrixQ;
23
24    CxHexp Hexp = new CxHexp(D, Q, 0);
25    CxMatrix MatTemp = Hexp.GetCxMatrix;
26    CxMatrix d = ~MatTemp * y0Start;
27
28    // 列印空間相依狀態參數。
29    Console.Write("\n***{0,12}空{0,7}間{0,7}狀{0,7}態{0,7}參{0,7}
        數{0,12}***\n", "");
30    Console.Write("\n***{0,5}系統特徵值V{0,5}***\n{1}\n", "", new
      PR(V));
31    Console.Write("\n***{0,5}系統特徵向量Q{0,5}***\n{1}\n", "",
        new PR(Q));
32    Console.Write("\n***{0,5}係數向量d{0,5}***\n{1}\n", "", new
        PR(d));
33
34    double step = 0.5;
35    int iRow = (int)(50 / step + 1);
36    int iCol = M.GetLength(1) + 1;
37    ReMatrix Disp = new ReMatrix(iRow, iCol);
38    ReMatrix Vel = new ReMatrix(iRow, iCol);
39    ReMatrix Acc = new ReMatrix(iRow, iCol);
40
41    for (int i = 0; i != iRow; i++)
42    {
43        double t = step * i;
44
45        Hexp = new CxHexp(D, Q, t);
46        MatTemp = Hexp.GetCxMatrix;
47        CxMatrix yh_Cx = MatTemp * d;
48        ReMatrix yh_Re = (ReMatrix)yh_Cx;
49
50        Vel.Matrix[i, 0] = t;
51        Vel.Matrix[i, 1] = yh_Re.Matrix[0, 0];
52        Vel.Matrix[i, 2] = yh_Re.Matrix[1, 0];
53        Disp.Matrix[i, 0] = t;
54        Disp.Matrix[i, 1] = yh_Re.Matrix[2, 0];
```

```
55          Disp.Matrix[i, 2] = yh_Re.Matrix[3, 0];
56
57          CxMatrix yhDot_Cx = A * yh_Cx;
58          ReMatrix yhDot_Re = (ReMatrix)yhDot_Cx;
59          Acc.Matrix[i, 0] = t;
60          Acc.Matrix[i, 1] = yhDot_Re.Matrix[0, 0];
61          Acc.Matrix[i, 2] = yhDot_Re.Matrix[1, 0];
62
63          }
64
65      // 列印節點的變位，速度，和加速。
66      Console.Write("\n***{0,12}空{0,7}間{0,7}狀{0,7}態{0,7}響{0,7}
          應{0,12}***\n", "");
67      Console.Write("\n{0,5}***位移反應量***{0,5}\n{0,8}時間(秒)" +
68        "{0,8}第0點位移{0,8}第1點位移\n\n{1}", "", new PR(Disp));
69      Console.Write("\n{0,5}***速度反應量***{0,5}\n{0,8}時間(秒)" +
70        "{0,8}第0點速度{0,8}第1點速度\n\n{1}", "", new PR(Vel));
71      Console.Write("\n***{0,5}加速度反應量{0,5}***\n{0,8}時間(秒)"
          + 72 "{0,8}第0點加速度{0,7}第1點加速度\n\n{1}", "", new
          PR(Acc));
73
74              }
75          }
76      }
```

程式碼說明 :

L2 : 加入名稱空間(namespace)，檔案 Matrix_0.dll 是動態連接程式庫。

L11 : t = 0 的初始條件(本例是 t=0，但 t 並不一定是 0，可以是任何值)。

L17 : 建構系統矩陣 A。

L20 - L22 : 特徵值 V 和特徵向量 Q (動態系統一般是複數值)。

L24 - L26 : 由 t = 0，求得 係數向量 d。”~”是矩陣的逆矩陣的運算子。

L29 - L32 : 空間相依狀態參數，即特徵值 V，特徵向量 Q，和系統向量 d

L34 - L36 : 頭尾時間總個數 iRow，響應總變數 iCol

L37 - L39 : 建構三個矩陣 Disp，Vel，和 Acc，其尺寸為 iRow 和 iCol

L41 - L63 : 在時間間隔內，將響應的資料放入 Disp、Vel、和 Acc 的矩陣內。

L66 - L76 : 列印空間和狀態響應資料，即變位(Disp)、速度(Vel)、和加速度(Acc)

相關的 PDF 檔案包含，程式碼文件檔、執行後的文件檔案、空間和狀態參數，即特徵值、特徵向量和係數向量，所謂狀態是某個特定時間軸上的空間和狀態響應值，即變位、速度、和加速度。

使用 SMS 類別庫，ConsoleApp6L 程式碼和相關的說明如下 ：

```
using System;
using Matrix_0;

namespace ConsoleApp6L
{
    internal class Program
    {
        static void Main(string[] args)
        {

double[,] y0Start = { { 0 }, { 0 }, { 1 }, { 2 } };

double[,] M = { { 2, 0 }, { 0, 1 } };
double[,] K = { { 3, -1 }, { -1, 1 } };
double[,] C = { { 0.4, -0.05 }, { -0.05, 0.2 } };
MKCMatrix MKC = new MKCMatrix(M, K, C);
ReMatrix A = new ReMatrix(MKC.Matrix);

EIG eig = new EIG(A);
CxMatrix D = eig.CxMatrixD;
CxMatrix V = eig.CxVector;
CxMatrix Q = eig.CxMatrixQ;

CxHexp Hexp = new CxHexp(D, Q, 0);
CxMatrix MatTemp = Hexp.GetCxMatrix;
CxMatrix d = ~MatTemp * y0Start;

double step = 0.5;
int iRow = (int)(50 / step + 1);
int iCol = M.GetLength(1) + 1;
```

```
31    ReMatrix Disp = new ReMatrix(iRow, iCol);
32    ReMatrix Vel = new ReMatrix(iRow, iCol);
33    ReMatrix Acc = new ReMatrix(iRow, iCol);
34
35    for (int i = 0; i != iRow; i++)
36    {
37        double t = step * i;
38
39        Hexp = new CxHexp(D, Q, t);
40        MatTemp = Hexp.GetCxMatrix;
41        CxMatrix yh_Cx = MatTemp * d;
42        ReMatrix yh_Re = (ReMatrix)yh_Cx;
43
44        Vel.Matrix[i, 0] = t;
45        Vel.Matrix[i, 1] = yh_Re.Matrix[0, 0];
46        Vel.Matrix[i, 2] = yh_Re.Matrix[1, 0];
47        Disp.Matrix[i, 0] = t;
48        Disp.Matrix[i, 1] = yh_Re.Matrix[2, 0];
49        Disp.Matrix[i, 2] = yh_Re.Matrix[3, 0];
50
51        CxMatrix yhDot_Cx = A * yh_Cx;
52        ReMatrix yhDot_Re = (ReMatrix)yhDot_Cx;
53        Acc.Matrix[i, 0] = t;
54        Acc.Matrix[i, 1] = yhDot_Re.Matrix[0, 0];
55        Acc.Matrix[i, 2] = yhDot_Re.Matrix[1, 0];
56
57        }
58
59        // 時間 :
60        Console.Write("\n 時間 : \n\n");
61        for( int i = 0; i != iRow; i++)
62        {
63            Console.Write("{0,10:F2}", Disp.Matrix[i, 0]);
64        }
65        Console.Write("\n\n");
66
67        // 位移
68        Console.Write("\n 位移 :\n");
```

```
69          Console.Write("\n 第 0 點 \n");
70          for(int i = 0; i != iRow; i++)
71          {
72              Console.Write("{0,10:F4}", Disp.Matrix[i, 1]);
73          }
74          Console.Write("\n 第 1 點 \n");
75          for(int i = 0; i != iRow; i++)
76          {
77              Console.Write("{0,10:F4}", Disp.Matrix[i, 2]);
78          }
79          Console.Write("\n\n");
80
81          // 速度
82          Console.Write("\n 速度 :\n");
83          Console.Write("\n 第 0 點 \n");
84          for (int i = 0; i != iRow; i++)
85          {
86              Console.Write("{0,10:F4}", Vel.Matrix[i, 1]);
87          }
88          Console.Write("\n 第 1 點 \n");
89          for (int i = 0; i != iRow; i++)
90          {
91              Console.Write("{0,10:F4}", Vel.Matrix[i, 2]);
92          }
93          Console.Write("\n\n");
94
95          // 加速度
96          Console.Write("\n 加速度 :\n");
97          Console.Write("\n 第 0 點 \n");
98          for (int i = 0; i != iRow; i++)
99          {
100             Console.Write("{0,10:F4}", Acc.Matrix[i, 1]);
101         }
102         Console.Write("\n 第 1 點 \n");
103         for (int i = 0; i != iRow; i++)
104         {
105             Console.Write("{0,10:F4}", Acc.Matrix[i, 2]);
106         }
```

```
107         Console.Write("\n\n\n");
108
109             }
110         }
111     }
```

ConsoleApp6L 的程式碼，是依據時間軸、變位、速度、和加速度分別輸出，這種輸出方式，利用 Python 的 Matplotlib 套件，方便繪製視覺化的圖表。讀者可利用提供的 TXT 資料數據自行測試。

使用 SMS 類別庫，ConsoleApp6M 程式碼和相關的說明如下 ：

```
1   using System;
2   using Matrix_0;
3
4   namespace ConsoleApp6M
5   {
6      internal class Program
7      {
8          static void Main(string[] args)
9          {
10
11     // 已知陣列 M, K, and C
12     double[,] M = { { 2, 0 }, { 0, 1 } };
13     double[,] K = { { 3, -1 }, { -1, 1 } };
14     double[,] C = { { 0.4, -0.05 }, { -0.05, 0.2 } };
15
16     // 建構系統矩陣A :
17     ReMatrix Mi = ~(ReMatrix)M;
18     Iden iden = new Iden(2);
19     ReMatrix I = iden.Matrix;
20     Zero zero = new Zero(2);
21     ReMatrix O = zero.Matrix;
22     ReMatrix A = ((-1 * Mi * C) & (-1 * Mi * K)) | (I & O);
23
```

```
24      EIG eig = new EIG(A);
25      CxMatrix D = eig.CxMatrixD;
26      CxMatrix V = eig.CxVector;
27      CxMatrix Q = eig.CxMatrixQ;
28
29      // 邊界值 @t = 4.5
30      double[,] y0 = { { -0.68877 }, { -1.37729 } };
31      // 邊界值 @t = 16.5
32      double[,] y1 = { { 0.11710 }, { 0.23071 } };
33      ReMatrix BVal = (ReMatrix)y0 | y1;
34
35      // 建構係數向量
36      CxHexp Hexp = new CxHexp(D, Q, 4.5);
37      CxMatrix MatTemp = Hexp.GetCxMatrix;
38      RowSlice rowSlice = new RowSlice(MatTemp, 2, 1);
39      CxMatrix Mat1 = rowSlice.GetCxMatrix;
40      Hexp = new CxHexp(D, Q, 16.5);
41      MatTemp = Hexp.GetCxMatrix;
42      rowSlice = new RowSlice(MatTemp, 2, 1);
43      CxMatrix Mat2 = rowSlice.GetCxMatrix;
44      CxMatrix Mat = Mat1 | Mat2;
45
46      CxMatrix d = ~ Mat * BVal;
47
48      // 列印空間相依狀態參數。
49      Console.Write("\n***{0,10}空{0,5}間{0,5}狀{0,5}態{0,5}參{0,5}
          數{0,10}***\n", "");
50      Console.Write("\n***{0,5}特徵值V{0,5}***\n{1}\n", "", new
          PR(V));
51      Console.Write("\n***{0,5}特徵向量矩陣Q{0,5}***\n{1}\n", "",
         new  PR(Q));
52      Console.Write("\n***{0,5}係數向量d{0,5}***\n{1}\n", "", new
          PR(d));
53
54          }
55      }
56  }
```

ConsoleApp6M 的程式碼，是利用邊界條件(BVC，Boundary Value Conditions)求得係數向量 d，與 ConsoleApp6K 是利用初始條件(IBC，Initial Value Conditions)求得係數向量，雖然程式碼或許較複雜。但求得的係數向量 d 完全相同。作者使用 ConsoleApp6K 的輸出結果，在 t = 4.5 秒時，第 0 點和第 1 點的變位 y，分別是 [-0.68877，-1.37729]，在 t = 16.5 秒時，則為 [0.11710, 0.23071]。 換言之相同的系統條件，得到相同的空間相依狀態參數(Spatial Dependency State)，特徵值(V)、特徵向量(Q)、和係數向量(d)，也就是有相同的空間相依狀態響應(Response)。作者有提供的 TXT 文件檔數據，請自行測試。

7.3 第 2 測試至第 4 測試部分

第 2 測試至第 4 測試的程式碼，幾乎與第 1 測試的程式碼完全相同，讀者可由提供的已知數據條件和程式碼，請自行測試，其結果與提供的數據是否相符。

至於資料的視覺化(輸出結果)，讀者可以將結果在 Excel 上繪製圖表。或是在 Visual Studio IDE 上執行 Python 專案，繪製圖表，也可以在 Visual Studio Code 上，執行 Python 程式，繪製圖表。