SciLab 介紹 [1]

Scilab Group [2]　　李運璋 (翻譯) [3]

[1]本文排版成三種格式: 螢幕閱讀格式 、A4 列印格式 及 HTML 網頁格式 讀者可視不同需要參考。有關 Scilab 請參考: http://www-rocq.inria.fr/scilab/ 或 中科院 Scilab 中文版主網頁 。

[2]INRIA Meta2 Project/ENPC Cergrene

[3]中山科學研究院二所

目 錄

第一章

Scilab 簡介

內 容

1.1. 中文版前言

本文以英文版 INTRODUCTION TO SCILAB 爲本進行翻譯，但對原文章節順序略作調動。原文所參考之 Scilab 爲 2.x 之版本，而中文版翻譯時所對照之軟體版本爲 3.1.1 版 (ADE 中文修正版)。因此原文之範例皆在新版 Scilab 重測，並做必要之修改以適應新版軟體。

本文中文 LaTeX 原文隨附於 ADE Scilab 中文版 中之次目錄 doc\cintro 中，讀者系統中若安裝 Omega–16 位元科學排版系統 可以自行修改排版。若想自行排版出 HTML 格式，請安裝 LaTeX2Html–LaTeX 轉 HTML 。

1.2. Scilab 是什麼?

Scilab 由法國國家資訊及自動化研究所 (INRIA) 主導開發，以原始碼形式自由散發。 Scilab 主要應用於動態系統模擬、控制及訊號處理等工程問題中。

Scilab 由三部分組成編譯連結而成：解譯器 (interpreter)， Scilab 巨集函數庫及 Fortran 與 C 程式庫。這些 Fortran 與 C 程式庫 (嚴格來講，並不屬於 Scilab，但被解譯器於內部呼叫) 個別都是有趣的主題，大部分可從 Netlib 下載取得，少部分由 Scilab 修改以配合 Scilab 之解譯器效能。

Scilab 解譯器語法上最主要的特性在能夠直接處理矩陣：矩陣資料的結合 (concatenation)，抽取 (extraction) 或轉置 (transpose) 以及加法及乘法運算都能直接操作。除了數值矩陣之外，Scilab 也能處理更複雜的的資料結構。例如，控制工程上的轉換函數所需的分式矩陣或多項式矩陣皆是。在 Scialb 中，用 list 或 typed list 資料結構能夠以符號的形式表達複雜的數學觀念，例如 Scilab 中的轉換函數 (transfer functions)，線性系統 (linear systems) 及圖論 (graphs) (參考 Section 4.7) 等皆是。

多項式 (Polynomials)，多項式矩陣 (polynomials matrices) 及轉換矩陣 (transfer matrices) 在 Scilab 內都已預先定義，而操作這類矩陣的語法和操作普通向量及矩陣的方法完全一樣。

對非線性系統之分析應用， Scilab 也提供強大的基礎功能，不管是顯式或隱式動態性統都可藉以進行數值模擬或積分。而 Scilab 所附的應用程式 scicos 能夠以圖形方式定義及模擬複雜連結之混合動態系統。

最佳化 (optimization) 數值工具則提供解非線性最佳化 (包含不可微分類型之最佳化)，二次式 (quadratic) 最佳化或線性最佳化功能。

Scilab 是一開放的程式環境，函數及程式庫之產生完全由使用者掌控 (參考 Chapter 5)。函數或程式庫也被視爲一個 Scilab 的資料物件。例如， Scilab 所定義的函數可以像一般資料一樣透過參數傳遞給另一個 Scilab 函數內部加以應用。

Scilab 所提供的字串 (character string) 資料結構可以在 Scilab 程式內動態生成 Scilab 函數。字串可以形成更複雜的字串矩陣 (Matrices of character strings)，其操作語法與一般矩陣相同。

最後， Scilab 可以很容易的設計介面與 Fortran 或 C 語言函數溝通。因此可以在 Scilab 內使用許多標準程式庫。

Scilab 的設計哲學 (philosophy) 是提供以下計算環境：

- 資料結構必須變化且彈性，而語法則須自然又易於使用。
- 提供合理尺度之基礎功能，作爲各類型計算之共用基礎。
- 作爲一個開放式架構，新的功能能夠容易加入到 Scilab 底層。 Scilab 系統中內含輔助工具 intersci 能用以開發連結 Fortran 或 C 程式之介面函數。
- 以工具箱 (toolboxes) 提供函數支持程式庫開發者進行各類特定應用開發 (線性控制，訊號處理，網路分析，非線性控制等)。

本文目的爲提供使用者有關 Scilab 功能之概念。所有 Scilab 函數之線上文件可以在 Scialb 境中，以 help 指令取得。

1.3.軟體架構

Scilab 系統軟體分布在幾個次目錄內。主目錄 SCIDIR 內含檔案: scilab.star (啓動執行程序)，英、法、中文版權宣告檔 licence.txt 、 license.txt t 、 版權.txt 組態檔 configure。各次目錄內容如下：

- bin 內含編譯後之執行檔。

  Scilab 在 MS–Windows 環境的執行檔爲 wscilex.exe，而在 Unix/linux 環境則爲 scilex.exe。此目錄內也包含其他輔助工具，例如：intersci.exe,intersci-n.exe 爲撰寫 Fortran， C 函數之 Scilab 介面之輔助工具。Blatexprs.exe,Blatexpr2.exe 爲製作 Latex 文件所需 Postscript 圖檔的輔助工具。

- demos 內含 Scilab 範例之次目錄。

  這些範例對啓發初習者學習 Scilab 很有用處。在 Scilab 交談式環境中，Scilab/範例 按鍵會啓動本目錄內之檔案 alldems.dem 。大部分的繪圖指令以一些簡單範例展示使用方法以協助學習。注意，如果執行繪圖指令時完全不輸入參數時就會執行一個使用到此指令的繪圖範例。

- examples 內含許多以 Fortran ,C 及 C++語言連結 Scilab 環境的範例。
- doc 內含 Scilab 原始文件: LaTeX , PDF 檔。

  例如本文原始檔位於 SCIDIR/doc/cintro/cintro.tex。
- pvm3 內含網路分散計算境 PVM3。
- imp 列印及處理 Postscript 圖檔所需的程式庫
- libs Scilab 編譯過程所需的編譯程式庫，這些程式庫是由 routines 內的 Fortran 及 C 程式庫之原始碼編譯而成。
- macros 內含以 Scialb 語言所設計的巨集函數庫。

  這些函數在 Scilab 系統內可以直接線上操作。新的程式庫很容易再添加 (請參考目錄內檔案 Makefile–Unix 或 Makefile.mak–Windows)。此目錄內含許多次目錄，包含控制、訊號處理工具箱 (Toolboxes)。嚴格來講， Scilab 並不依照這些工具箱功能來架構；以 signal 次目錄而言，次目錄並不包含完整的訊號處理功能，但所有的函數皆爲訊號處理相關函數。
- man 內含中、英、法文線上文件，主要格式爲 HTML 及 XML。

  要得到某個項目的線上說明，可以利用指令 help 即能啓動線上手冊。在 Scilab 的交談式介面內也可以開啓線上文件。讀者也可以利用指令 apropos key-word 來詢有關 key-word 的線上文件。例如 apropos cos 查詢函數 cos 相關文件。在 man\cht 各次目錄內的 whatis.htm 檔中之文字是能夠在 Scilab 中被查詢的保留字 (key-word)。要添加能夠被查詢的次目錄，開發者必須改 Scilab 變數 %helps 內之資料。
- maple 內含 Maple 原始碼，用以轉換 Maple 物件到 Scilab 函數。

  爲了效率因素，轉換是透過 Fortran 碼再動態聯結到 Scilab。
- routines 內含 Fortran 和 C 數值程式庫。

  次目錄 default 需要特別注意，因爲內含使用者用來特殊化 Scilab 以符合個人需要的程式碼。 尤其是 ODE/DAE 模擬或者是最佳化所需的 C 或 Fortran 可以置放在此處 (也可以透過動態連結方式與系統結合)。
- examples 內含各類 Fortran 或 C 程式碼與 Scilab 連結的範例。
- intersci 內含建立 Fortran 或 C 與 Scilab 連結的介面函數。編譯後之工具 (intersci 及 intersci-n) 置於 bin 內。
- scripts 內含 Unix 環境的一些 scripts 檔。
- tests Scilab 的測試檔，檔案 “demos.tst” 測試所有範例。

1.4.安裝及系統需求

Scilab 以原始碼方式散發，在 Windows 上編譯 Scilab 程序如下：

- 需要先準備好一版編譯後之 TCL/TK 環境，放置於上層目錄； ADE Scilab 中文版 所散發的版本已內含 TCL/TK 環境。
- 修改 Makefile.inc.mak 內容以符合個人需要。

- 需要 Visual C++ 6.0 以上編譯器。 Scilab 中已內含一版 f2c 工具，能將 Fortran 77 轉成 C 再編譯連結，因此可以不用 Fortran 編譯器。如果要選用其他 Fortran 編譯器，可以在檔案 Makefile.incl.mak 中設定。
- 執行 make standard 即能編譯出 Scilab

Scilab 內部使用一堆疊 (stack) 作爲解譯器記憶資料的位置。堆疊空間的大小可以透過指令 stacksize，可在啓動指令檔 (scilab.star) 用變數 newstacksize 來設定內定數值。

有關安裝問題，與可參考主目錄內的 README 檔案。

1.5. 文件

本文由 LaTeX 中文碼 intro.tex 編譯而成。原始 LaTeX 碼位於 doc\cintro 目錄內。

其他相關文件爲

- 訊號處理
- SCICOS –動態系統建構及模擬器
- Scilab 內部技術
- Intersci : Scilab 介面工具
- LMITOOL:線性矩陣不等式 (LMILinear Matrix Inequalities) 最佳化套件
- Metanet User’ s Guide and Tutorial) 最佳化套件
- 中科院 Scilab 中文版主網頁
- http://www-rocq.inria.fr/scilab/

第二章

Scilab 簡易教學

內 容

2.1.啓動

Scilab 是由目錄 SCIDIR/bin 內執行檔 WScilex (Windows) 或 Scilex (unix) 所啓動，其中 (SCIDIR 代表 Scilab 的安裝目錄，在 Windows 環境中，執行檔 Scilex 爲沒有圖形介面的版本。

啓動中文 Scilab 後可以看到以下訊息：

```
 ---------------------------------------------
                  scilab-3.1.1

            Copyright (c) 1989-2005
          Consortium Scilab (INRIA, ENPC)
          中山科學研究院 ADE 中文修正版
 ---------------------------------------------

啟動檔 (Startup) 執行中 .....
  載入初始環境


 -->
```

最後一行出現 --> 時，代表 Scilab 正等待使用者輸入指令，此時若輸入指令 help 會開啓線上手冊。

與 Scilab 第一次接觸可以由 Scilab 範例開始。在 Scilab 交談式介面的 線上文件/Scilab 範例中按鍵選擇執行各範例。範例執行過程會停頓等待使用者輸入，這時僅需在 Scilab 中輸入 Enter 鍵即可繼續進行。

也可以以 Scilab 指令啓動範例，例如：

```
cd SCI\demos;      // 改變工作目錄
exec alldems.dem // 執行指令檔 alldems.dem
```

內定載入系統的巨集程式庫是由檔案 SCIDIR/scilab.star 中設定。有經驗的 Scilab 開發者可以改變此檔內容以符合個人需要。

2.2.編輯

Scilab 交談式介面的 編輯選項，或指令 scipad 能啓動 Scilab 編輯器以編輯 Scilab 指令檔，例如：

```
scipad('alldems.dem')  // 編輯 alldems.dem
```

將目前工作目錄內的檔案 alldems.dem 載入編輯器內。檔案在編輯狀態，可以隨時透過 Execute/Load into Scilab 選項將編輯內容載入 Scilab 系統測試，方便開發程式。

指令行的游標可以用 (←↑↓→) 鍵移動位置以進行編輯。

- Ctrl-p 上一行
- Ctrl-n 下一行
- Ctrl-b 移到下一字元
- Ctrl-f 移到前一字元
- Delete 清除目前字元
- Ctrl-h 清除前一字元
- Ctrl-d 清除目前字元
- Ctrl-a 移到第一字元
- Ctrl-e 移到行尾
- Ctrl-k 清除到行尾
- Ctrl-u 放棄本行
- !prev 叫出前面以 prev 開頭的指令行，例如 !hel 可能會叫前面執行的 help 指令行
- Ctrl-c 中斷 Scilab 並停頓等待輸入

2.3.變數宣告、分行、多行連結及註解

現在可以開始試試一些簡單的指令了。下列 Scilab 螢幕內容中，以 --> 開始指令行是使用者輸入的，其餘行爲 Scilab 輸出結果。輸入行後面要記得鍵入 Enter。

```
-->a=1;       //注意這兩行無輸出結果

-->A=2;

-->a+A        //<-- 這一行會輸出
 ans  =

    3.

-->//兩指令可以利用分號結成一行

-->c=[1  2];b=1.5
 b  =

    1.5
```

```
-->//一個指令也可以利用 ... 分為數行

-->u=1000000*(a*sin(A))^2+...
-->       2000000*a*b*sin(A)*cos(A)+...
-->       1000000*(b*cos(A))^2
 u  =

    81268.994
```

此例中，可學到

- Scilab 變數不需要事先宣告，且大小寫敏感。此例變數 a 和 A 分別設定爲 1 和 2。
- 行尾後面的分號；停止將計算結果輸出到螢幕
- 注意未加分號；的輸入行，在螢幕上都有輸出結果
- 最後一個指令利用 ... 數行字串連接成一個指令
- 符號 // 後面代表程式註解，不引響計算結果

2.4.計算環境、系統變數、常數及特殊變數

```
-->a=1;b=1.5;

-->2*a+b^2
 ans  =

    4.25

-->//We have now created variables and can list them by typing:

-->who
your variables are...

 ans          b          a          bugmes     scicos_pal          %helps
 with_pvm   WSCI       home       SCIHOME    PWD         TMPDIR     MSDOS
 SCI          sparselib             xdesslib   percentlib          polylib
 intlib     elemlib    utillib    statslib   alglib      siglib     optlib
 autolib    roblib     soundlib   metalib    armalib     tkscilib   tdcslib
 s2flib     mtlblib    %F         %T         %z          %s         %nan
 %inf       COMPILER   %gtk       %pvm       %tk         $          %t
 %f         %eps       %io        %i         %e
```

```
 using          6334  elements   out  of      5000000.
              and              52  variables  out  of          9231

your  global  variables  are...

 LANGUAGE   %helps      demolist   %browsehelp            LCC
 %scipad_language
 using          1076  elements   out  of        11000.
              and               6  variables  out  of           767
```

- 指令 who 列印出目前定義的變數，包含剛剛設定的 a,b,c,A 及所有系統變數
- 以 % 起頭的變數代表常數
- 變數 ans 代表剛剛計算出的結果

2.5.矩陣數內容的設定及讀取

```
-->I=1:3
 I  =

!    1.      2.      3.  !

-->W=rand(2,4);

-->W(1,I)
 ans   =

!      .2113249        .0002211        .6653811  !

-->W(:,I)
 ans   =

!      .2113249        .0002211        .6653811  !
!      .7560439        .3303271        .6283918  !

-->W($,$-1)
 ans   =

       .6283918
```

- 定義整數指標向量 (a vector of indices) I
- W 爲 2 x 4 隨機矩陣
- W(1,I) 取得 $W_{1,1}, W_{1,2}, W_{1,3}$ 數值
- W(:,I) 取得 $W_{1,1}, W_{1,2}, W_{1,3}$ 及 $W_{2,1}, W_{2,2}, W_{2,3}$ 數值這裡矩陣指標： 代表所有範圍
- 矩陣指標 $ 代表最後一行或最後一列，因此 W($,$-1) 代表 W(2,3)

2.6.函數呼叫

```
-->sqrt([4  -4])
 ans  =

!   2.      2.i  !
```

呼叫函數 sqrt 將向量 $(4, -4)$ 求方根得複數向量 $(2, 2i)$

2.7.使用多項式

```
-->p=poly([1 2 3],'z','coeff')
 p  =

                  2
    1 + 2z + 3z

-->//p is the polynomial in z with coefficients 1,2,3.

-->//p can also be defined by :

-->s=poly(0,'s');p=1+2*s+s^2
 p  =

                2
    1 + 2s + s
```

此例分別以符號 z, 及 s 作爲符號變數，產生多項式。

2.8.多項式矩陣及行列式之符號運算

```
-->exec('d1p4.code',-1);

-->M=[p, p-1; p+1 ,2]
 M   =

!              2           2 !
!   1 + 2s + s      2s + s   !
!                            !
!              2             !
!   2 + 2s + s      2        !

-->det(M)
 ans   =

           2     3    4
    2 - 4s - 4s - s
```

- 以指令 exec 載入前例之程式碼，d1p4.code
- 定義多項式矩陣 M
- 以指令 det 求取多項式矩陣 M 行列式值

注意本例之運算皆爲符號運算，而不是數值計算。

2.9.分式多項式矩陣及其分子及分母多項式

```
-->exec('d1p5.code',-1);

-->F=[1/s      ,(s+1)/(1-s)
-->       s/p     ,    s^2   ]
 F   =

!   1               1 + s   !
!   -               -----   !
!   s               1 - s   !
!                           !
!                    2      !
!       s           s       !
!   ---------        -      !
```

```
!              2           !
!   1 + 2s + s      1      !

-->F.num
 ans  =

!   1      1 + s  !
!                 !
!          2      !
!   s      s      !

-->F.den
 ans  =

!   s              1 - s  !
!                        !
!              2         !
!   1 + 2s + s     1     !

-->F.num(1,2)
 ans  =

    1 + s
```

- 以指令 exec 載入前例之程式碼，d1p5.code
- 定義分式多項式矩陣 (matrix of rational polynomials) F
- 以指令 F.num 及 F.den 分別得到分式多項式矩陣 F 的分子及分母多項式矩陣
- 以指令 F.num 本身也是一矩陣，因此如同矩陣一樣利用指標取值。例如本例中 F.num(1,2) 即是

此例展示： Scilab 之複雜物件，可以擁有屬於自身之資料物件。

2.10.計算環境之層次

```
-->exec('d1p6.code',-1);

-->pause

-1->pt=return(s*p)
```

```
-->pt
 pt  =

          2    3
   s + 2s + s
```

- 以指令 pause 進入下一層計算環境，上一層的變數環境在下一層中還可使用。
- Scilab 以新的回應字串 -1-> 代表新環境。
- 以指令 return 返回上一層環境，如果不帶參數會清除下層環境所產生的變數後才回到上層，否則只傳回 return 參數內容回上一層。
- pause ,return用於偵錯時非常方便

2.11.頻率響應及繪圖

```
-->exec('d1p6.code',-1);

-->F21=F(2,1);v=0:0.01:%pi;frequencies=exp(%i*v);

-->response=freq(F21.num,F21.den,frequencies);

-->plot2d(v,abs(response),style=-1,rect=[0,0,3.5,0.7],nax=[5,4,5,7]);

-->xtitle(' ','radians','magnitude');
```

- 取得前例中，分式多項式矩陣數值 F(2,1)
- v=0:0.01:%pi 代表一向量 $v = (0., 0.01, 0.02, ...\pi)$
- 計算向量 e^{iv_i}
- 由 Laplace 轉換和 Fourier 轉換之間的關係 ($s = i\omega$)，計算頻率響應
- plot2d 繪圖 (參考 Figure 2.1).

2.12.合成函數

```
-->exec('d1p8.code',-1);

-->w=(1-s)/(1+s);f=1/p
```

```
 f  =

          1
    ---------
                2
    1 + 2s + s

-->horner(f,w)
 ans  =

                2
    1 + 2s + s
    ----------
          4
```

函數 horner 以數值或符號計算合成函數，本例計算 $f(w(s))$

2.13.動態系統、狀態空間及轉換函數

```
-->A=[-1,0;1,2];B=[1,2;2,3];C=[1,0];

-->Sl=syslin('c',A,B,C);

-->ss2tf(Sl)
 ans  =

!      1          2     !
!   -----      -----    !
!   1 + s      1 + s    !
```

- 考慮狀態方程式

$$\begin{aligned}\dot{\mathbf{x}} &= A\mathbf{x}+\mathbf{B}\mathbf{u}\\ \mathbf{y} &= C\mathbf{x}\end{aligned}$$

其中 A,B,C 爲矩陣，代表此動態系統

- 函數 ss2tf 由狀態方程式表示法轉成轉移函數表示法

2.14.狀態空間資料的詳細內容

```
-->s=poly(0,'s');

-->R=[1/s,s/(1+s),s^2]
 R   =

!                       2 !
!   1         s         s  !
!   -       -----       -  !
!   s       1 + s       1  !

-->Sl=syslin('c',R);

-->tf2ss(Sl)
 ans   =


         ans(1)    (state-space system:)

!lss   A   B   C   D   X0   dt   !

         ans(2) = A matrix =

! -   .5   -   .5 !
! -   .5   -   .5 !

         ans(3) = B matrix =

! - 1.      1.      0. !
!   1.      1.      0. !

         ans(4) = C matrix =

! - 1.   - 1.110E-16 !

         ans(5) = D matrix =

!                   2 !
!   0       1       s  !
```

```
        ans(6) = X0 (initial state) =

!   0. !
!   0. !

        ans(7) = Time domain =

 c
```

- 函數 tf2ss 由轉移函數表示法轉成狀態方程式表示法
- Scilab 以 typed lists 代表狀態空間物件，內含 7 項子物件除第 1 項儲存名稱外，其餘 6 項爲 A B C D ，爲系統矩陣， X0 爲初值，dt 爲模擬步進 (time step) 時間。

2.15.動態子系統之並聯

```
-->exec('d1p10.code',-1);

-->exec('d1p11.code',-1);

-->sl1=[Sl;2*Sl+eye()]
 sl1   =

!                          2   !
!   1            s         s    !
!   -          -----       -    !
!   s          1 + s       1    !
!                               !
!                            2  !
!   2 + s       2s         2s   !
!   -----       ----       ---  !
!     s        1 + s       1    !

-->size(sl1)
 ans   =

!   2.    3. !

-->size(tf2ss(sl1))
 ans   =
```

! 2. 3. !

- 由兩子系統 (Sl 及 $2Sl + I$) 的轉換函數並連成一新系統

2.16.設計 Scilab 函數

```
-->function Cl=compen(Sl,Kr,Ko)
-->  [A,B,C,D]=abcd(Sl);
-->  A1=[A-B*Kr ,B*Kr; 0*A ,A-Ko*C]; Id=eye(A);
-->  B1=[B; 0*B];
-->  C1=[C ,0*C];Cl=syslin('c',A1,B1,C1)
-->endfunction
```

compen(Sl,Kr,Ko) 展示設計一個 Scilab 函數 compen 。此函數由原系統狀態空間表示參輸 Sl及輸出及制動器 Gain 值 Ko, Kr，形成一個補償器次系統。

2.17.補償器設計範例

```
-->exec('d1p13.code',-1);

-->A=[1,1 ;0,1];B=[0;1];C=[1,0];Sl=syslin('c',A,B,C);

-->Cl=compen(Sl,ppol(A,B,[-1,-1]),...
-->                              ppol(A',C',[-1+%i,-1-%i])');

-->Aclosed=Cl.A,spec(Aclosed)
 Aclosed  =

!  1.    1.    0.    0. !
! - 4.  - 3.    4.    4. !
!  0.    0.  - 3.    1. !
!  0.    0.  - 5.    1. !
 ans  =

! - 1.          !
! - 1.0000000 !
! - 1. + i     !
! - 1. - i     !
```

- 載入前例所設計的 compen 函數 (檔案 d1p13.code)
- 利用函數 ppol 以所需要的極點 (pole) 位置計算出 gain 陣 Kr 及 Ko。
- 呼叫前例所設計的函數 compen 形成補償器次系統。
- 利用函數 spec 檢查補償器之特徵植 (eigen values)

2.18.執行系統指令

```
-->//Saving the environment in a file named : myfile

-->save('myfile')

-->//執行系統指令 del ,清除檔案

-->unix_s('del myfile')
```

指令 unix_s 執行系統指令。

2.19.以 Fortran 或 C 語言提供動態程式庫

```
-->     //提供 C 原始碼, 測試用實際上 C 原始碼可以是已存在的檔案

-->     f1= ['#include <math.h>'
-->          'void fooc(c,a,b,m,n)'
-->           'double a[],*b,c[];'
-->           'int *m,*n;'
-->           '{'
-->           '    int i;'
-->           '    for ( i =0 ; i < (*m)*(*n) ; i++) '
-->           '       c[i] = sin(a[i]) + *b; '
-->           '}'];

-->     mputl(f1,'fooc.c'); // 產生 C 語言檔案 fooc.c

-->     //產生動態程式庫 (過程中自動產中介檔 (gateway), 軟體建構檔 (Makefile) 及
```

```
-->     // 載入此動態程式庫的程序檔 loader.sce)

-->     ilib_for_link('fooc','fooc.o',[],"c") ;
    generate a loader file
    generate a Makefile: Makelib
    running the makefile
    compilation of fooc
    building shared library (be patient)

-->     // 載入此剛編譯完成的動態程式庫

-->     exec loader.sce ; // loader.sce 為自動產生
shared archive loaded
Link done

-->     //測試

-->     a= [1,2,3;4,5,6]
 a   =

!    1.      2.      3. !
!    4.      5.      6. !

-->     b= %pi
 b   =

     3.1415927

-->     [m,n]=size(a);

-->     // 輸入          參數位置     型態

-->     // a                2        double

-->     // b                3        double

-->     // n                4        integer

-->     // m                5        integer

-->     // 輸出          參數位置     型態      宣告大小
```

```
-->     // c                    1       double    [m,n]

-->     c=call("fooc",a,2,"d",b,3,"d",m,4,"i",n,5,"i","out",[m,n],1,"d")
 c  =

!    3.9830636      4.0508901      3.2827127 !
!    2.3847902      2.1826684      2.8621772 !


-->      //提供 Fortran 原始碼, 測試用實際上 Fortran 原始碼可以是已存在的檔案

-->     f1= ['         subroutine foof(c,a,b,n,m)'
-->          '         integer n,m'
-->          '         double precision a(*),b,c(*)'
-->          '         do 10 i=1,m*n '
-->          '           c(i) = sin(a(i))+b'
-->          '    10 continue'
-->          '         end'];

-->     mputl(f1,'foof.f'); // 產生 Fortran 語言檔案 foof.f

-->     //creating the shared library (a gateway, a Makefile and a loader are

-->     //generated.

-->     ilib_for_link('foof','foof.o',[],"f") ;
   generate a loader file
   generate a Makefile: Makelib
   running the makefile
   compilation of foof
   building shared library (be patient)

-->     // 載入此剛編譯完成的動態程式庫

-->     exec loader.sce ; // loader.sce 為自動產生
shared archive loaded
Link done


-->     //using the new primitive
```

```
-->      a= [1,2,3;4,5,6];b= %pi;

-->      [m,n]=size(a);

-->      c=call("foof",a,2,"d",b,3,"d",m,4,"i",n,5,"i","out",[m,n],1,"d")
 c  =

!    3.9830636      4.0508901      3.2827127 !
!    2.3847902      2.1826684      2.8621772 !
```

- 利用將 C 或 Fortran 程式寫入字串向量再利用 mput 指令寫入檔案內。實際工作上， C 或 Fortran 檔案來源以外部提供爲主。此例， C 及 Fortran 碼內實作出以下向量函數

$$\mathbf{c} = \sin \mathbf{a} + \mathbf{b}$$

  其中，$\mathbf{a}, \mathbf{c}$ 向量，b 爲純量。

- 以高階指令 ilib_for_link 將 C 或 Fortran 程式編譯並連結出動態原始碼。同時自動產生此動態程式庫的載入指令 loader.sce 下次使用此函數功能時，僅需呼叫此指令即可。
- 指令 call 先將動態程式庫內函數 foof 的呼叫參數及 Scilab 變數相互對應後再呼叫此動態程式庫函數 foof。

2.20.常微分方程組應用範例

```
-->function  ydot=f(t,y),ydot=[a-y(2)*y(2)  -1;1  0]*y,endfunction

-->a=1;y0=[1;0];t0=0;instants=0:0.02:20;

-->y=ode(y0,t0,instants,f);

-->plot2d(y(1,:),y(2,:),style=-1,rect=[-3,-3,3,3],nax=[10,2,10,2])

-->xtitle('Van  der  Pol')
```

本例求解常微分方程組

$$\dot{\mathbf{y}} = \left[\begin{array}{c|c} a - y_2^2 & -1 \\ \hline 1 & 0 \end{array}\right] \left[\begin{array}{c} y_1 \\ y_2 \end{array}\right]$$

步驟如下：

- 以 Scialb 語言定義向量微分方程組，$\dot{\mathbf{y}} = f(t, \mathbf{y})$，此函數也可直接用 C 或 Fortran 語言設計。
- 設定向量 $\mathbf{y}$ 的初值 y0=[1;0]；及初時 t0=0
- instants=0:0.02:20；設定輸出時間位置爲 0 到 20 秒間，每 0.02 秒輸出一筆數據。
- 呼叫常微分方程組步進指令 ode 計算出輸出向量 $\mathbf{y}$。注意函數 $f(t, \mathbf{y})$ 以參數形式出現在指令 ode 的輸入參數內。
- 以繪圖指令 plot2d 繪出結果如圖 2.2

2.21.運算子設計

```
-->m=['a'  'cos(b)';'sin(a)'  'c']
 m   =

!a          cos(b)   !
!                    !
!sin(a)   c          !

-->//m*m'    --> error message : not implemented in scilab

-->function x=%c_m_c(a,b)
-->   [l,m]=size(a);[m,n]=size(b);x=[];
-->   for j=1:n,
-->     y=[];
-->     for i=1:l,
-->       t='';
-->       for k=1:m;
-->         if k>1 then
-->           t=t+'+('+a(i,k)+')*'+'('+b(k,j)+')';
-->         else
-->           t='(' + a(i,k) + ')*' + '(' + b(k,j) + ')';
-->         end
-->       end
-->       y=[y;t]
-->     end
-->     x=[x  y]
-->   end
-->endfunction

-->m*m'
```

```
 ans    =

!(a)*(a)+(cos(b))*(cos(b))    (a)*(sin(a))+(cos(b))*(c)    !
!                                                          !
!(sin(a))*(a)+(c)*(cos(b))    (sin(a))*(sin(a))+(c)*(c)    !
```

本例展示如何針對特定資料結構設計一個新的運算子。新的資料結構爲一字串矩陣，而運算子則爲字串矩陣相成的運算子。

- Scilab 運算子之定義如同一般函數定義，但名稱必特定格式。以此例而言 格式規定請參考 5.3 內容。
- 一旦字串矩陣的乘法運算已定義，則指令 m*m' 能夠進行矩陣相乘的符號運算。

2.22.函數參數的傳遞

```
-->function  y=calcul(x,method),z=method(x),y=poly(z,'x'),endfunction

-->function  z=meth1(x),z=x,endfunction

-->function  z=meth2(x),z=2*x,endfunction

-->calcul([1,2,3],meth1)
 ans    =

                    2    3
  - 6 + 11x - 6x + x

-->calcul([1,2,3],meth2)
 ans    =

                     2    3
  - 48 + 44x - 12x + x
```

本例測試如何將 Scilab 函數當作一個參數傳遞給另一個函數

- cacul , meth1 及 meth2 爲使用者自訂的函數。
- cacul 透過第二個參數，將傳入之參數當作函數一般運用。本例分別傳遞 meth1 及 meth2 兩函數以檢驗計算結果之不同。

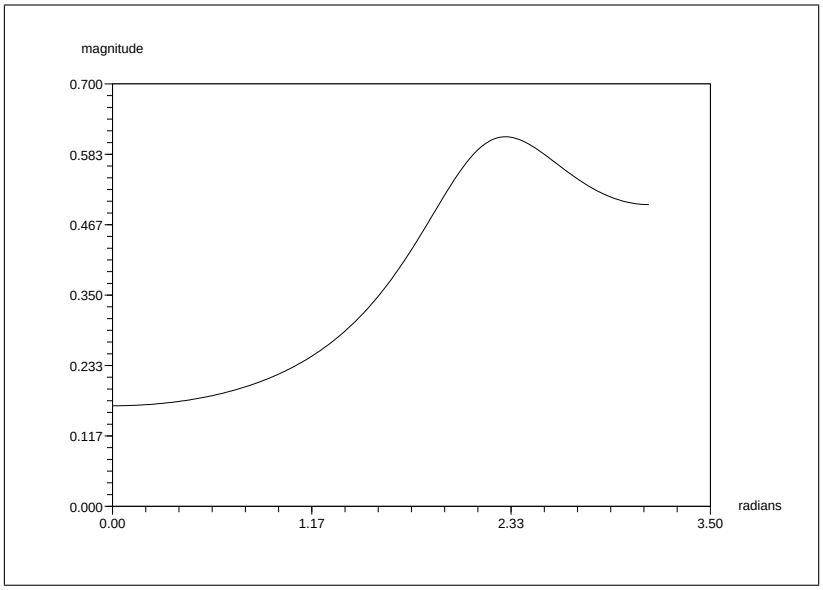


圖 2.1: A Simple Response

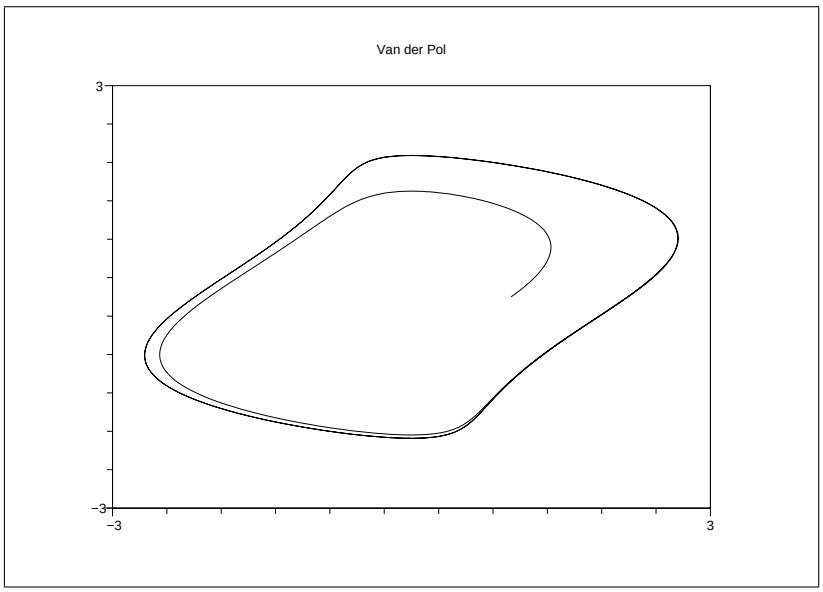


圖 2.2: Phase Plot

第三章

操作環境

內容

本章說明 Scilab 的基本環境設定、輸入輸出、交談介面及常用函數。

3.1.輸入輸出及環境

本節介紹如何自動設定計算環境以及如何將計算環境儲存及載入系統內。

3.1.1.環境

Scilab 啓動後，基礎變數及函數已可使用。指令 who 能顯示目前可變數名稱。指令 whos() 能夠提供變數更詳細的資料。

指令 who 也指出目前可用之元素及變數數目。函數的線上說明可以利用 help <函數名稱> 指令查詢。

目前之變數內容可以利用 save 將她們存在一外部檔案。而存在外部檔案之變數內容，可以藉指令 load 載入系統內。

注意，在指令 clear x y 之後，變數 x 和 y 就不存在，必須重新宣告才能使用。 save 指令若不含參數，會將 Scilab 環境中所有變數存起。同樣的，clear 指令若不帶參數，也會將所有變數、函數、程式庫清空。

函數程式庫以 lib 指令載入環境。 disp 指令可以查詢程式庫內有那些函數。

3.1.2.Scilab 之啓動指令

當 Scilab 啓動時 (在 Windows 環境執行 wscilex)，檔案 SCIDIR/scilab.star 的內容會自動執行。如果啓動 Scilab 的工作目錄內有一檔案名爲 .scilab 此檔案也會接著被自動執行。一般而言，SCIDIR/scilab.star 內容爲系統設定。而 .scilab 爲使用者設定。在 Unix/Linux 環境中，個人主目錄 (home directory) 內的 .scilab 檔會緊接著系統 SCIDIR/scilab.star 檔之後而在工作目錄之 .scilab 之前執行。

在 .scilab 或 SCIDIR/scilab.star 檔內，將常用函數或程式庫以指令 getf、 exec 或 load 先行載入，對經常性之作業非常方便。

3.1.3.輸入及輸出

雖然指令 save 及 load 可以將參數內容存入檔案及載入系統中，但檔案內容卻是不可讀性之格式。 Scilab 另外提供類似 Fortran 之輸出入指令 read 及 write。使用這兩函數的語法如下列所示：

```
--> x=[1 2 %pi;%e 3 4]
 x           =

!   1.          2.      3.1415927  !
!   2.7182818   3.      4.         !

--> write('x.dat',x)

--> clear x

--> xnew=read('x.dat',2,3)
 xnew        =

!   1.          2.      3.1415927  !
!   2.7182818   3.      4.         !
```

注意 read 需指定矩陣 x 之行列數目。也可以使用較複雜的格式。

同樣的，指令 mfscanf 及 mfprintf 也提供如同 C 一般的輸出入格式。使用範例如下：

```
--> x=[1 2 %pi;%e 3 4]
 x           =

!   1.          2.      3.1415927  !
!   2.7182818   3.      4.         !
```

```
--> fd=mopen('x_c.dat','w')

--> mfprintf(fd,'%f  %f  %f\n',x)

--> mclose(fd)

--> clear  x

--> fd=mopen('x_c.dat','r')

--> xnew(1,1:3)=mfscanf(fd,'%f  %f  %f\n')  ;

--> xnew(2,1:3)=mfscanf(fd,'%f  %f  %f\n')

 xnew   =

!    1.              2.      3.141593  !
!    2.718282      3.      4.          !
--> mclose(fd)
```

3.2.線上手冊 (Help)

Scilab 之線上手冊可以輸入 help 指令啓動。線上手冊本身爲交談式設計，可以進一步查詢函數內容。但也可以在 Scilab 內輸入 help 函數名稱 直接查詢函數用法。

3.3.常用函數

以下是一些常用函數清單，使用者以這些函數爲進入點查詢，再透過線上文件之間的相互連結 (見線上文件中的 其他參考連結) 可以進一了解其他細節。

- 基本函數: sum, prod, sqrt, diag, cos, max, round, sign, fft
- 排序: sort, gsort, find
- 特殊矩陣: zeros, eye, ones, matrix, empty
- 線性代數: det, inv, qr, svd, bdiag, spec, schur
- 多項式: poly, roots, coeff, horner, clean, freq
- 交談介面: x_choose, x_dialog, x_mdialog, getvalue, addmenu
- 線性系統: syslin
- 隨機變數: rand

- Scillab 程式語言: function, deff, argn, for, if, end, while, select, warning, error, break, return
- 比較運算子: ==, >=, >, =, & (and),| (or)
- 批次執行: exec
- 偵錯: pause, return, abort
- 內插: splin, interp, interpln
- 字串: string, part, evstr, execstr
- 圖形: plot, xset, driver, plot2d, xgrid, locate, plot3d, Graphics
- 常微分方程: ode, dassl, dassrt, odedc
- 最佳化: optim, quapro, linpro, lmitool
- 動態系統連結: scicos
- 加入 C 或 Fortran 函式: link, fort, addinter, intersci

3.4.非線性計算

Scilab 針對非線性模擬及最佳化問題提供很有用工作環境。

3.4.1.基楚觀念

針對非線性計算， Scilab 提供幾個工作機制。

微分方程組的模擬由 ode 指令提供。 ode 指令內部支持許多求解微分方程組之選擇，大部分功能來自 Fortran 程式庫 odepack 。隱式常微分系統之求解可以使用 dassl 。常微分系統也可以有隱式停止時間 (stopping time) 功能，也就是滿足某些條件後才停止計算的功能 (參考 ode 及 dassrt 指令)。

模擬或求解常微方程組的細部選項 (誤差容忍， jacobian，近似階次，時間步進等) 可以利用變數 %ODEOPTIONS 設定。

非線性函數的最佳化問題可以使用 optim 函數求解。 optim 提供選擇不同的演算法 (包含不可微分函數之最佳化) 原始碼來自 INRIA 的 modulopt 程式庫。詳細說明請鍵入 help optim 查詢。

3.4.2.函數參數

特定之 Scilab、 C 或 Fortran 函數可以做爲 Scilab 高階指令 (例如：ode, optim, dassl...) 的參數。這些函數稱爲函數參數 (argument functions) 或外部函數。這些函數參數本身的輸出入參數必須滿足呼叫它們的高階指令所要求的規格。

例如，函數 costfunc 是 optim 的一個函數參數，它的輸出入參數必須爲 [f,g,ind]=costfunc(x,ind) 。以下 Scilab 函數需要函數參數： ode, optim, impl, dassl, intg, odedc, fsolve。

如果計算執行效率需要計較的問題，建議以 C 或 Fortran 語言提供函數參數。次目錄 SCIDIR/routines/default 內有許多這類例子。可參考目錄內的 README 檔案。

這類以 C 或 Fortran 語言設計的函數，可以透過 link 指令動態的聯結到 Scilab 中或者放在 SCIDIR/routines/default 內再從新編譯 Scilab 連結成內部呼叫之函數。

3.5. 交談式選單

再 Unix/Linux 環境中， Scilab 函數內部可以開啓 XWindows 以交談式的方式輸入參數。請參考指令 x_dialog， x_choose， x_mdialog， x_matrix 及 x_message.

3.6. TCL/TK Tk-Tcl 選單

Scilab 也提供 Tk-Tcl 相關函數以提供與作業平台無關之交談介面介面。透過 uicontrol 函數之功能可以設計使用者圖形介面。 Tk-Tcl 相關之指令爲 TK_EvalFile， TK_EvalStr 及 TK_GetVar, TK_Setvar。

第四章

資料型態(Data Types)

內 容

Scilab 內建幾種資料型態 (data types)。 Scilab 之純量 (scalar) 物件有數值、布林值、多項式、字串及分式多項式。有這些純量進一步又定義出各類型之陣列，陣列之數據型態爲各純量型態。 Scilab 另外又提供串列 (lists) 及型態串列 (typed-lists) 資料結構其功能接近 C 語言的 struc，能夠用基本型態組合成成複雜型態。 Scilab 也有稀疏矩陣功能，不過只有數值及布林值兩種純量的稀疏矩陣。本章目的即描述各別之資料型態。

4.1.特殊常數

Scilab 提供以下內定特殊常數： %i, %pi,%e, and %eps。常數 %i 代表虛數 $\sqrt{-1}$，%pi 爲 $\pi = 3.1415927\cdots$，%e 爲指數常數數值爲 $e = 2.7182818\cdots$ 而 %eps 則代表目前計算機的計算精度 (%eps 代表滿足 $1 + \%eps = 1$) 的最大數值)。 %inf 及 %nan 分別代表 “無限大 “ (Infinity) 及 “非數字 “ (NotANumber) 兩義意。%s 代表多項式以符號 s 變數的特殊多項式： poly(0,'s') 也就是一次多項式

%t 及 %f 代表 “眞 “ (true) “假 “ (false) 兩布林值。注意 %t 等同於 1==1，而 %f 等同於 ~%t。

這些變數在 Scilab 是當作事先定義之數值，它們之內容已被系統保護，因此無法刪除，也無法以指令 save 存成檔案。使用者也可以用指令 自訂這些特定常數。設定這類自訂常數最好的方式是在系統或使用者之啓動指令檔 (檔案 scilab.star 或.scilab) 內設定。當然，使用者也可以如下使用 i=sqrt(-1) 而不用 %i。

4.2.數值矩陣 (Constant Matrices)

(中文注：本文數值矩陣，原文 Constant Matrices，但似乎不宜解釋爲常數矩陣。以文章內容判斷，此意爲元素爲實數，整數或複數之矩陣，因此翻譯爲數值矩陣)

Scilab 將純量及向量資料皆視爲矩陣資料的特例，也就是將純量及向量視爲一個退化之矩陣。

4.2.1.純量 (Scalars)

純量是實數或複數。純量的數值可以設定到一個使用者指定的變數名稱。

```
--> 3
 ans  =

    3.

--> a=5+2*%i
 a  =

    5. + 2.i

--> b=-2+%i
 b  =

  - 2. + i

--> a*b
 ans  =

  - 12. + i
```

```
--> c=a+b;

--> c
 c  =

    3. + 3.i
```

注意，每行指令若鍵入 Return 鍵， Scilab 系統就開始計算此行指令的結果。但若最後一個字元是分號 (;) 時，計算結果並會輸出到螢幕上。

4.2.2.向量 (Vectors)

產生向量最常用的方法是利用逗號 (,) 、空白鍵或分號；：

```
--> v=[2 -3+%i 7]
 v  =

!   2.   - 3. + i        7. !

--> v'
 ans  =

!   2.          !
! - 3. - i     !
!   7.          !

--> w=[-3;-3-%i;2]
 w  =

! - 3.          !
! - 3. - i     !
!   2.          !

--> v'+w
 ans  =

! - 1.          !
! - 6. - 2.i !
!   9.          !
```

```
--> v*w
 ans  =

    18.

--> w'.*v
 ans  =

!  -  6.     8.  -  6.i     14.  !
```

- 注意向量內的元素是以逗號 (,) 或空白鍵隔開而行向量之間則以分號；分隔。
- 空矩陣 (empty matrix) 是行、列長度都爲 0 的矩陣；以 [] 代表。
- v' 代表 v 的轉置矩陣。
- 行、列長度皆相等的矩陣可以進行加減運算。
- 符號 (.*) 及 (./) 代表以矩陣元素爲單元 (Element-wise) 的乘法及除法運算。

元素數值以等間隔遞增或遞減的向量可以如下產生

```
--> v=5:-.5:3
 v  =

!   5.     4.5     4.     3.5     3.  !
```

- 產生的向量以第一個輸入數值爲起始值，以第三個數值爲結束值，並以第二個數值爲間隔值，內定之間隔值爲 1。
- ones (zeros) 代表元素數值皆爲 1 (0) 之距陣。

```
--> v=[1 5 6]
 v  =

!   1.     5.     6.  !

--> ones(v)
 ans  =

!   1.     1.     1.  !
```

```
--> ones(v')
 ans  =

!    1. !
!    1. !
!    1. !

--> ones(1:4)
 ans  =

!    1.      1.      1.      1. !

--> 3*ones(1:4)
 ans  =

!    3.      3.      3.      3. !
```

注意 ones 或 zeros 將矩陣、向量內之元素以 1 或 0 取代。

4.2.3.矩陣 (Matrices)

```
--> A=[2 1 4;5 -8 2]
 A  =

!    2.      1.      4. !
!    5.   - 8.      2. !

--> B=ones(2,3)
 B  =

!    1.      1.      1. !
!    1.      1.      1. !

--> A+B
 ans  =

!    3.      2.      5. !
!    6.   - 7.      3. !
```

```
--> A.*B
 ans  =

!    2.      1.      4. !
!    5.   - 8.      2. !

--> A*B'
 ans  =

!    7.      7. !
! - 1.   - 1. !
```

- 矩陣內的行內之元素以空白鍵或逗號 (,) 間隔，而列元素之間則以分號 (;) 間隔。
- 矩陣與純量、向量或其他矩陣之乘法與一般表達形式相同。
- 符號 (.*) 及 (./) 代表以矩陣元素爲單元 (Element-wise) 的乘法及除法運算。

```
-->A=[1 2;3 4]
 A  =

!    1.      2. !
!    3.      4. !

-->B=[5 6;7 8];

-->C=[9 10;11 12];

-->D=[A,B,C]
 D  =

!    1.      2.      5.      6.      9.      10. !
!    3.      4.      7.      8.      11.     12. !

-->E=matrix(D,3,4)
 E  =

!    1.      4.      6.      11. !
!    3.      5.      8.      10. !
!    2.      7.      9.      12. !
```

```
-->F=eye(E)
 F   =

!    1.      0.      0.      0. !
!    0.      1.      0.      0. !
!    0.      0.      1.      0. !

-->G=eye(4,3)
 G   =

!    1.      0.      0. !
!    0.      1.      0. !
!    0.      0.      1. !
!    0.      0.      0. !
```

- 矩陣可以成爲更大矩陣內的一個元素 (次矩陣)。例如 D 矩陣。
- 矩陣之維度 (行、列大小) 可以改變。
- 基本函數 matrix 利用 D 矩陣之資料另產生一個維度不同的 E 矩陣。注意元素排列順序是以一列元素算完再計算下一列的順序排列。
- 函數 eye 產生一個在主對角線上爲 1 的矩陣。若爲方陣相當於一單位矩陣。

4.3.字串矩陣 (Matrices of Character Strings)

單引號或雙引號用來代表字串。加法算 + 用在字串時，代表字串之聯結 (concatenation)。字串矩陣的產生和數值矩陣一樣，是以中括號來代表。字串矩陣一個很重要的特點是能夠用來操作或產生函數。另外數學觀念的符號運算也常利用字串矩陣來完成。

```
-->  A=['x'  'y';'z'  'w+v']
 A   =

!x   y     !
!          !
!z   w+v   !

-->  At=trianfml(A)
 At   =

!z   w+v              !
!                     !
```

```
!0   z*y-x*(w+v)   !

--> x=1;y=2;z=3;w=4;v=5;

--> evstr(At)
 ans   =

!    3.      9. !
!    0.   - 3. !
```

- trianfml 以符號運算方式，將矩陣三角化 (triangularization)。
- 指令 evstr 則將字串矩陣內的符號轉爲數值結果

字串矩陣的一個重要功能是能夠自動產生一個新的函數。以下 Scilab 片段展示如何產生兩變數 s 及 t 的多項式函數。由於 Scilab 只提供單變數多項式功能，但可以利用 Scilab 的串列資料功能建構一新的資料結構以代表變數多項式，本例之多項式爲：$(t^2+2t^3)-(t+t^2)s+ts^2+s^3$。

```
-->getf("../macros/make_macro.sci");

-->s=poly(0,'s');t=poly(0,'t');

-->p=list(t^2+2*t^3,-t-t^2,t,1+0*t);

-->pst=makefunction(p) //pst is a function t->p (number->polynomial)
 pst   =

[p]=pst(t)

-->pst(1)
 ans   =

                2    3
    3 - 2s + s + s
```

此處，多項式變數 s 各項之係數以逗點分隔作爲函數 list 參數以產生串列 p。此串列再做爲參數傳給函數 makefunction 以自動產生一函數 pst。函數 pst 計算出固定變數 t 後之單變數多項式。

函數 makefunction 內容如下：

```
function  [newfunction]=makefunction(p)
```

```
// Copyright INRIA
num=mulf(makestr(p(1)),'1');
for k=2:size(p);
    new=mulf(makestr(p(k)),'s^'+string(k-1));
    num=addf(num,new);
end,
text='p='+num;
deff('[p]=newfunction(t)',text),


function [str]=makestr(p)
n=degree(p)+1;c=coeff(p);str=string(c(1));x=part(varn(p),1);
xstar=x+'^',
for k=2:n,
    if c(k)<>0 then,
    str=addf(str,mulf(string(c(k)),(xstar+string(k-1))));
    end;
end
```

- 函數 makefunction 參數爲一串列 (list p) 而輸出一函數 pst。
- 函數 makefunction 內部呼叫另一函數 makestr 用來製作二元多項式之各項。
- 函數 addf 及 mulf 用來進行兩字串 (符號) 的加法及乘法運算 (例如： addf(x,y) 結果爲字串x+y。
- 產生函數的最基本指令是 deff，此例：函數 p 由兩字串 '[p]=newfunction(t)' 及 text 組成計算二元多項式。

4.4.多項式及多項式矩陣 (Polynomial Matrices)

Scilab 中可以很容易的產生多項式。而多項式之運算語法運算，大部分與數值矩陣相同。
基礎指令 poly 以多項式之係數或根來定義一個新的多項式。

```
--> p=poly([1 2],'s')
 p  =

                2
    2 - 3s + s

--> q=poly([1 2],'s','c')
 q  =
```

```
    1 + 2s

--> p+q
 ans  =

             2
    3 - s + s

--> p*q
 ans  =

               2     3
    2 + s - 5s + 2s

              !--error       4
undefined variable : pdiary


-->quit;quit;quit;quit;quit;quit;
-->exec('SCI/scilab.quit',-1);quit;
```

注意多項式 p 擁有 1 和 2 兩根，而多項式 q 則有 1 和 2 兩係數。指令 poly 的第三個參數用來設定多項式這兩種不同的產生方式。如果 poly 的第一個參數是一個方陣，則採用 roots 選項代表以此方陣之特徵多項式 (characteristic polynomial)。

```
--> poly([1 2;3 4],'s')
 ans         =

               2
  - 2 - 5s + s
```

兩多項式若使用相同的符號變數，可以進行一般之加、減、乘、除運算，
就如同實數或複數，多項式也可以作爲矩陣內之元素。

```
-->s=poly(0,'s');

-->A=[1 s;s 1+s^2]
 A           =

!   1      s       !
```

```
!                  !
!               2  !
!   s      1 + s   !

--> B=[1/s  1/(1+s);1/(1+s)  1/s^2]
 B           =

!   1            1     !
! ------       ------  !
!   s          1 + s   !
!                       !
!     1        1       !
!    ---      ---      !
!               2       !
!   1 + s      s        !
```

由以上範例可知，矩陣可以用多項式或分式多項式作爲內部元素。

4.4.1. 分式多項式之簡化

當分式多項式的分子及分母部份有共同因子時， Scilab 自動呼叫內部函數 simp 進行極點–零點 (pole-zero) 簡化。極點–零點簡化從數值分析的角度，是一個困難的問題，因此函數 simp 通常以保守的方式進行簡化。有時候使用者並不希望 Scilab 的自動化簡機制，這時可以將 Scilab 簡化機制關掉，請利用指令 help simp_mode 查詢必要資料。函數 trfmod 也可以用來建簡化特定極點–零點對 (pole-zero pairs)。

4.5. 布林矩陣

布林 (Boolean) 常數 %t 和 %f 代表眞假兩值。同樣的，布林數也可作爲矩陣內的元素。而操作語法和一般矩陣並無不同。

布林矩陣的特定運算符號爲 == 及 ~，可以用來運算或產生新的布林矩陣。

如果 B 是一布林矩陣，or(B) 和 and(B) 進行邏輯 (logical) or (或) 及 and (且) 運算.

```
-->%t
 %t   =

  T

-->[1,2]==[1,3]
 ans   =

! T  F !
```

```
-->[1,2]==1
 ans  =

!  T  F  !

-->a=1:5; a(a>2)
 ans  =

!    3.    4.    5.  !

-->A=[%t,%f,%t,%f,%f,%f];

-->B=[%t,%f,%t,%f,%t,%t]
 B  =

!  T  F  T  F  T  T  !

-->A|B
 ans  =

!  T  F  T  F  T  T  !

-->A&B
 ans  =

!  T  F  T  F  F  F  !
```

當兩稀疏矩陣進行比較運算時，會對應的產生稀疏布林矩陣。稀疏布林矩陣的操作方式與一般布林矩陣相同。

4.6.整數矩陣

Scilab 共有 6 種整數型態，這些型態都是以數值 8 作爲相同的主類型 (major type) (參考 type 函數)，而次型態 (sub-types) 則用來分辨這 6 種不同整數 (參考 inttype 函數)

- 32 位元有號 (signed) 整數 (sub-type 4)
- 32 位元無號 (unsigned) 整數 (sub-type 14)
- 16 位元有號 (signed) 整數 (sub-type 2)
- 16 位元無號 (unsigned) 整數 (sub-type 23)

- 8 位元有號 (signed) 整數 (sub-type 2)
- 8 位元無號 (unsigned) 整數 (sub-type 12)

由一般數值矩陣 (參考 4.2.3)，可以利用 int32, uint32, int16, uint16, int8, uint8 轉換成各類整數矩陣。

```
-->x=[0 3.2 27 135] ;

-->int32(x)
 ans  =

!0   3   27   135 !

-->int8(x)
 ans  =

!0   3   27   -121!
-->uint8(x)
 ans  =

!0   3   27   135 !
```

同樣的函數，也可以在不同的整數間相互轉換。而 double 函數，則將整數轉爲實數。

```
-->y=int32([2 5 285])
 y  =

!2   5   285 !

-->uint8(y)
 ans  =

!2   5   29 !

-->double(ans)
 ans  =

!   2.    5.    29. !
```

算數及比較 (comparison) 運算也能用於整數型態

```
-->x=int16([1 5 12])
```

```
 x  =

!1   5   12 !

-->x([1 3])
 ans  =

!1   12 !

-->x+x

 ans  =

!2   10   24 !

-->x*x'
 ans  =

 170
-->y=int16([1 7 11])
 y  =

!1   7   11 !
-->x>y
 ans  =

! F  F  T !
```

運算子 &, | 及 ~ 用在這類形態當於 AND, OR 及 NOT 位元運算 (bit-wise operations)。

```
-->x=int16([1 5 12])
 x  =

!1   5   12 !

-->x|int16(2)
 ans  =

!3   7   14 !

-->int16(14)&int16(2)
 ans  =
```

```
 2
-->~uint8(2)
 ans  =

 253
```

4.7.串列 (Lists)

Scilab 的串列型態 (lists) 代表許多物件之集合，這些物件可能屬於不同之型態，甚至是另一個串列。串列主要用於具資料結構之物件。

Scilab 語言中有兩種串列：一般串列及型態串列 (typed-lists)。一般串列由基礎函數 list 定義。以下是一些簡單例子：

```
-->L=list(1,'w',ones(2,2))  //L is a list made of 3 entries
 L  =


         L(1)

      1.

         L(2)

 w

         L(3)

!   1.    1. !
!   1.    1. !

-->L(3)   //extracting entry 3 of list L
 ans  =

!   1.    1. !
!   1.    1. !

-->L(3)(2,2)  //entry 2,2 of matrix L(3)
 ans  =

      1.
```

```
-->L(2)=list('w',rand(2,2)) //nested list: L(2) is now a list
 L  =


         L(1)

      1.

         L(2)


          L(2)(1)

 w

          L(2)(2)

!   0.6653811    0.8497452  !
!   0.6283918    0.6857310  !

         L(3)

!   1.    1.  !
!   1.    1.  !

-->L(2)(2)(1,2)  //extracting entry 1,2 of entry 2 of L(2)
 ans  =

     0.8497452

-->L(2)(2)(1,2)=5; //assigning a new value to this entry.
```

型態串列 (typed-lists) 的第一項元素具有特殊意義。第一項必須是一字串 (用以代表型態) 或一字串向量 (分別代表型態及其他元素的名稱)。型態串列類似於 C++語言中的 class，可用來定義新型態。以下爲使用範例：

```
-->L=tlist(['Car';'Name';'Dimensions'],'Nevada',[2,3])
 L  =


         L(1)
```

```
!Car              !
!                 !
!Name             !
!                 !
!Dimensions   !

          L(2)

 Nevada

          L(3)

!    2.      3. !

-->L.Name      //same as L(2)
 ans  =

 Nevada

-->L.Dimensions(1,2)=2.3

 L  =


          L(1)

!Car              !
!                 !
!Name             !
!                 !
!Dimensions   !

          L(2)

 Nevada

          L(3)

!    2.      2.3 !
```

```
-->L(3)(1,2)
 ans  =

    2.3

-->L(1)(1)
 ans  =

 Car
```

型態串列一個主要特點，是能夠用來定義運算以操作本身資料。例如：兩型態串列 L1 及 L2 之相乘 L1*L2。本文範例，線性系統 (linear systems) 之運算就是利用型態串列這項功能而設計出來的。

4.8.N-維列陣 (N-dimensionnal arrays)

N-維列陣如下定義及運算：

```
-->M(2,2,2)=3
 M  =

(:,:,1)

!    0.    0. !
!    0.    0. !
(:,:,2)

!    0.    0. !
!    0.    3. !

-->M(:,:,1)=rand(2,2)
 M  =

(:,:,1)

!    0.9329616     0.312642   !
!    0.2146008     0.3616361  !
(:,:,2)

!    0.    0. !
!    0.    3. !
```

```
-->M(2,2,:)
 ans  =

(:,:,1)

      0.3616361
(:,:,2)

      3.
-->size(M)
 ans  =

!    2.      2.      2. !

-->size(M,3)
 ans  =

      2.
```

N-維列陣可以利用維度向量及資料項量兩項來定義，例：

```
-->hypermat([2  3,2],1:12)
 ans  =

(:,:,1)

!    1.      3.      5. !
!    2.      4.      6. !
(:,:,2)

!    7.      9.      11. !
!    8.      10.     12. !
```

N-維列陣式是以串列設計並具有兩個成員資料欄的新型態 mlists :

```
-->M=hypermat([2  3,2],1:12);
-->M.dims
 ans  =

!    2.      3.      2. !
```

```
-->M.entries
 ans  =

!    1.  !
!    2.  !
!    3.  !
!    4.  !
!    5.  !
!    6.  !
!    7.  !
!    8.  !
!    9.  !
!    10. !
!    11. !
!    12. !
```

4.9.線性系統之表示式

在 Scilab 中線性系統 (Linear systems) 是利用串列功能所設計出的新型態。定義一線性系統的基礎函數是 syslin。線性系統可以用狀態空間 (state-space) 或轉換函數 (transfer function) 兩種表示法。若使用狀態空間表示法syslin 所接受的參數爲定義此系統的數值矩陣。而若以轉換函數定義線性系統，syslin 的參數則是分式多項式矩陣。

更明確的說，函數 syslin 的輸入參數形式是Sl=syslin('dom',A,B,C,D,x0) 或 Sl=syslin('dom',trmat) 其中 dom 是字串 'c' 或 'd' 代表連續或離散系統。值得注意的是， D可以是一個多項式矩陣，在實務上方便定義常見的系統輸出函數 (output function)； D 及 x0 是選擇性輸入項。trmat 爲一分式多項式，用來代表系統的轉換函數。

syslin 將所有輸入參數轉爲一型態串列 (typed list) Sl。如果狀態空間表示法 Sl 的資料內容爲 tlist(['lss','A','B','C','D'],A,B,C,D,'dom')，則這個型態串列允許使用者直接存取內含元素，例如：Sl('A')，能取得 A–矩陣 (也可用 Sl(2))。

狀態空間及轉移函數表示法之間可以利用函數 ss2tf 及 tf2ss 進行轉換。

```
-->//list defining a linear system

-->A=[0 -1;1 -3];B=[0;1];C=[-1 0];

-->Sys=syslin('c',A,B,C)
 Sys  =


          Sys(1)    (state-space system:)

!lss   A   B   C   D   X0   dt   !
```

```
        Sys(2) = A matrix =

!    0.   - 1. !
!    1.   - 3. !

        Sys(3) = B matrix =

!    0. !
!    1. !

        Sys(4) = C matrix =

! - 1.      0. !

        Sys(5) = D matrix =

     0.

        Sys(6) = X0 (initial state) =

!    0. !
!    0. !

        Sys(7) = Time domain =

 c

-->//conversion from state-space form to transfer form

-->Sys.A   //The A-matrix
 ans   =

!    0.   - 1. !
!    1.   - 3. !

-->Sys.B
 ans   =

!    0. !
!    1. !
```

```
-->hs=ss2tf(Sys)
 hs   =

          1
     ---------
                2
     1 + 3s + s

-->size(hs)
 ans   =

!    1.      1. !

-->hs.num
 ans   =

     1

-->hs.den
 ans   =

                2
     1 + 3s + s

-->typeof(hs)
 ans   =

 rational

-->//inversion of transfer matrix

-->inv(hs)
 ans   =

                2
     1 + 3s + s
     ----------
          1

-->//inversion of state-space form
```

```
-->inv(Sys)
 ans  =


         ans(1)    (state-space system:)

!lss   A   B   C   D   X0   dt   !

         ans(2) = A matrix =

       []

         ans(3) = B matrix =

       []

         ans(4) = C matrix =

       []

         ans(5) = D matrix =

                  2
     1 + 3s + s

         ans(6) = X0 (initial state) =

       []

         ans(7) = Time domain =

 c

-->//converting this inverse to transfer representation

-->ss2tf(ans)
 ans  =

                  2
     1 + 3s + s
```

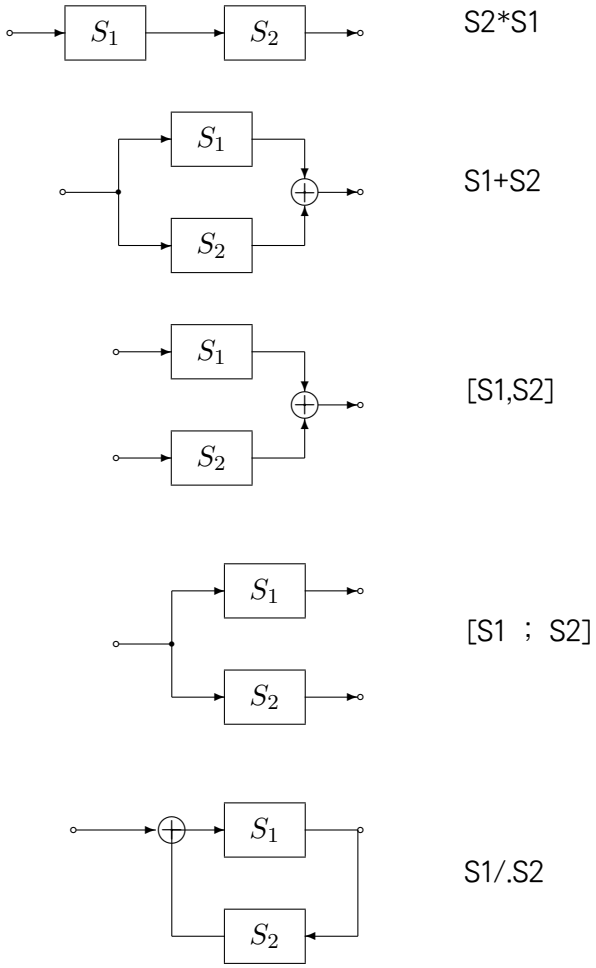


圖 4.1: 線性系統之聯結

串列讓線性系統可以表現的像一抽象物件。例如，線性系統可以和另一線性系統結合成一新系統，或利用 ss2tf 得到系統之轉函數。

線性系統一個很有用的觀念是能夠將線性系統當做一個資料物件般操作。線性系統可以如圖 4.1 連結。

兩系統 S1 和 S2間不同的連接方式所對應的指令可見圖 4.1 右邊指令。注意，回饋 (feedback) 連結的指令爲 S1/.S2。

線性系統可以使用狀態空間表示或使用轉換函數轉表示。這兩者可以使用函數 tf2ss 及 ss2tf 相互轉換。以下範例展示線性系統的產生、轉換、連結組合，

```
-->//system  connecting

-->s=poly(0,'s');

-->S1=1/(s-1)
 S1    =
```

```
      1
    -----
  - 1 + s

-->S2=1/(s-2)
 S2   =

      1
    -----
  - 2 + s

-->S1=syslin('c',S1);

-->S2=syslin('c',S2);

-->Gls=tf2ss(S2);

-->ssprint(Gls)

.
x = | 2 |x + | 1 |u

y = | 1 |x

-->hls=Gls*S1;

-->ssprint(hls)

.    | 2   1 |      | 0 |
x = | 0   1 |x + | 1 |u

y = | 1   0 |x

-->ht=ss2tf(hls)
 ht   =

         1
    ---------
              2
    2 - 3s + s
```

```
-->S2*S1
 ans  =

         1
    ---------
              2
    2 - 3s + s

-->S1+S2
 ans  =

    - 3 + 2s
    ----------
              2
    2 - 3s + s

-->[S1,S2]
 ans  =

!      1          1     !
!   -----      -----   !
! - 1 + s    - 2 + s   !

-->[S1;S2]
 ans  =

!      1     !
!   -----   !
! - 1 + s   !
!           !
!      1     !
!   -----   !
! - 2 + s   !

-->S1/.S2
 ans  =

    - 2 + s
    ---------
              2
```

```
    3 - 3s + s

-->S1./(2*S2)
 ans  =

  - 2 + s
    -----
  - 2 + 2s
```

以上段落稍微冗長些，但展示一些處理線性系統的重要事項。

首先，兩個線性系統利用函數 syslin 以設定系統轉換函數的方式產生。這兩系統在本例中設定成連續 (continuous) 系統。基礎函數 tf2ss 將其中一個線性系統轉爲等價的狀態函數表示法，在 Scilab 中以串列 (list) 資料來代表此線性系統 (注意，函數 ssprint 列印狀態空間資料時可讀性較高)。

接著將兩個系統進行乘法運算，這相當於製造一個以序列 (series) 結合的合成系統。注意這兩子系統，一個以轉移函數，一個以狀態空間方式代表系統，而合成結果以內定之狀態空間法表示。

最後，函數 ss2tf 將合成系統之狀態表示法轉爲轉換函數表示法。

4.10.函數 (巨集)

函數是指令的集合並在一新的計算環境執行，以隔離函數內變數和原始環境變數之間的干擾。函數可以用不同的方式產生及執行。函數能傳遞參數，具備程式控制結構如迴圈 (loop) 及條件判斷 (conditionals)，並允許遞迴呼叫 (recursively called)。函數也可以當作一般參數傳遞給另一函數，或作爲串列資料的一個成員。定義函數最常見的方式是在文書編輯器內製作，但也可以在 Scilab 環境中直接以語法 function 或 deff 定義新函數。

```
--> function [x]=foo(y)
-->     if y>0 then, x=1; else, x=-1; end
--> endfunction

--> deff('[x]=foo(y)','if y>0 then, x=1; else, x=-1; end')

--> foo(5)
 ans          =

    1.

--> foo(-3)
 ans          =

  - 1.
```

函數通常定義在檔案內，以編輯器編輯內容，並以指令
exec('filename')
載入 Scilab 環境。當然，也可以透過 Scilab 的交談式介面將檔案載入。
檔案 filename 的第一行必須如下：

function [y1,...,yn]=macname(x1,...,xk)

其中 y1,...,yn 爲輸出參數而 x1,...,xk 爲輸入參數。
有關函數進一步討論請參考 5.2。

4.11.程式庫 (Libraries)

Scilab 程式庫是函數的集合，在 Scilab 環境啓動時自動載入系統，但也可以選擇由使用者設定何時載入。程式庫是以指令 lib 產生。在 SCIDIR/macros 內有許多程式庫的實例。注意，在SCIDIR/macros 的次目錄內，檔案 “names” 內容爲此程式庫包含的函數名稱，檔案 *.sci 爲各函數的原始檔而檔案 *.bin 則爲各函數之編譯結果。系統建構檔 (Makefile–Unix, Makefile.mak–Windows) 呼叫 Scilab 將原始碼 *.sci 編譯成 *.bin 並呼叫目錄內之 genlib 程序以製作成程式庫。系統自動載入的程式庫是在系統啓動檔 SCIDIR/scilab.star 內進行設定。

4.12.物件 (Objects)

結束本章前，介紹函數 typeof 的傳回值以了解 Scilab 物件的各類型態。 Scilab 定義了以下型態：

- constant 實數或複數矩陣 (Scilab 中，矩陣包含純量，向量)
- polynomial 多項式矩陣
- boolean 布林矩陣
- string 字串矩陣
- function 函數
- rational 分式多項式矩陣 (syslin lists)
- state-space 線性系統狀態空間表示法
- sparse 稀疏矩陣
- boolean sparse 稀疏布林矩陣
- list 串列
- tlist 型態串列
- mlist 矩陣串列 (matrix oriented typed lists)
- library 程式庫

4.13.矩陣運算 Matrix Operations)

下表列出 Scilab 中基礎矩陣運算之語法：

SYMBOL	OPERATION
[ ]	矩陣定義、接合 (concatenation)
;	行分隔
( )	元素讀取 m=a(k)
( )	元素設定: a(k)=m
'	轉置
+	加
−	減
*	乘
\	左除 (left division)
/	右除 (right division)
^	指數
.*	逐元素 (elementwise) 乘法 multiplication
.\	逐元素左除
./	逐元素右除
.^	逐元素指數
.*.	kronecker 乘法
./.	kronecker 右除
.\.	kronecker 左除

4.14.指標 (Indexing)

以下範例介紹 Scilab 如何以指標標定內容，讀取及設定矩陣或串列內元素。詳細內容可利用 help extraction 或 help insertion 查詢。

4.14.1.矩陣指標 (Indexing in matrices)

矩陣元素可以利用行或列之指標，也可以利布林指標 (boolean indices) 及特殊符號 $ 以選取內容。

```
-->A=[1 2 3;4 5 6]
 A   =

!   1.    2.    3. !
!   4.    5.    6. !

-->A(1,2)
 ans   =

    2.
```

```
-->A([1 1],2)
 ans  =

!    2. !
!    2. !

-->A(:,1)
 ans  =

!    1. !
!    4. !

-->A(:,3:-1:1)
 ans  =

!    3.     2.     1. !
!    6.     5.     4. !

-->A(1)
 ans  =

     1.

-->A(6)
 ans  =

     6.

-->A(:)
 ans  =

!    1. !
!    4. !
!    2. !
!    5. !
!    3. !
!    6. !

-->A([%t %f %f %t])
 ans  =
```

```
!    1. !
!    5. !

-->A([%t  %f],[2  3])
 ans  =

!    2.       3. !

-->A(1:2,$-1)
 ans  =

!    2. !
!    5. !

-->A($:-1:1,2)
 ans  =

!    5. !
!    2. !

-->A($)
 ans  =

      6.

-->//

-->x='test'
 x  =

 test

-->x([1  1;1  1;1  1])
 ans  =

!test   test   !
!               !
!test   test   !
!               !
!test   test   !
```

```
-->//

-->B=[1/%s,(%s+1)/(%s-1)]
 B  =

!   1       1 + s   !
!   -       -----   !
!   s     - 1 + s   !

-->B(1,1)
 ans  =

    1
    -
    s

-->B(1,$)
 ans  =

    1 + s
    -----
  - 1 + s

-->B(2) // the numerator
 ans  =

!   1       1 + s   !

-->//

-->A=[1 2 3;4 5 6]
 A  =

!   1.      2.      3. !
!   4.      5.      6. !

-->A(1,2)=10
 A  =

!   1.      10.     3. !
```

```
!    4.      5.       6. !

-->A([1  1],2)=[-1;-2]
 A   =

!    1.   - 2.      3. !
!    4.      5.      6. !

-->A(:,1)=[8;5]
 A   =

!    8.   - 2.      3. !
!    5.      5.      6. !

-->A(1,3:-1:1)=[77  44  99]
 A   =

!    99.      44.      77. !
!    5.       5.       6.  !

-->A(1,:)=10
 A   =

!    10.      10.      10. !
!    5.       5.       6.  !

-->A(1)=%s
 A   =

!    s        10       10   !
!                           !
!    5        5        6    !

-->A(6)=%s+1
 A   =

!    s        10       10       !
!                               !
!    5        5        1 + s    !

-->A(:)=1:6
```

```
 A  =

!   1.      3.      5. !
!   2.      4.      6. !

-->A([%t  %f],1)=33
 A  =

!   33.     3.      5. !
!   2.      4.      6. !

-->A(1:2,$-1)=[2;4]
 A  =

!   33.     2.      5. !
!   2.      4.      6. !

-->A($:-1:1,1)=[8;7]
 A  =

!   7.      2.      5. !
!   8.      4.      6. !

-->A($)=123
 A  =

!   7.      2.      5.    !
!   8.      4.      123.  !

-->//

-->x='test'
 x  =

 test

-->x([4  5])=['4','5']
 x  =

!test            4   5   !
```

4.14.2.串列標定 (Indexing in lists)

本節展示如何利用指標得取或設定串列內元素。詳細內容可利用 help extraction 或 help insertion 查詢。

```
-->a=33;b=11;c=0;

-->l=list();l(0)=a
 l  =


          l(1)

     33.

-->l=list();l(1)=a
 l  =


          l(1)

     33.

-->l=list(a);l(2)=b
 l  =


          l(1)

     33.

          l(2)

     11.

-->l=list(a);l(0)=b
 l  =


          l(1)
```

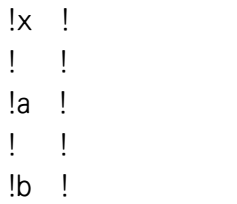
```
 11.

    l(2)

 33.

-->l=list(a);l(1)=c
 l  =


    l(1)

 0.

-->l=list();l(0)=null()
 l  =

   ()

-->l=list();l(1)=null()
 l  =

   ()

-->//

-->i='i';

-->l=list(a,list(c,b),i);l(1)=null()
 l  =


    l(1)


      l(1)(1)

 0.

      l(1)(2)
```

```
      11.

          l(2)

 i

-->l=list(a,list(c,list(a,c,b),b),'h');

-->l(2)(2)(3)=null()
 l  =


          l(1)

      33.

          l(2)


           l(2)(1)

      0.

          l(2)(2)


            l(2)(2)(1)

      33.

            l(2)(2)(2)

      0.

          l(2)(3)

      11.

          l(3)

 h
```

```
-->//

-->dts=list(1,tlist(['x';'a';'b'],10,[2  3]));

-->dts(2).a
 ans  =

      10.

-->dts(2).b(1,2)
 ans  =

      3.

-->[a,b]=dts(2)(['a','b'])
 b  =

!    2.       3.  !
 a  =

      10.

-->//

-->l=list(1,'qwerw',%s)
 l  =


          l(1)

      1.

          l(2)

 qwerw

          l(3)

      s
```

```
-->l(1)='Changed'
 l  =


          l(1)

 Changed

          l(2)

 qwerw

          l(3)

     s

-->l(0)='Added'
 l  =


          l(1)

 Added

          l(2)

 Changed

          l(3)

 qwerw

          l(4)

     s

-->l(6)=['one  more';'added']
 l  =


          l(1)
```

```
 Added

         l(2)

 Changed

         l(3)

 qwerw

         l(4)

      s

         l(5)

      Undefined

         l(6)

!one  more   !
!              !
!added       !

-->//

-->dts=list(1,tlist(['x';'a';'b'],10,[2  3]));

-->dts(2).a=33
 dts   =


         dts(1)

      1.

         dts(2)


           dts(2)(1)
```



dts(2)(2)

33.

dts(2)(3)

! 2. 3. !

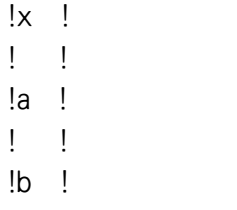
-->dts(2).b(1,2)=-100
dts =

dts(1)

1.

dts(2)

dts(2)(1)



dts(2)(2)

33.

dts(2)(3)

! 2. - 100. !

```
-->//

-->l=list(1,'qwerw',%s);

-->l(1)
 ans  =

    1.

-->[a,b]=l([3  2])
 b  =

 qwerw
 a  =

    s

-->l($)
 ans  =

    s

-->//

-->L=list(33,list(l,33))
 L  =


        L(1)

    33.

        L(2)


         L(2)(1)


           L(2)(1)(1)
```

1.

L(2)(1)(2)

qwerw

L(2)(1)(3)

s

L(2)(2)

33.

第五章

語言結構及函數

內容

Scilab 其中一個有用的功能是能夠以高階語法自訂及使用函數。這讓特定領域問題能夠很輕易的透過諸如 Scilab 程式庫的功能，整合到系統中。本章將討論以下主題：

- Scilab 語言控制結構
- 函數定義及使用
- 定義新資料型態之運算函數
- 偵錯

產生、製作 Scilab 程式庫將再下一章中討論。

5.1. Scilab 語言控制結構

Scilab 支持完整的程式語言控制結構如：迴圈 (loops)、條件式 (conditionals)、 case 選擇及產生新環境等。大部分程式 Scilab 語言之應用都是在函數之內的局部環境中進行。

5.1.1.比較運算子

Scilab 資料物件的以下五種比較運算子：

==	等於
<	小於
>	大於
<=	小於等於
>=	大於等於
<> or ~=	不等於

這些比較算子適用於條件判斷式中。

5.1.2.迴圈

Scilab 語言有兩種迴圈： for 迴圈及 while 迴圈。 for 迴圈將一指標向量逐項指定，每一指標設定後，計算至保留字 end 後再進行下一指標計算。

```
--> x=1;for k=1:4,x=x*k,end
 x           =

    1.
 x           =

    2.
 x           =

    6.
 x           =

    24.
```

for 迴圈可以用向量的元素或矩陣之列向量元素作爲迭代之資料。

```
--> x=1;for k=[-1 3 0],x=x+k,end
 x           =

    0.
 x           =

    3.
 x           =

    3.
```

for 迴圈也可以用串列元素作爲迭代資料，語法與使用矩陣或向量相同。

```
-->l=list(1,[1,2;3,4],'str')

-->for k=l, disp(k),end

      1.

!    1.      2. !
!    3.      4. !

 str
```

while 迴圈不斷重複進行一序列之指令，直到某一條件不再滿足爲止。

```
--> x=1; while x<14,x=2*x,end
 x            =

      2.
 x            =

      4.
 x            =

      8.
 x            =

      16.
```

for 或 while 迴圈可以用指令 break 強迫終止：

```
-->a=0;for i=1:5:100,a=a+1;if i > 10  then  break,end; end

-->a
 a  =

      3.
```

在巢狀迴圈 (nested loops) 中 break 指令從最內層迴圈中結束計算。

```
-->for k=1:3; for j=1:4; if k+j>4 then break;else disp(k);end;end;end
```

1.

1.

1.

2.

2.

3.

5.1.3.條件式

Scilab 語言有兩種條件判斷式： if-then-else 判斷式及select-case 判斷式。 if-then-else 判斷式先計算布林運算，若爲眞值則計算 then 和 else (或 end) 指令之間的運算。若布林運算之結果爲假值，則計算 else 和 end 指令之間的運算。 else 指令並非必要。 elseif 與其他程式語言意義相同，因此也是 Scilab 語言可以使用之保留字。

```
--> x=1
 x           =

    1.

--> if  x>0  then,y=-x,else,y=x,end
 y           =

   - 1.

--> x=-1
 x           =

   - 1.

--> if  x>0  then,y=-x,else,y=x,end
 y           =

   - 1.
```

select-case 判斷式比對幾組運算，並選擇第一個和原始給定算結果相符的計算區塊。

```
--> x=-1
```

```
 x             =

  - 1.

--> select  x,case  1,y=x+5,case  -1,y=sqrt(x),end
 y             =

     i
```

也可以加上 else 指令，以容許與其它運算皆不相符的狀況。

5.2.函數定義及使用

Scilab 環境中可以直接定義函數，但最方便的方式還是以編輯器將函數寫在獨立的檔案內。本節介紹函數的結構以及幾個和函數定義相關的 Scilab 指令。

5.2.1.函數結構

函數之結構必須滿足以下格式：

```
function  [y1,...,yn]=foo(x1,...,xm)
    .
    .
    .
```

其中 foo 爲函數名稱，而 xi 代表的第 i 個輸入參數。，yj 則爲第 j 個輸出參數。輸入及輸出參數可以是任何 Scilab 物件 (包含函數本身)。以下爲一個計算階乘 ($k!$) 的函數的範例：

```
function  [x]=fact(k)
  k=int(k)
  if  k<1  then  k=1,end
  x=1;
  for  j=1:k,x=x*j;end
endfunction
```

此函數存在檔案 fact.sci 內，指令exec 或 getf 能將檔案內之函數載入 Scilab 環境中：

```
--> exists('fact')
 ans          =

     0.

--> exec('../macros/fact.sci',-1);
```

```
--> exists('fact')
 ans          =

     1.

--> x=fact(5)
 x            =

     120.
```

以上範例指令 exists 測試函數是否已載入系統中，而 exec 指令將檔案載入系統。指令 x=fact(5) 則開始使用此新添入之階乘函數, 並將 5! 之結果存在變數 x 中。

5.2.2.載入函數

函數也是一個 Scilab 物件，因此不要將他們視爲一檔案。爲了能夠在 Scilab 環境中使用，這些函數必須透過如 getf(filename) 或 exec(filename,−1) ；之指令將函數載入系統。函數 foo 只有在內含此函數的檔案已透過 getf(filename) 或 exec(filename,−1) ；指令載入後才能執行。

一個檔案可以包含數個函數。函數也可以線上的以交談式方式在 Scilab 環境中利用 function/endfunction 語法輸入。函數 deff 也可用在動態產生新函數之用。

一組函數可以組合成一個程式庫 (參考指令 lib)。 Scilab 標準程式庫 (linear algebra, control,...) 定義在 SCIDIR/macros/內之各次目錄中。

5.2.3.全域及局部變數

函數內的變數如果未在函數內定義 (也未出現在輸入、輸出參數內)，則此變數視爲呼叫此函數的計算環境中之變數。

在函數定義之變數視爲局部變數，也就是改變其值不影響外部環境之同名變數，除非使用了resume 指令。

函數可以使用較預定爲少之輸入輸出參數。例如：

```
function [y1,y2]=f(x1,x2)
  y1=x1+x2
  y2=x1-x2
endfunction

-->[y1,y2]=f(1,1)
 y2  =
     0.
 y1  =
     2.

-->f(1,1)
 ans  =
```

```
    2.

-->f(1)
y1=x1+x2;
          !--error       4
undefined variable : x2
at line          2  of  function  f

-->x2=1;

-->[y1,y2]=f(1)
 y2   =
    0.
 y1   =
    2.

-->f(1)
 ans   =

    2.
```

注意，只有當呼叫環境提供所缺少的變數時，才能成功的以較少參數呼叫函數。 a

```
function  [y]=f(x1,x2)
  if  x1<0  then  y=x1,  else  y=x2;end
endfunction


-->f(-1)
 ans   =

  - 1.

-->f(-1,x2)
         !--error       4
undefined variable : x2

-->f(1)
 undefined variable : x2
at line          2  of  function      f        called  by :
f(1)
```

```
-->x2=3;f(1)

-->f(1)
 ans  =

    3
```

全域變數以 global 指令定義。全域變數能夠在各函數內部進行讀寫變數內容之動作。詳細內容請使用指令 help global 查詢。

5.2.4.特殊函數指令

以下特殊指令在定義函數時經常使用：

- argn：傳回輸入及輸出參數個數。
- error：錯誤產生時暫停計算、列印錯誤訊息、並返回到上一層呼叫環境。
- warning 列印警告訊息。
- pause：暫停計算。
- break：強迫結束一迴圈。
- return 或 resume ：返回上層呼叫環境，並將局部變數由函數內環境轉移至呼叫環境。

下列範例執行函數 foo 藉以了解上述指令用途。

- 定義函數

```
function [z]=foo(x,y)
[out,in]=argn(0);
if x==0 then,
      error('division by zero');
end,
slope=y/x;
pause,
z=sqrt(slope);
s=resume(slope);
endfunction
```

- 使用函數

```
--> z=foo(0,1)
error('division by zero');
                                !--error 10000
division by zero
at line       4 of function     foo       called by :
 z=foo(0,1)


--> z=foo(2,1)

-1-> resume
 z           =

    0.7071068

--> s
 s           =

    0.5
```

此例，第一次呼叫 foo 時傳入一個無法計算的參數，函數終止計算並將錯誤訊息傳給使用者。第二次呼叫 foo 在計算 slope 之後暫停計算。此時，使用者可以檢查函數內之參數內容、繪圖或其他任何 Scialb 允許的指令。螢幕反應訊息 -1-> 代表由 pause 指令所產生的計算環境而非呼叫函數前之計算環境。在反應訊息爲 -1-> 時，指令 return 能返回函數局部環境內。指令 quit 或 abort 可以結束函數之操作。最後，函數將變數 z 傳回給呼叫環境。同時局部變數 s 也透過 resume 傳給全域計算環境。

5.3.定義新資料型態之運算函數

在 Scilab 中可以爲個別的資料型態提供一致性的基本運算符號 (請以指令 help overloading 查詢)。也就說，對任意資料型態的兩物件，可以提供相同的符號以代表諸如加、減、乘、除等運算。例如：兩 Scilab 之線性系統，可以用加法運算代表平行連結，而用乘法運算代表序列連結。需要進行這些運算時， Scilab 搜索特定格式的函數名稱並進行呼叫。因此對 Scilab 系統而言，運算子和函數是相同的。

開發者必須依循特定習慣爲函數命名。這類運算函數的名稱由 3 或 4 欄組成。第一欄永遠爲符號 %。第三欄由下表決定，代表運算子之類型。

第三欄	
符號	運算
a	+ ，加法
b	: ，範圍 (range generator)
c	[a,b] ，列連結 (column concatenation)
d	./ 逐元素除法
e	() ，讀取 (extraction) : m=a(k)
f	[a;b] ，行連結 (row concatenation)
g	| ，邏輯或 (logical or)
h	& ，邏輯且 (logical and)
i	() ，寫入 (insertion) : a(k)=m
j	.^ ，逐元素指數 (element wise exponent)
k	.*.
l	\ ，左除 (left division)
m	*
n	<> ，不等於
o	== ，等於
p	^ ，指數
q	.\
r	/ ，右除 (right division)
s	-
t	' ，轉置 (transpose)
u	*.
v	/.
w	\.
x	.*
y	./.
z	.\.
0	.'
1	<
2	>
3	<=
4	>=
5	~ ，非 (not)

運算函數名稱的第二及第四欄分別代表第一及第二運算元的資料型態。可以使用的資料型態及其代表符號可參考下表：

第二及第四欄	
符號	變數型態
s	純量 (scalar)
p	多項式 (polynomial)
l	串列 (list)
c	字串 (character string)
b	布林 (boolean)
sp	稀疏矩陣 (sparse)
spb	布林稀疏矩陣 (boolean sparse)
m	函數 (function)
xxx	型態串列 (typed list)

Scilab 的型態串列 (typed list) 能夠藉由串列，定義一新型態。型態串列第一項爲字串，代表此一形態之名稱。上表型態串列對應的符號 xxx 及代表此自訂之名稱。例如代表線性系統之型態串列爲：

tlist(['lss','A','B','C','D','X0','dt'],a,b,c,d,x0,'c').

因此上表中，符號 xxx 在此例就是 lss。

兩線性系統之乘法運算 (序列連結) 之函數名稱依照上表爲 %lss_m_lss。名稱第一欄爲% 代表運算函數, 第二及第四欄爲 lss 代表線性系統 (linear state-space), 第三欄爲 m 表乘法運算。以下爲一個實作範例：

```
function [s]=%lss_m_lss(s1,s2)
[A1,B1,C1,D1,x1,dom1]=s1(2:7),
[A2,B2,C2,D2,x2]=s2(2:6),
B1C2=B1*C2,
s=lsslist([A1,B1C2;0*B1C2' ,A2],...
        [B1*D2;B2],[C1,D1*C2],D1*D2,[x1;x2],dom1),
endfunction
```

運算函數經定義並載入系統後，可以如下使用新定義之乘法運算子 *：

```
-->A1=[1 2;3 4];B1=[1;1];C1=[0 1;1 0];

-->A2=[1 -1;0 1];B2=[1 0;2 1];C2=[1 1];D2=[1,1];

-->s1=syslin('c',A1,B1,C1);

-->s2=syslin('c',A2,B2,C2,D2);
```

```
-->ssprint(s1)

.    | 1   2 |      | 1 |
x  = | 3   4 |x  + | 1 |u

     | 0   1 |
y  = | 1   0 |x

-->ssprint(s2)

.    | 1 -1 |      | 1   0 |
x  = | 0   1 |x  + | 2   1 |u

y  = | 1   1 |x  + | 1   1 |u

-->s12=s1*s2;    //This is equivalent to s12=%lss_m_lss(s1,s2)

-->ssprint(s12)

     | 1   2   1   1 |      | 1   1 |
.    | 3   4   1   1 |      | 1   1 |
x  = | 0   0   1  -1 |x  + | 1   0 |u
     | 0   0   0   1 |      | 2   1 |

     | 0   1   0   0 |
y  = | 1   0   0   0 |x
```

注意函數 %lss_m_lss 並未直接被使用者呼叫。而是在兩線性系統進行乘法運算 (*) 時系統自動呼叫。

目錄 SCIDIR/macros/percent 內含所有這類用於 Scilab 之標準運算函數。

5.4.偵錯

要對 Scilab 函數進行偵錯的最簡單方式是在函數中加入 pause 指令。當函數執行盪此指令時，計算環境會由函數局部環境再往下產生一層，而螢幕應對訊息變成 -1-> ，若在此環境在輸入 pause 指令則螢幕應對訊息變成 -2-> 。在這些下層計算環境中，包含函數環境在內的所有上層計算環境中的變數皆可列印或操作。

要結束偵錯可以使用 return 或 resume 指令將計算環境回覆到上一層。

也可以在函數中插入中斷點 (breakpoints) 以方便進行偵錯，請參考指令 setbpt, delbpt, disbpt。

錯誤訊息也可以設計成自動捕捉 (to trap errors)，請參考指令 errclear 及errcatch。

最後也可以在 Scilab 設定偵錯層次如： debug(i) 其中 i=0,..,4 代表偵錯層次 (debugging level)。

第六章

繪圖

內容

6.1.圖形視窗

初習者在 Scilab 環境輸入指令 plot 即能繪出關 plot 指令的一張範例圖。因此本章所有相關繪圖指令，皆可以無參數的方式嚐試繪出範例，以輔助學習。

再內定狀態下，繪圖之輸出裝置是螢幕。 Scilab 的螢幕繪圖軟體還進一步提供交談式介面供各項圖形或輸出設定。有關圖形交談式介面之使用方法，建議使用者自行多做嘗試，即可知道用法。本章主要討論以程式指令方式控制圖形的方法。

6.2.輸出媒體

Scilab 提供幾個繪圖設施以供輸出繪圖指令到螢幕或紙張。內定之輸出是 Scilab Graphic (0) 視窗。
Scilab 所提供的繪圖設施 (drivers) 是：

- X11 : X11 (Unix/Linux 環境) 視窗
- Rec :螢幕設施，此設施也記錄所有繪圖指令，爲內定設施
- Wdp :螢幕設施，不記錄繪圖指令，圖形直接會在影像檔中。圖形以指令 xset("wshow") 送到圖形視窗，並以指令 xset("wwpc") 或 xbasc() 清除影像
- Pos : Postscript 圖檔之輸出設施
- Fig : Xfig 圖檔之輸出設施
- GIF : GIF 圖檔之輸出設施

前三種爲視窗輸出，後三種爲檔案輸出。

基本的 Scilab 繪圖指令如下：

- driver:選擇一個繪圖設施

  以下指令只對螢幕繪圖設施有效：
- xclear:清空圖形視窗，但不影響視窗之繪圖內文 (graphics context) 設定
- xbasc:清空圖形視窗，清除繪圖紀錄，但不影響視窗之繪圖內文 (graphics context) 設定
- xpause:以毫秒 (milliseconds) 爲單位暫停繪圖
- xselect:將目前繪圖視窗提到幕前
- xclick:等待滑鼠點擊 (click)
- xbasr:重繪視窗內圖形
- xdel:刪除一圖形視窗 (等效於 Close 按鍵)

  以下指令只對檔案圖形設施 (Pos,Fig,Gif) 有效：
- xinit:初始一圖形設施 (檔案)。
- xend:結束一繪圖設施。

事實上，最常使用的圖形設施是 Rec。在 Scilab 中有些指令可以防止不當改變圖形設施；有時候使用者也可以直接以 xbasimp, xs2fig 指令將圖形送到特定檔案。例如：

```
-->driver('Pos')

-->xinit('foo.ps')

-->plot(1:10)

-->xend()

-->driver('Rec')

-->plot(1:10)

-->xbasimp(0,'foo1.ps')
```

我們得到兩張完全相同的 Postscript 檔: 'foo.ps' 及 'foo1.ps.0' (檔案後添加 0 代表繪圖來源的視窗編號).

內定的繪圖是重疊模式，也就是若要繪出不同的圖形必須利用 xbasc(window-number) 或 xclear 先清除前一張圖形。

如果將圖形放大， Scilab 自動執行指令 xbasr(window-number) 以重繪圖形。必要時，使用者也可以自行呼叫此指令。

指令 xset 、 xselect 或 xinit 可以新開圖形視窗，數目並無限制。

6.3.圖形之全域參數

6.3.1.圖形內文 (Graphics Context)

部分圖形參數是由圖形內文 (graphic context) 所控制 (例如線寬) 另外一些則由繪圖指令之參數控制。

圖形內文都有內定值，但可以透過指令 xset 變更內容：若不輸入參數，也就是輸入 xset() 時，將會開啓一對話視窗方便改變圖形內文。以下則是變更圖形內文的指令格式：

- xset :設定圖形內文

  (i)-xset("font",fontid,fontsize) :設定字型

  (ii)-xset("mark",markid,marksize) :設定繪圖符號 (mark)

  (iii)-xset("use color",flag) :選擇彩色或灰階繪圖，flag 爲 (1 或 0)

  (iv)-xset("colormap",cmap) :設定色彩對應 (colormap)， cmap 爲 m x 3 矩陣。 m 爲色彩數目， cmap[i,1],cmap[i,2], cmap[i,3] 分別代表第 i 顏色之 RGB 數 (0 到 1 之間)。

  (v)-xset("window",window-number) :將視窗 window-number 設爲目前工作視窗

(vi)-xset("wpos",x,y) :設定視窗之左上角座標

xset 指令的其他功能爲:

-使用影像區 (pixmap) :圖形可以直接顯示到螢幕，或先繪到影像區 (pixmap) 再利用 xset("wshow") 指令將影像展示到螢幕；這是動畫顯像的常用方法。

-影像之邏輯函數:透過參數 alufunction (以 help alufunction 查詢) 能夠在繪圖時在影像間進行邏輯運算，一些特殊效果可以透過此參數設定達成。以下爲應用範例：

```
xset('default');
plot3d();
plot3d();
xset('alufunction',7);
xset('window',0);
plot3d();
xset('default');
plot3d();
xset('alufunction',6);
xset('window',0);
plot3d();
```

有關字型的另一個指令爲：

- xlfont :載入一族新字型 (family of fonts)

  xset 指令的接收端爲:

- xget :取得圖形內文資訊

  所有能夠被 xset 設定的參數皆可透過 xget 取得。例：

```
-->pos=xget("wpos")
 pos    =

!    105.       121.  !
```

其中 pos 代表圖形視窗的左上角座標。

6.3.2.圖形操作 (Manipulations)

座標轉換

- isoview :等量 (isometric scale) 變換，不改變視窗尺度。

```
t=(0:0.1:2*%pi)';
plot2d(sin(t),cos(t));
xbasc()
isoview(-1,1,-1,1);
plot2d(sin(t),cos(t));
```

- square :等量 (isometric scale) 變換，但改變視窗尺度，以參數控制尺度
- scaling :改變數據尺度
- rotate :旋轉

  scaling 及 rotate 將一代表 (x,y) 座標點集合的執行彷射轉換 (affine transform)
- xgetech, xsetech :改變視窗內之繪圖尺度

  繪圖尺度一般是由高階繪圖指令決定，但也可以直接使用 xgetech, xsetech 改變。xsetech 指令可以將目前視窗內切出一塊局部繪圖，例：

```
t=(0:0.1:2*%pi)';
xsetech(wrect=[0.,0.,0.6,0.3],frect=[-1,1,-1,1]);
plot2d(sin(t),cos(t));
xsetech(wrect=[0.5,0.3,0.4,0.6],frect=[-1,1,-1,1]);
plot2d(sin(t),cos(t));
```

6.4.2D 繪圖

6.4.1.基礎 2D 繪圖

最簡單的繪圖指令是 plot(x,y) 或 plot(y) ：其中 x and y 爲兩向量。如果 x 省略，代表 x 爲 (1,size(y,'*'))。如果 y 是矩陣，將繪出矩陣之所有行向量。plot(x,y) 還有其他選擇性參數。

第一個繪圖範例如下，其中一張結果爲圖 6.1：

```
t=(0:0.05:1)';
ct=cos(2*%pi*t);
// plot the cosine
plot(t,ct);
// xset() opens the toggle panel and
// some parameters can be changed with mouse clicks
// given by commands for the demo here
xset("font",5,4);xset("thickness",3);
```

```
// plot with captions for the axis and a title for the plot
// if a caption is empty the argument ' ' is needed
plot(t,ct,'Time','Cosine','Simple Plot');
// click on a color of the xset toggle panel and do the previous plot again
// to get the title in the chosen color
```

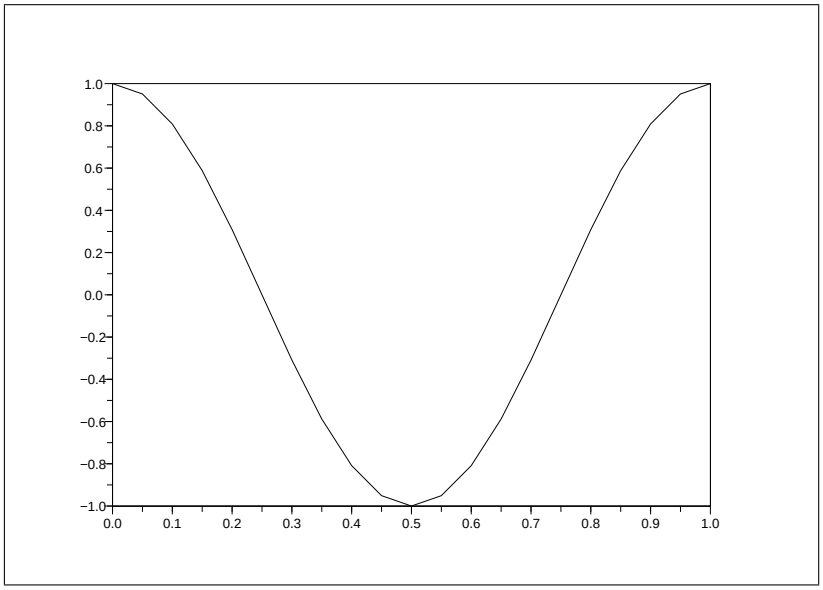


圖 6.1: First example of plotting

最一般化之 2D 多圖指令爲
plot2di(x,y,<options>)

- plot2d 後之指標: i=無,2,3,4.
  依照指標 i 之不同有如下變化：
  i=無 :局部 (piecewise) 線性/對數 (linear/logarithmic) 繪圖
  i=2 :局部常數繪圖
  i=3 :垂直方塊 (vertical bars)
  i=4 :箭號格式 (arrows style)，例如常微分方程的相空間 (phase space)

```
t=(1:0.1:8)';xset("font",2,3);
subplot(2,2,1)
plot2d([t t],[1.5+0.2*sin(t) 2+cos(t)]);
xtitle('Plot2d-Piecewise linear');
//
subplot(2,2,2)
plot2d(t,[1.5+0.2*sin(t) 2+cos(t)],logflag='ll');
```

```
xtitle('Plot2d1-Logarithmic scales');
//
subplot(2,2,3)
plot2d2(t,[1.5+0.2*sin(t) 2+cos(t)]);
xtitle('Plot2d2-Piecewise constant');
//
subplot(2,2,4)
plot2d3(t,[1.5+0.2*sin(t) 2+cos(t)]);
xtitle('Plot2d3-Vertical bar plot')
```

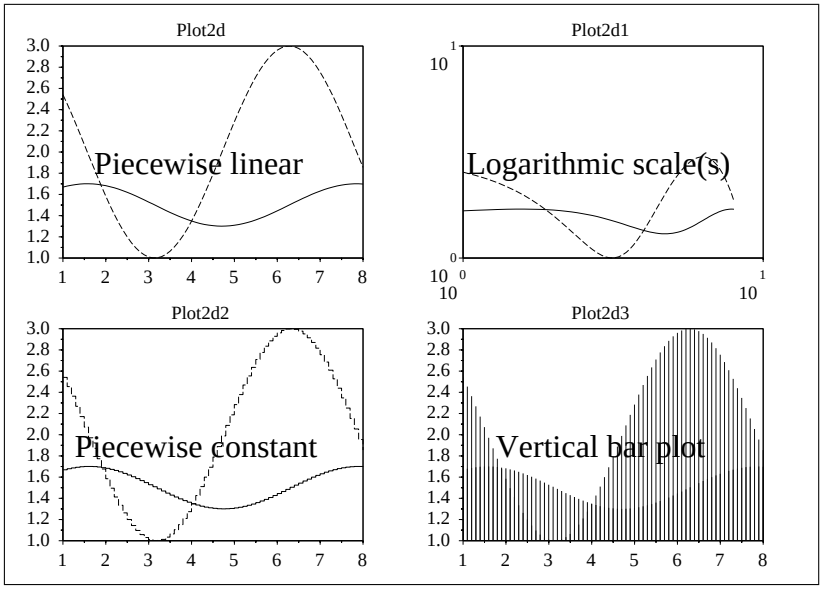


圖 6.2: Different 2D plotting

- 參數 x,y :兩個代表曲線座標大小爲 [nl,nc] 的矩陣，nc 是曲線之數目， nl 是每一曲線上的點數。

  如果爲單一曲線可以使用行向量或列向量，例如：
  plot2d(t',cos(t)') plot2d(t,cos(t)) 兩指令等效。

- 選項參數 style :長度爲 (1,nc) 的實數向量，第 j 條曲線之符號型式爲 style(j)

  ```
  xmax=5.;x=0:0.1:xmax;
  u=[-0.8+sin(x);-0.6+sin(x);-0.4+sin(x);-0.2+sin(x);sin(x)];
  u=[u;0.2+sin(x);0.4+sin(x);0.6+sin(x);0.8+sin(x)]';
  //start trying the symbols (negative values for the style)

  plot2d(x,u,style=[-9,-8,-7,-6,-5,-4,-3,-2,-1,0])
  ```

```
x=0:0.2:xmax;
v=[1.4+sin(x);1.8+sin(x)]';
xset("mark  size",5);
plot2d(x,v,style=[-7,-8])
xset('default');
```

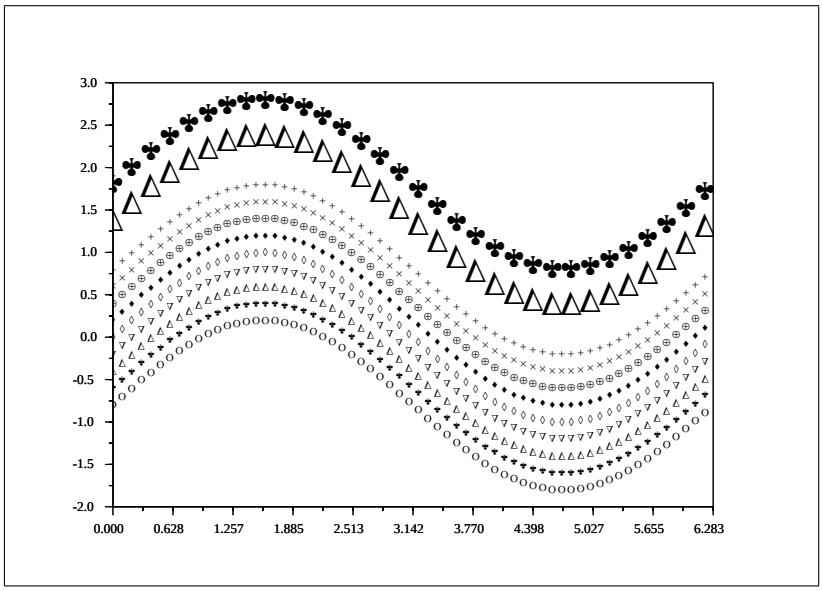


圖 6.3: Black and white plotting styles

- 選項參數 frameflag :爲對應到下表之純量：

  當實際座標範圍超出原先給定的範圍時，利用此參數決定如何繪圖，請參考 plot2d 指令說明

需求 實際	範圍 由前一張圖 決定	範圍 由參數 決定	範圍 由參數 x， y 計算
依照 需求	0	1	2
由等量 (isometric) 尺度計算 範圍		3	4
以各軸 列印美觀 角度計算		5	6
前圖重繪 與現圖 合成一起		7	8

- 選項參數 axesflag 控制座標軸之繪製：

- ▷ 0 :不繪出座標軸
- ▷ 1 :繪出座標軸，左邊顯現 y=axis
- ▷ 2 :以外框圍起圖形，無座標短線 (ticks)
- ▷ 3 :繪出座標軸，右邊顯現 y=axis
- ▷ 4 :座標軸繪在外框範圍之正中間
- ▷ 5 :座標軸繪在 (0,0) 點

- 選項參數 leg ：此參數爲一字串代表各曲線之標題字串。各標題字串間以符號 @ 間隔：例如 ``module@phase''
- 選項參數 rect ：rect=[xmin,ymin,xmax,ymax] 爲一向量值代表圖形之邊界座標極限值。
- 選項參數 nax :四個數值之向量 [nx,Nx,ny,Ny] 用來代表 x, y 軸上之主短線 (ticks) 及小短線。 Nx 代表 x 軸上的主短線數目，而 nx 代表在主短線間的小短線數目。 Ny, ny 在 y 軸代表相同意義。

```
//captions for identifying the curves
//controlling the boundaries of the plot and the tics on axes
x=-5:0.1:5;
y1=sin(x);y2=cos(x);
X=[x;x]; Y=[y1;y2];
plot2d(X',Y',style=[-1 -3]',leg="caption1@caption2",...
rect=[-5,-1,5,1],nax=[2,10,5,5]);
```

6.4.2.文字標示及圖形顯示

- xgrid ：加上 2D 格線，參數爲色彩編號
- xtitle ：加上圖標題，及各軸名稱
- titlepage :在圖之中央加上標題

```
//Presentation
x=-%pi:0.1:%pi;
y1=sin(x);y2=cos(x);y3=x;
X=[x;x;x]; Y=[y1;y2;y3];
plot2d(X',Y',style=[-1 -2 -3],leg="caption1@caption2@caption3",...
rect=[-3,-3,3,2],nax=[2,10,2,5]);
xtitle(["General Title";"(with xtitle command)"],...
```

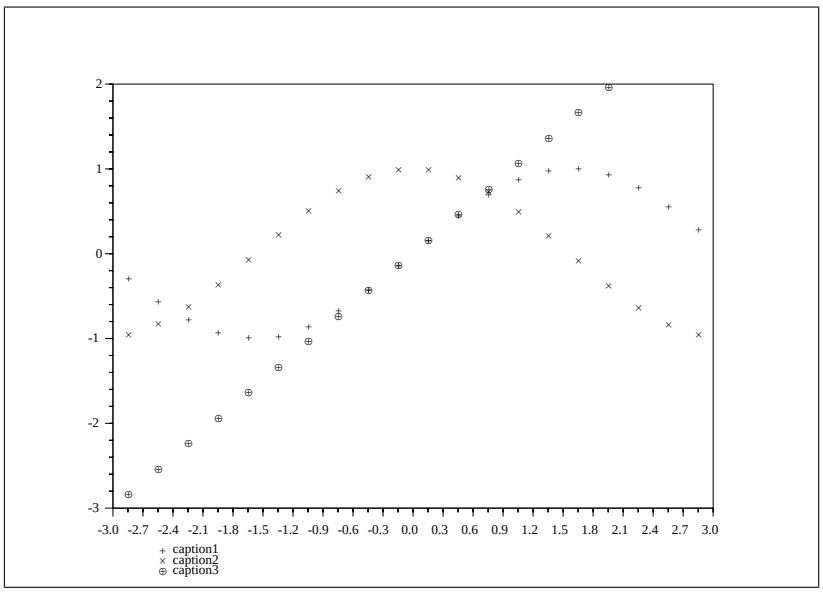


圖 6.4: Box, captions and tics

```
"x-axis title","y-axis title (with xtitle command)");
xgrid();
xclea(-2.7,1.5,1.5,1.5);
titlepage("Titlepage");
xstring(0.6,.45,"(with titlepage command)");
xstring(0.05,.7,["xstring command after";"xclea command"],0,1);
plot2d(X',Y',style=[-1 -2 -3],leg="caption1@caption2@caption3",...
rect=[-3,-3,3,2],nax=[2,10,2,5]);
```

- plotframe :加上外框及格線

座標短線、繪圖大小、外框及格線等可以設定一次而用於接續之圖形中。以下之簡單指令即能繪出 2D 繪圖座標標示的一些必要功能。

```
rect=[-%pi,-1,%pi,1];
tics=[2,10,4,10];
plotframe(rect,tics,[%t,%t],...
['Plot with grids and automatic bounds','angle','velocity']);
```

- graduate : a simple tool for computing pretty axis graduations before a plot.

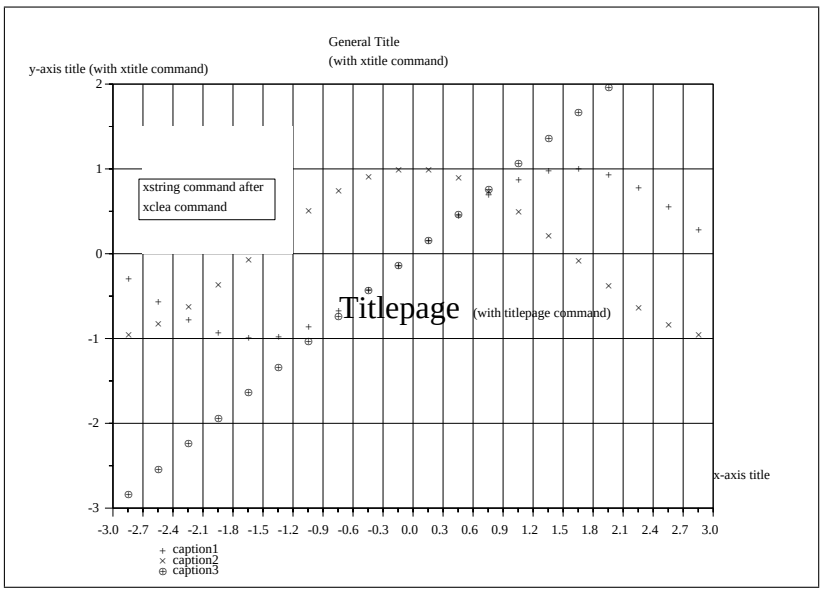


圖 6.5: Grid, Title eraser and comments

6.4.3.特殊 2D 繪圖

- champ :

```
//try champ
x=[-1:0.1:1];y=x;u=ones(x);
fx=x.*.u';fy=u.*.y';
champ(x,y,fx,fy);
xset("font",2,3);
xtitle(['Vector field plot';'(with champ command)']);
//with the color (and a large stacksize)
x=[-1:0.004:1];y=x;u=ones(x);
fx=x.*.u';fy=u.*.y';
champ1(x,y,fx,fy);
```

- fchamp : R^2 空間的向量場繪圖，座標由函數定義
- fplot2d :以函數繪出 2D 曲線
- grayplot :以 2D 灰階程度繪出一曲面，曲面座標以矩陣數值代表格點座標
- fgrayplot :以 2D 灰階程度繪出一曲面，曲面座標以函數計算座標值
  事實上，以上兩函數可以使用色彩對應 (colormap) 以彩色繪圖。

```
R=[1:256]/256;RGB=[R' R' R'];
xset('colormap',RGB);
```

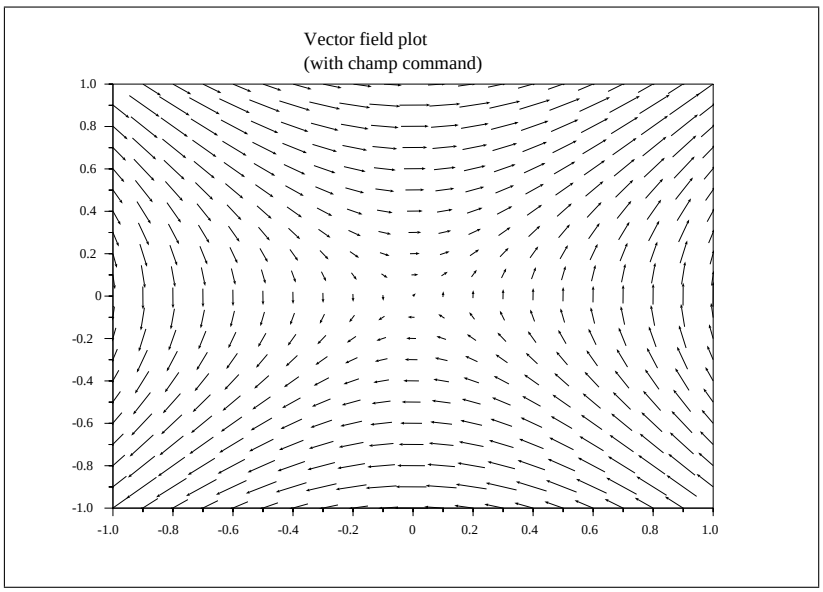


圖 6.6: Vector field in the plane

```
deff('[z]=surf(x,y)','z=-((abs(x)-1)**2+(abs(y)-1)**2)');
fgrayplot(-1.8:0.02:1.8,-1.8:0.02:1.8,surf,rect=[-2,-2,2,2]);
xset('font',2,3);
xtitle(["Grayplot";"(with fgrayplot command)"]);
//the same plot can be done with a ``unique'' given color
R=[1:256]/256;
G=0.1*ones(R);
RGB=[R' G' G'];
xset('colormap',RGB);
fgrayplot(-1.8:0.02:1.8,-1.8:0.02:1.8,surf,rect=[-2,-2,2,2]);
```

- errbar :在圖形上加繪誤差帶 (error bars)

6.4.4.特定幾何之繪圖

6.4.4.1.多邊形繪圖

- xsegs :不相連線段 (segments) 集合之繪線
- xrect :矩形繪線
- xfrect :矩形填色
- xrects :矩形集合之填色
- xpoly :多邊形繪線
- xpolys :多邊形集合填色

- xfpoly :多邊形填色
- xfpolys :多邊形集合填色
- xarrows :不相連箭線 (arrows) 集合之繪線
- xclea :將視窗內一矩形範圍內抹空 (erases)

6.4.4.2.曲線繪圖

- xarc :橢圓繪線
- xfarc :橢圓填色
- xarcs :對一橢圓及合繪線或填色

6.4.5.繪出字串

- xstring :繪出字串或字串矩陣
- xstringl :計算包含字串所需的方框
- xstringb :在一方框內繪出字串
- xnumb :繪出數字集合

比較以下範例及繪出之結果 (圖 6.7)，對有助於理解各指令用法。

```
//          initialize default environment variables
xset('default');
xset("use color",0);
xset("font",4,3)
xsetech(frect=[1,1,10,10]);
xrect(0,1,3,1)
xfrect(3.1,1,3,1)
xstring(0.5,0.5,"xrect(0,1,3,1)")
xstring(4.,0.5,"xfrect(3.1,1,3,1)")
xset("alufunction",6)
xstring(4.,0.5,"xfrect(3.1,1,3,1)")
xset("alufunction",3)
xv=[0 1 2 3 4]
yv=[2.5 1.5 1.8 1.3 2.5]
xpoly(xv,yv,"lines",1)
xstring(0.5,2.,"xpoly(xv,yv,""lines"",1)")
xa=[5 6 6 7 7 8 8 9 9 5]
ya=[2.5 1.5 1.5 1.8 1.8 1.3 1.3 2.5 2.5 2.5]
xarrows(xa,ya)
```

```
xstring(5.5,2.,"xarrows(xa,ya)")
xarc(0.,5.,4.,2.,0.,64*300.)
xstring(0.5,4,"xarc(0.,5.,4.,2.,0.,64*300.)")
xfarc(5.,5.,4.,2.,0.,64*360.)
//xset("alufunction",6)
xclea(5.6,4.4,2.8,0.8);
xstring(5.8,4.,"xfarc  and  then  xclea")
//xset("alufunction",3)
xstring(0.,4.5,"WRITING-BY-XSTRING()",-22.5)
xnumb([5.5  6.2  6.9],[5.5  5.5  5.5],[3  14  15],1)
isoview(0,12,0,12)
xarc(-5.,12.,5.,5.,0.,64*360.)
xstring(-4.5,9.25,"isoview  +  xarc",0.)
A=["  1" "          2" "  3";"  4" "          5" "  6";"68" "  17.2" "  9"];
xstring(7.,10.,A);
rect=xstringl(7,10,A);
xrect(rect(1),rect(2),rect(3),rect(4));
```

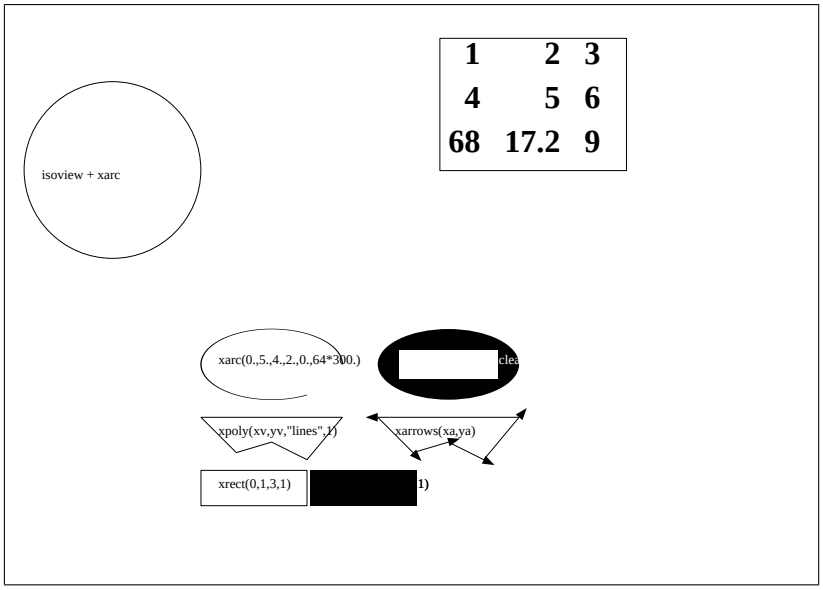


圖 6.7: Geometric Graphics and Comments

6.4.6. 自動控制常用之繪圖指令

Scilab 提供以下自動控制領域常用之繪圖指令：

- bode : Bode 圖，繪出線性系統頻率響應之相位及振幅

- gainplot :與 Bode 圖相同，但僅繪出頻率響應之振幅
- nyquist : Nyquist 圖，線性系統頻率響應之實部及虛部 p
- m_circle : M-圓 (M-circle) 圖，用於 Nyquist 圖示
- chart :繪出 Nichols 圖
- black :繪出 Black diagram (Nichols 圖) for a linear system.
- evans : Evans 根軌跡
- plzr :繪出線性系統之極–零點 (pole-zero)

```
s=poly(0,'s');
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01));
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225));
//bode
subplot(2,2,1)
gainplot([h1;h],0.01,100);
//nyquist
subplot(2,2,2)
nyquist([h1;h])

//chart and black
subplot(2,2,3)
black([h1;h],0.01,100,['h1';'h'])
chart([-8 -6 -4],[80 120],list(1,0));
//evans
subplot(2,2,4)
H=syslin('c',352*poly(-5,'s')/poly([0,0,2000,200,25,1],'s','c'));
evans(H,100)
```

6.4.7.其他

- edit_curv :交談式曲線圖形編輯器
- gr_menu :另一個交談式圖形編輯器
- locate :用 2D 圖形上取得滑鼠所在位置之座標值，例如：x=locate(5,1) 會將滑鼠所點取的 5 個 xy 座標存在 x 矩陣內

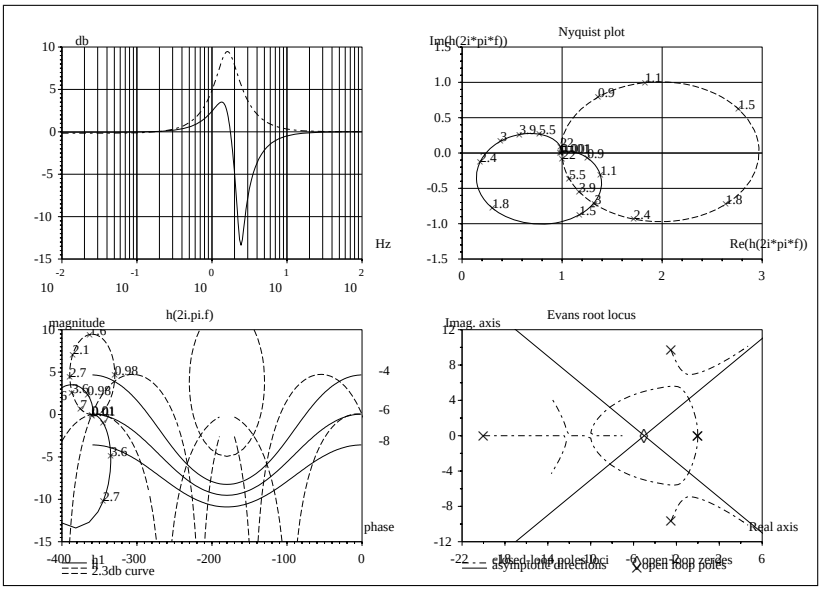


圖 6.8: Some Plots in Automatic Control

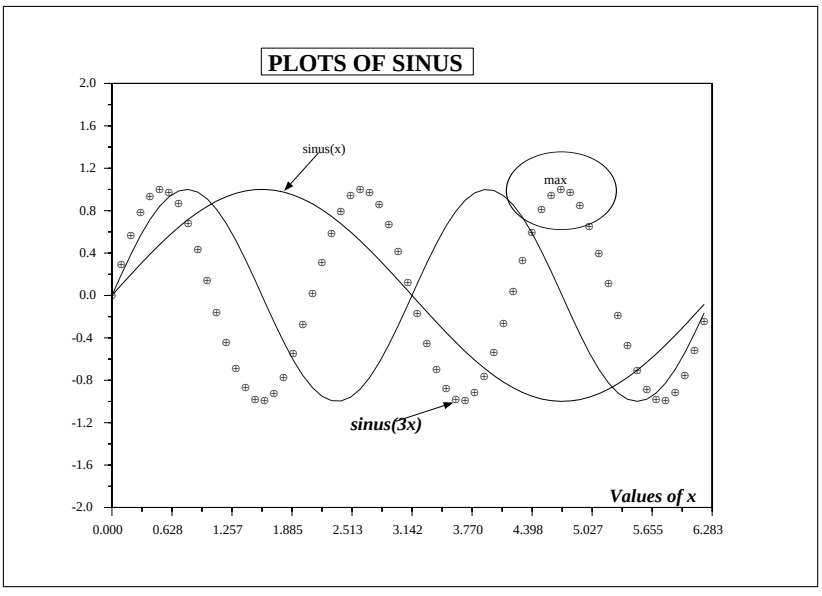


圖 6.9: Presentation of Plots

6.5.3D 繪圖

6.5.1.3D 繪圖一般指令

- plot3d : 3D 空間座標曲面之繪圖: plot3d(x,y,z) x,y,z 爲 3 座標矩陣, z 爲在 x ,y 座標點上之數值。其他參數爲選項。
- plot3d1 : 3D 空間曲面座標之繪圖，以灰階 (gray levels) 表示。
- fplot3d : 3D 空間座標曲面之繪圖: fplot3d(x,y,f) 中，參數 x,y 爲座標矩陣; f 是用來計算 z 座標的函數
- fplot3d1 : 3D 空間座標曲面之繪圖以灰階 (gray levels) 表示: fplot3d1(x,y,f) 中， x,y 爲座標矩陣; f 是用來計算 z 座標的函數

6.5.2.3D 繪圖特定指令

- param3d : 3D 參數化曲線 (parametric curves) 繪圖
- contour : 3D 等高線繪圖
- grayplot10 : 2D 灰階圖
- fcontour10 : 3D 等高線繪圖，以函數表示高度 l
- hist3d : 3D 直方圖 (histogram)
- secto3d :由截面建構 3D 曲面
- eval3d :計算網格 (grid) 上之函數值 (請參考 feval)

6.5.3.2D 及 3D 混合繪圖

當使用 3D 繪圖時，內定的繪圖邊界只在 3D 圖形有效。如果想要在 3D 圖中加入其他圖形資訊，可以利用 geom3d 函數將 3D 座標轉爲可直接操作的 2D 座標。圖 6.10 展示此函數功能，原始碼如下：

```
xinit('d7-10.ps');
    r=(%pi):-0.01:0;x=r.*cos(10*r);y=r.*sin(10*r);
    function  z=surf(x,y),z=sin(x)*cos(y);endfunction
    t=%pi*(-10:10)/10;
    fplot3d(t,t,surf,theta=35,alpha=45,leg="X@Y@Z",flag=[-3,2,3]);
    z=sin(x).*cos(y);
    [x1,y1]=geom3d(x,y,z);
    xpoly(x1,y1,"lines");
    [x1,y1]=geom3d([0,0],[0,0],[5,0]);
    xsegs(x1,y1);
    xstring(x1(1),y1(1),' The  point  (0,0,0)');
xend();
```

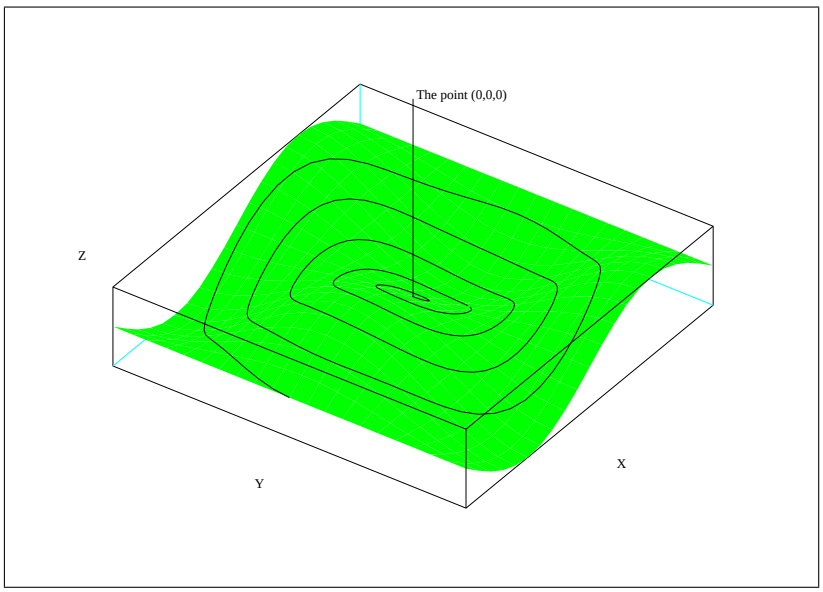


圖 6.10: 2D and 3D plot

6.5.4.次視窗

單一圖形視窗內，可以力分割出區域分繪圖，例如： (圖 6.11).

```
xinit('d7-8.ps');
  t=(0:.05:1)';st=sin(2*%pi*t);
  subplot(2,1,1)
  plot2d2(t,st);
  subplot(2,1,2)
  plot2d3(t,st);
  xsetech([0,0,1,1]);
xend();
```

6.5.5.繪圖範例集

下一個範例對 2D 或 3D 繪圖做一簡略之總結。圖 6.12 是由由以下程式碼中產生四張 Postsctipt 圖檔，再以 Scilab 所附的工具 Blatexprs 將原始 Postsctipt 圖檔合成一張能用於 LaTeX 檔案之格式。

```
//some examples
str_l=list();
//
str_l(1)=['plot3d1();';
        'title=["plot3d1 : z=sin(x)*cos(y)"];';
        'xtitle(title,"  ")'];
```

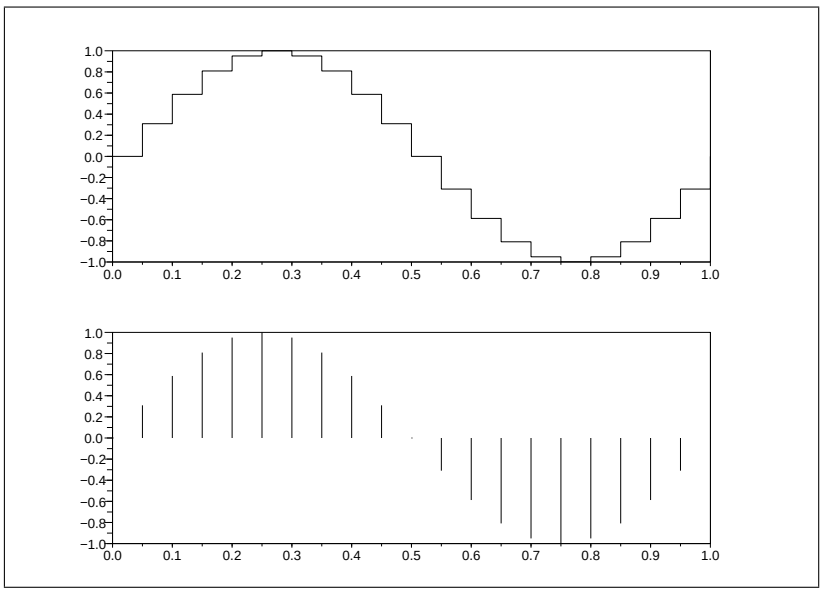


圖 6.11: Use of subplot

```
//
str_l(2)=[
        'set("figure_style","old")';
        'contour();';
        'title=["contour "];';
        'xtitle(title," "," ");'];
//
str_l(3)=[
         'set("figure_style","old")';
         'champ();';
        'title=["champ "];';
        'xtitle(title," "," ");'];
//
str_l(4)=['t=%pi*(-10:10)/10;';
              'deff("[z]=surf(x,y)","z=sin(x)*cos(y)");';
              'rect=[-%pi,%pi,-%pi,%pi,-5,1];';
              'z=feval(t,t,surf);';
              'contour(t,t,z,10,35,45,"X@Y@Z",[1,1,0],rect,-5);';
              'plot3d(t,t,z,35,45,"X@Y@Z",[2,1,3],rect);';
              'title=["plot3d and contour "];';
              'xtitle(title," "," ");'];
//
for i=1:4,xinit('d7a11.ps'+string(i));
```

```
execstr(str_l(i)),xend();end
```

6.6.在 LaTeX文件中插入 Scilab 圖檔

本節介紹 Scilab 所提供輔助工具，以方便將 Scilab 圖檔插入文件中。這些工具位於 SCIDIR/bin 目錄中。

6.6.1.由視窗到紙張

產生書面圖案的最簡單方式是在圖形視窗中，選擇 File/Print 按鍵，即能製作 Postscript 檔或直接在印表機列印。

6.6.2.產生 Postscript 檔

但要能與其他文件整合，較方便的方式還是先產生 Postscript 檔。以下是以程序方式自動產生 Postscript 圖檔的範例：

```
-->driver('Pos')

-->xinit('foo.ps')

-->plot3d1();

-->xend()

-->driver('Rec')

-->plot3d1()

-->xbasimp(0,'foo1.ps')
```

所產生的 Postsctipt 檔 (foo.ps or foo1.ps) 還不能直接送到印表機，因爲還缺 Postscript 前文 (preamble)。工具 Blpr 能夠補充這些前文，指令如下：

```
Blpr string-title file1.ps file2.ps > result.ps
```

此例， Postscript 前文和 file1.ps， file2.ps 兩檔組合成可以列印之 Postscript 檔 result.ps 。也可以使用 ghostview (在 Winddows 爲 gsview32) 瀏覽已加上前文的 Postscript 檔。

6.6.3.LaTeX文章內加入 Postscript 圖檔

工具 Blatexpr 、 Batexpr2 及 Blatexprs 可以用來產生能插入 LaTeX文件的 Postsctipt 圖檔。:
以前面的檔案 foo.ps 爲例，使用以下指令：]

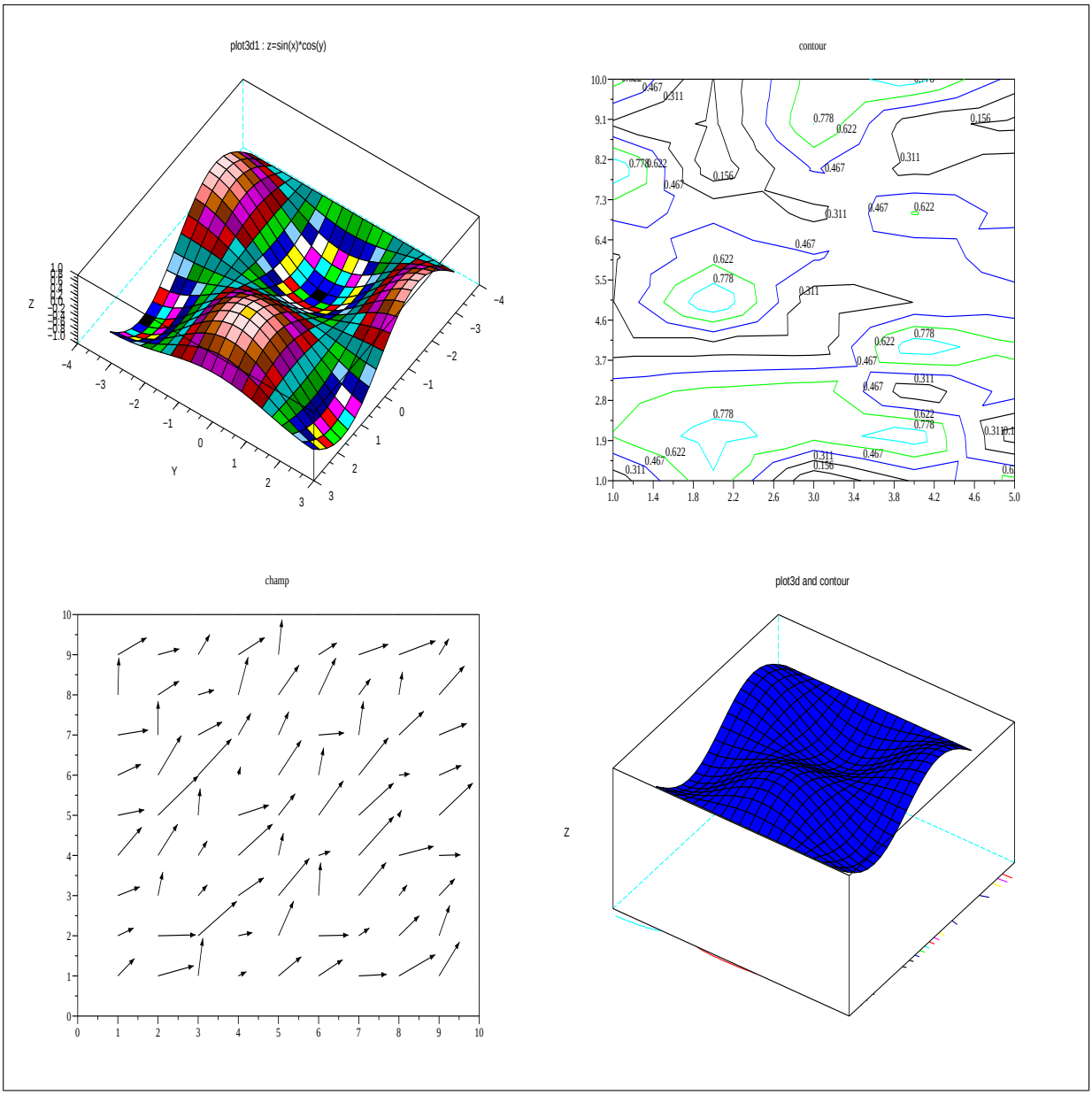


圖 6.12: Group of figures

```
Blatexpr 1.0 1.0 foo.ps
```

產生 foo.epsf 及 foo.tex 兩檔。要在 LaTeX爲文件中插入此圖，只要將所產生的 foo.tex 檔如下讀入

```
\input foo.tex
\dessin{圖標題}{The-label}
```

其中 The-label 是 LaTeX文章中準備引用此圖之文字標籤，例如 LaTeX文章中可能爲"請參考圖 \ref{abc}"，則範例中的 The-label 就是 abc。

工具 Blatexprs 也做類似的事：如果幾張 Postscript 檔要合併成一張 LaTeX圖檔時使用它。

以下範例，先選擇 Postscript 圖形設施 (指令：driver('Pos'))，再繪出四張 Postscript 圖檔 fig1.ps, ..., fig4.ps 。完成繪圖後將圖形設施再設爲內定之設施 (螢幕，指令爲：driver('Rec'))。

```
-->//multiple Postscript files for Latex

-->driver('Pos')


-->t=%pi*(-10:10)/10;


-->xinit('fig1.ps')

-->plot3d1(t,t,sin(t)'*cos(t),theta=35,alpha=45,flag=[2,2,4]);

-->xend()


-->xinit('fig2.ps')

-->contour(1:5,1:10,rand(5,10),5);

-->xend()


-->xinit('fig3.ps')

-->champ(1:10,1:10,rand(10,10),rand(10,10));

-->xend()
```

```
-->xinit('fig4.ps')

-->t=%pi*(-10:10)/10;

-->function z=surf(x,y),z=sin(x)*cos(y),endfunction
Warning :redefining function: surf


-->rect=[-%pi,%pi,-%pi,%pi,-5,1];

-->z=feval(t,t,surf);

-->contour(t,t,z,10,35,45,'X@Y@Z',[1,1,0],rect,-5);

-->plot3d(t,t,z,theta=35,alpha=45,flag=[2,1,3],ebox=rect);

-->title=['plot3d and contour '];

-->xtitle(title,' ',' ');

-->xend()


-->driver('Rec')
```

然後在 Windows 的 DOS 環境或 Uinx 的 shell 環境執行以下指令：

```
Blatexprs multi fig1.ps fig2.ps fig3.ps fig4.ps
```

如此可產生 multi.tex 及 multi.ps 兩檔案。要插入圖檔，請在你的 LaTeX 文章中加入以下幾行：

```
\input multi.tex
\dessin{圖標題}{The-label}
```

同樣的，如果要將兩張擺在一起可以如下使用工具 Blatexpr2 ：

```
Blatexpr2 Fileres file1.ps file2.ps
```

LaTeX文章內的顯現結果爲圖 6.13：

使用 LaTeX 文件時，將圖檔放在文件目錄下的次目錄內管理上較爲方便。例如，圖檔若放在 figures 次目錄時，可以在 LaTeX 文章內如下定義

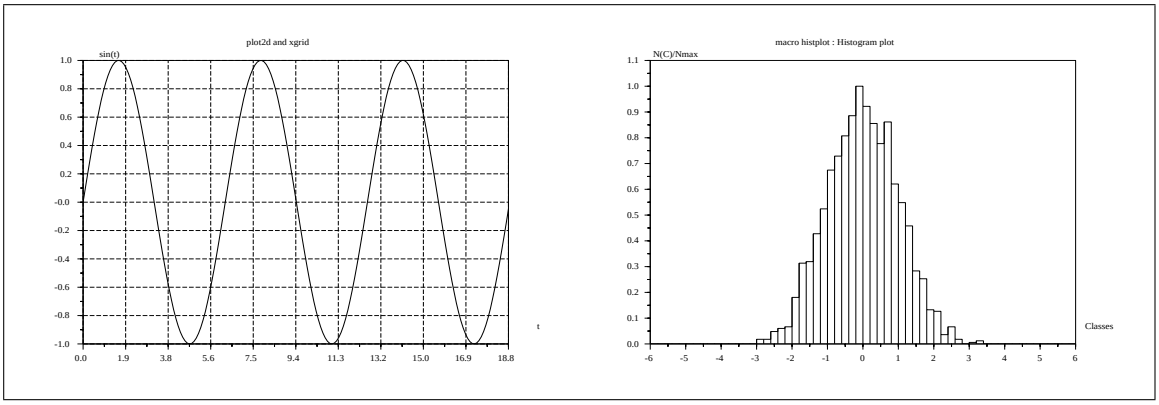


圖 6.13: Blatexp2 Example

```
\def\Figdir{figures/}  % 定義圖檔次目錄
... 其他內
...
\input{figures/fig.tex}  % 讀入插圖指令
\dessin{The caption of you picture}{The-label}
```

其中，LaTeX指令 \dessin 是由 Blatexpr 這類工具所產生的檔案 fig.tex 內部所定義，而主文所定義的 \Figdir 指令，則會在 \dessin 被引用以正確找到 Postscript 檔 fig.ps 。

6.6.4.EPS 檔 (Encapsulated Postscript)

如果產生 EPS 檔而非 Postscript 檔時，可以使用指令 BEpsf 而非 Blatexpr 。這時所產生的圖檔名稱格式爲.epsf 而非.ps。

第七章

以C或Fortran程式與Scilab溝通

內容

Scilab 可以很容易以 Fortran 或 C 語言設計介面。這項能力有以下好處：
itemize

計算效率的提升

方便使用者針對特定數值問題須要提供計算函數。如：模擬、最佳化等問題中使用者須要提供函數以代表求解之方程式

須要使用現有 Fortran 或 C 程式庫擴大功能。例如 Lapack 或 netlib 之各數值模組。在軟體環境中使用發展完備之數值程式庫是相當實際的作法。

當然， Fortran 或 C 程式碼必須與 Scilab 環境連結才能使用。連結方式可以使用動態聯結或重新產生一版內含所添加的 Fortran、 C 程式碼的 Scilab 執行檔皆可。

爲了能夠在 Scilab 環境中執行 Fortran、 C 程式，函數參數在不同語言環境中必須正確轉換。但爲了節省開發時間， Scilab 中的部分函數，設計成可以直接接受 Fortran、 C 程式的參數格式， Scilab 系統會自行在內部進行參數格式的轉換。例如用於常微分方程組 $\dot{x} = f(t, x)$ 的 ode 指令所需提供的函數 f ，在 Scilab 中可以使用 Scilab、 C 及 Fortran 三種語言的原始格式提供。 Scilab 系統中，非線性最佳化模組功能也提供此類直接聯結之功能。

要與 Fortran、 C 這內外部函數連結最簡單的方式是使用 link 指令，將動態程式庫動態連結、載入到系統中再使用 call 指令呼叫這些函數。有關外部函數之參數資料格式，是透過 call 指令的參數告知 Scilab 以正確轉換參數。 Scilab 次目錄 SCIDIR/examples/link--examples--so 內含許多這類範例，

較高階的方連接方法是自行提供介面函數，此介面函數負責不同語言環境間的變數轉換。Scilab 次目錄 examples/interface-tutorial-so 及 examples/interface-tour-so 中提供許多這類範例。另外次目錄 examples/mex--examples 則提供類似 Matlab Mex 函數類似之介面以方便 Matlab 使用者轉移程式。

介面函數也可以利用工具 intersci, intersci-n 由一描述 C、 Fortran 函數界面訊息的檔案 .desc 自動產生。

最後，也可以將上所建立的函數，直接以靜態連結方式成爲 Scilab 環境系統之一基本函數。這可以更新檔案 routines/default/fundef 內容。再重新編譯 Scilab 即可。

7.1.使用動態聯結

目錄 examples/link-examples-so 內有一些使用動態聯結方式，呼叫 C/Fortran 函數的簡單範例。本節以其中之一爲例，解釋編譯、連結及呼叫方式。

7.1.1.動態聯結

檔案 ext1c.c 及 ext1f.f 爲 C 及 Fortran 函數，進行相同的數值工作，也就是兩向量相加：

$$\vec{\mathbf{c}} = \vec{\mathbf{a}} + \vec{\mathbf{b}}$$

以 Fortran 爲例，檔案內容如下：

```
      subroutine ext1f(n,a,b,c)
c     Copyright Inria/Enpc
      double precision a(*),b(*),c(*)
      do 1 k=1,n
         c(k)=a(k)+b(k)
 1    continue
      return
      end
```

將此 Fortran 函數編譯成動態連結檔的 Scilab 指令在檔案 ext1f.sce 中，檔案前半內容爲：

```
//
//      .... ext1f.sce 內容 (前半部)....
//
link_name = 'ext1f';      // 加入程式庫的所有函數
flag  = "f";                    // ext1f 為 a Fortran function
files = ['ext1f.o' ];    // 編譯檔 (Object files)
libs  = [];                     // 連結所需得其他外部程式庫
```

```
ilib_for_link(link_name,files,libs,flag);
//
// 其他指令 ...
//              ...
```

其中 flags 爲一旗標，告知所編譯的爲 Fortran 檔案。 ilib_for_link 爲 Scilab 之高階指令，早期版本有關編譯連結的指令以此高階指令取代。 ilib_for_link 實際上進行以下複雜之編譯、連結程序：

- 產生一個用來載入此動態程式之 Scilab 載入程序檔，檔名爲 loader.sci。
- 產生一個與作業系統相關的軟體建構檔 (Makefile)，檔名爲 Makelib (在 Windows 環境爲檔名爲 Makelib.mak)
- 執行軟體建構程序 (Unix 中啓動 Makefile, Windows 啓動 nmake)，產生動態程式庫，程式庫名稱若未給定，將以第一個函數名稱決定，以本例爲例，動態程式庫名稱爲 libext1f.dll

7.1.2.呼叫動態連結之程式

一旦動態程式庫編譯完成，隨時可以透過 ilib_for_link 所產生的載入程序檔 loader.sce 將此動態程式庫 libext1f.dll 載入系統再呼叫此 Fortran 函數，如：

```
//
//       .... ext1f.sce 內容 (後半部)....
//
//
// 將程式庫掛上
exec loader.sce; // loader.sce 由 指令 ilib_for_link 自動產生

// 利用 call 指令呼叫 C/Fortran 程式庫

a=[1,2,3];b=[4,5,6];n=3;
c=call('ext1f',n,1,'i',a,2,'d',b,3,'d','out',[1,3],4,'d');
```

C/Fortran 函數一旦載入，就可以使用 call 指令直接呼叫。call 指令的一般格式爲

```
[y1,y2,y3...]=call('函數名稱',輸入描述 ...
                           'out'       ,輸出描述 ...);
```

其中 '函數名稱' 爲動態程式庫內 C/Fortran 函數之名稱， 'out' 爲一固定字串，用來分隔輸入及輸出參數的描述。例如上例中之 call 指令之詳細註解如下：

```
c=call('ext1f', // 函數名稱為 ext1f
   //           Scilab 變數 , C/F77 函數參數位置   , 參數型態
                  n,              1,                    'i',//整數
                  a,              2,                    'd',//實數
```

```
                b,              3,                  'd',//實數
        'out'   , // 輸出分隔
//              變數大小        C/F77 函數參數位置   , 參數型態
                [1,3],          4,                  'd');//實數
```

請以指令 help call 查詢相關消息。值得注意的是，輸出參數須要提供參數大小資訊，但是不必提供 Scialb 變數。

7.2.介面程式

7.2.1.設計介面程式

如果要將 C/Fortran 函式直接設計成 Scilab 的一個指令，而不是透過 call 指令呼叫時，還需要進一步提供面函數以作爲連結 Scialb 指令及 C/Fortran 函數之介面。

設計介面函數最容易的學習方式，是由範例模仿著手。 examples/interface-tutorial-so 及 examples/interface-tour-so 有許多可供學習介面函數的範例。

7.2.2.範例

以下討論 examples/interface-tutorial 內的一個範例。

C 函數 matmul 位於檔案 intmatmul.c，功能爲如下之矩陣相乘。

```
/*矩陣相乘   C=A*B, (A,B,C stored columnwise) */

#define A(i,k) a[i + k*n]
#define B(k,j) b[k + j*m]
#define C(i,j) c[i + j*n]

void matmul(a,n,m,b,l,c)
double a[],b[],c[];
int n,m,l;
{
int i,j,k; double s;
for( i=0 ; i < n; i++)
    {
      for( j=0; j < l; j++)
          {
          s = 0.;
          for( k=0; k< m; k++)
              {
               s += A(i,k)*B(k,j);
              }
           C(i,j) = s;
           }
```

```
    }
}
```

我們希望提供一個 Scilab 指令 (一樣也叫座做 matmul)，使得在 Scilab 環境可以如下求得 A , B 矩陣之乘積：

```
-->C=matmul(A,B)
```

並將結果存在 C 矩陣內。因此在 Scilab 之 A, B 矩陣變數必須有管道傳入 C/Fortran 函數內。而變數 C 也需要在某一處先行產生再傳回 Scilab 環境中。

爲了產生 Scilab 指令 matmul, 我們提供以下 C 通道 (gateway) 函數 intmatmul 作爲介面。

```
#include "stack-c.h"

/*--------------------------------------------------------
 * intmatmul: 指令 matmul 之 C 介面函數
 *      在 Scilab 使用 : C=matmul(A,B)
 *--------------------------------------------------------*/

void matmul __PARAMS((double *a, int n, int m, double *b, int l, double *c));

int intmatmul(fname)
       char *fname;
{
  static int l1, m1, n1, l2, m2, n2, l3;
  static int minlhs=1, maxlhs=1, minrhs=2, maxrhs=2;

  /* 檢查輸入及輸出參數數目 (rhs=2) , (lhs=1) 是否落於允許範圍*/
  CheckRhs(minrhs,maxrhs) ;
  CheckLhs(minlhs,maxlhs) ;

  /* 取得 A 、 B 及產生 C 矩陣位置 */
  GetRhsVar(1, "d", &m1, &n1, &l1);    /* 取得第一 Scialb 參數之大小 m1,n1 及堆疊位置 l1 */
  GetRhsVar(2, "d", &m2, &n2, &l2);    /* 取得第二 Scialb 參數之大小 m2,n2 及堆疊位置 l2 */
  CreateVar(3, "d", &m1, &n2, &l3);    /* 產生第三 Scialb 參數,大小為 m1,m2  堆疊位置 l3 */

  /* 維度檢查    */
  if (!(n1==m2)) { sciprint("%s: Incompatible inputs \r\n", "matmul");
    Error(999);
    return 0;}

  /* 呼叫原始  C 函數, stk 函數由堆疊位置傳回記憶體位置
   *   輸入:stk(l1)->A, stk(l2)->B
```

```
    *   輸出:stk(l3)->C
    */
   matmul(stk(l1), m1, n1, stk(l2), n2, stk(l3));

   /*   輸出設為第三變數     */
   LhsVar(1) = 3;
   return 0;
}
```

介面函數 intmatmul 必須引入檔案 SCIDIR/routines/stack-c.h ，通常這是介面函數的第一行。介面函數的名稱爲 intmatmul 而輸入參數爲 fname。 fname 之型態爲 char *。介面函數之名稱爲任意，但輸入參數 fname 卻是強制性，一定要有此參數。介面函數內部程序首先建立 Scilab 環境輸入及輸出變數和 C/Fortran 函數之間的關係。 Scilab 對矩陣運算 C=A*B 而言，A, B 及 C 三矩陣在介面函數內以數字 1, 2 及 3 代表。

介面函數內

```
   static int minlhs=1, maxlhs=1, minrhs=2, maxrhs=2;

   /* 檢查輸入及輸出參數數目 (rhs=2) , (lhs=1) 是否落於允許範圍*/
   CheckRhs(minrhs,maxrhs) ;
   CheckLhs(minlhs,maxlhs) ;
```

目的是檢查輸入及輸出參數數目，是否爲預期。其中縮寫 Rhs (Right hand side) 代表輸入參數，Lhs (Left hand side) 則代表輸出參數。由於介面函數 intmatmul 爲 Scilab 在執行 C=matmul(A,B) 時爲 Scilab 內部呼叫，因此在執行到此處時，輸入及輸出參數之數目系統已可得。 CheckRhs 及 CheckLhs 即爲系統提供之檢查函數。

介面函數的下一件工作是在 C/Fortran 函數環境中建立 Scialb 變數， A, B 及 C。在介面函數內，所有的 Scilab 變數都以整數代表。因此 A, B 及 C 分別以數字 1, 2 及 3 代表。每一 Scilab 的輸入參數接須要以 GetRhsVar 產生 C/Fortran 函數所需要的對應位置。同樣的，每一 Scilab 的輸出參數也要以 CreateVar 產生記憶位置以及 Scilab 對應位置。

```
   ..
   /* 取得 A 、 B 及產生 C 矩陣位置 */
   GetRhsVar(1, "d", &m1, &n1, &l1);/* 取得第 1   參數之大小( m1,n1) 及堆疊位置 l1 */
   GetRhsVar(2, "d", &m2, &n2, &l2);/* 取得第 2   參數之大小( m2,n2) 及堆疊位置 l2 */
   CreateVar(3, "d", &m1, &n2, &l3);/* 產生第 3   參數, 大小為( m1,m2) 堆疊位置 l3 */
```

GetRhsVar 及 CreateVar 的第 1 參數爲 Scilab 變數的編號，必須依照順序與 Scilab 指令之輸入及輸出參數對應。第 2 參數代表參數型態。 3、 4 參數代表。矩陣大小。第 5 參數爲一整數值，代表 Scilab 變數所在的堆疊位置。 CreateVar 用於產生新的 Scilab 變數，因此 Scilab 的輸出變數都需要使用此函數先行產生記憶位置。輸出變數要緊接著輸入參數編號。此例，輸出參數爲 C，編號要緊接著 A, B 之編號因此爲 3。

C 函數中之判斷

```
if (n1 !=m2 )
      {Scierror(999,"%s: Uncompatible dimensions\r\n",fname);
                  return 0;}
```

用來確定傳入之矩陣 A 及 B 之維度正確，能夠進行乘法運算： A 之列數必等於 B 之行數。

Scilab 及 C 語言之變數對應完成後，就可呼叫實際計算的函數 matmul 了。

```
void matmul(a,n,m,b,l,c)
double a[],b[],c[]; int n,m,l;
```

我們必須將 A, B 及 C 矩陣之實數指標 (pointer) 傳給 matmul 。透過函數 stk，可以由變數之堆疊編號取得指標。因此如下呼叫 matmul This is done by :

```
matmul(stk(l1), m1, n1, stk(l2), n2, stk(l3));
```

A, B 及 C 矩陣之實數指標以堆疊編號 l1, l2 及 l3 間接取得。

結束介面函數之前，要將 Scilab 環境所等待的輸出變數資訊先置放在輸出堆疊上，

```
LhsVar(1) = 3;
```

這個指令的意思是將編號 3 變數當做第 1 個輸出變數 (Lhs: Left hand side, 代表指令左邊項，也就是輸出項)。重回 Scilab 環境時，系統可以據此資訊取的輸出變數內容。

一旦介面程式完成，就需要進行一般之編譯連結程序。以下爲一編譯連結 Scilab 程序檔 builder.sce

```
// 目錄 examples/interface-tutorial-so
// builder.sce 產生介面程式庫
// 共有兩個介面函數

ilib_name  = 'libtutorial'                  // 介面程式庫名稱
files = ['intview.o','intmatmul.o'] // objects 檔
                                            //
libs   = []                                   // 其它連結所需程式庫
table = [ 'view', 'intview';                 // (Scilab 函數名稱,介面函數名稱) 對應表
          'matmul','intmatmul'];      // 若使用 Fortran 介面, 利用巨集 'C2F(name)'

// 編譯、連結
// ------------------------------------------------
ilib_build(ilib_name,table,files,libs)
```

此例共將兩個介面函數 intview 、intmatmul 編譯在一個動態連結程式庫內。 Scilab 變數 table 爲 Scilab 與 C/F77 介面函數名稱之對應表。

ilib_build 與前節所介紹的 ilib_for_link 指令一樣，都是高階指令。執行 ilib_build 過程中，實際上與作業系統溝通進行以下便一連結程序

- 產生一個通道 (gateway) 函數原始檔，此函數對 Scilab 提供單一之窗口以代表依程式庫內所有介面函數。
- 產生一個用來載入此動態程式之 Scilab 載入程序檔，檔名爲 loader.sci。
- 產生一個與作業系統相關的軟體建構檔 (Makefile)，檔名爲 Makelib (在 Windows 環境爲檔名爲 Makelib.mak)

- 產生另一個與作業系統相關的高階軟體建構檔 (Makefile)，檔名爲 Makefile (在 Windows 環境爲檔名爲 Makefile.mak)，此檔控制所有建建構序。
- 執行軟體建構程序 (Unix 中啓動 Makefile, Windows 啓動 nmake)，產生動態程式庫

早期版本的 Scilab，使用者必須自行撰寫通道 (gateway) 函數，而 ilib_for_link 指令已簡化這個程序了。對 ilib_for_link 運作原理有興趣者，可以在執行完 builder.sce 後檢視所自動產生的檔案，以了解用途。

使用此編譯後之動態程式庫之程序，可以參考測試檔 libtutorial.tst ，內容如下：

```
// 目錄: interface-tutorial-so
// 檔案: libtutorial.tst 測試介面函數功能
tutorial_path=get_file_path('libtutorial.tst');
exec(tutorial_path+'/loader.sce'); // loader.sce 為自動產生
A=ones(2,2);B=ones(2,2);
C=matmul(A,B); // 計算 A*B, 併與系統計算結果相比
if norm(A*B-matmul(A,B)) > %eps then pause,end
```

其中檔案 loader.sce 爲前一步驟所自動建立之程序檔。

7.2.3.建立介面所需的函數

Scilab 解譯器的核心部份是由 Fortran 撰寫而成，所有 Scilab 變數、物件等資訊皆需要參考到這些 Fortran 函數。對 C/C++開發者， Scilab 提供 C 巨集以封裝這些 Fortran 函數，因此大部分介面函數是以 C 語言透過這些巨集與 Scilab 核心溝通。這類巨集，以函數之語法定義於檔案 stack-c.h 中，主要內容說明如下：

- CheckRhs(minrhs, maxrhs)
  CheckLhs(minlhs, maxlhs)

  函數 CheckRhs 用來檢查輸入參數之數目 Rhs。 Rhs 必須滿足
  minrhs <= Rhs <= maxrhs。函數 CheckLhs 用來檢查輸出參數之數目 Lhs。 Lhs 必須滿足
  minlhs <= Lhs <= maxlhs. (通常 minlhs=1 因爲 Scilab 函數永遠可以接收較預期爲少的輸出參數)。

- GetRhsVar(k,ct,&mk,&nk,&lk)

  GetRhsVar 取得第 k 個輸入變數之資訊。注意 k (整數) 及 ct (字串) 爲輸入而 mk,nk 及 lk (整數) 爲 GetRhsVar 之輸出。此函數取得第 k 個型態爲 (ct) 之變數堆疊位置。 mk,nk 爲此變數矩陣之維度大小 (純量、向量皆視爲矩陣之特例)。 lk 是 Scilab 核心中用來存放此變數之內部堆疊位置 (整數)。編號 k 所代表之變數型態用字串 ct 代表，此值可爲 "d", "r", "i","z" 或 "c" 代表雙精度實數 (double)、單精度實數 (float)、整數、雙精度複數 (double complex) 及字串。

  介面函數必須爲每一輸入參數依序呼叫一次 GetRhsVar 函數。

- CreateVar(k,ct,&mk,&nk,&lk)

CreateVar 在介面函數內新產生一個 Scilab 變數。此數 k,ct,&mk,&nk 爲 CreateVar 函數之輸入項而 lk 則爲 CreateVar 的輸出項。注意，當呼叫 CreateVar 時，變數編號 k 必須大於所有輸入參數之編號。

- CreateVarFromPtr(k,ct,&mk,&nk,&lk)

  此函數如同 CreateVar 一樣，也是新產生一 Scilab 變數。與 CreateVar 不同處是變數位置在呼叫 CreateVar 前已經存在。lk 爲 C 語言之指標 (pointer) 指向此記憶位置。

一旦所有的 Scilab 輸出入變數皆以函數 GetRhsVar、CreateVar 或 CreateVarFromPtr 編號對應後，就可以呼叫實際負責計算的函數。計算的函數所需要知道的記憶位置，可以由 Scilab 內部所維護的變數堆疊得知。 Scilab 的字串、整數、單精度實數、雙精度實數、雙精度複數堆疊分別以列陣cstk, istk, sstk, stk, zstk代表。使用者利用GetRhsVar或CreateVar 所得到堆疊位置即能取的變數記憶指標位置。

數值函數算完畢後，必須將 Scilab 所需的輸出變數資訊放置到定位置以供 Scilab 系統提取資料。利用全域變數 LhsVar 可以在介面函數結束前將輸出變數資訊存起。例如：

```
LhsVar(1) = 5;
LhsVar(2) = 3;
LhsVar(3) = 1;
LhsVar(4) = 2;
```

意味此 Scilab 函數共有 4 個輸出變數，分別對應到編號爲 k= 5, k=3, k=1, k=2 之變數。

函數 sciprint(amessage) 及 Error(k) 用在列印警告及錯誤訊息。

其他常用函數如下：

- GetMatrixptr("Aname", &m, &n, &lp);

  此函數將 Scilab 變數名稱爲 Aname 之數值讀入所指定之 C 函數指標lp 中。而矩陣大小則讀入 m,n 中。

- ReadString("Aname",&n,str)

  此函數將字串內容讀入 str 中，字串大小讀入 n 中。

7.2.4.範例

examples/interface-tour-so 內含許多以 C 及 Fortran 撰寫之範例。建立 Scilab 介面函數的最快方式，就是由這些範例中，複製一個類似例子再據以修改成你所需要的結果。

7.3.Intersci 工具

Scilab 除了內含諸如 ilib_for_link、ilib_build 等高階指令外，也提供 Intersci,Intersci-n 工具，利用介面描述檔自動產生介面函數以方便連結 Forntra 及 C 程式資源。

次目錄 examples/intersci-examples-so 內含許多使用 Intersci,Intersci-n 工具的範例可供學習。本節藉其中一例以解說 Intersci,Intersci-n 用法。

假設已有以下 C 函數：

```
int ext1c(n, a, b, c)
      int *n;
      double *a, *b, *c;
{
  int k;
  for (k = 0; k < *n; ++k)
       c[k] = a[k] + b[k];
  return(0);
}
```

此函數只是簡單的進行 n 維向量 a 及 b 之相加運算，並將結果存在向量 c 中。

我們希望設計介面使得在 Scilab 中能夠用 c=ext1c(a,b) 指令進行向量相加運算。因此，提供一界面描述檔 ex01fi.desc 內容爲：

```
ext1c          a b
a            vector          m
b            vector          m
c           vector   m

ext1c    m a b c
m           integer
a          double
b          double
c          double

out          sequence          c
**********************
```

此檔以空白行分隔成三部份。第一部分 (前四行) 描述 Scilab 函數：c=ext1c(a,b)，第二部份 (中間四行) 描述實際負責計算的 C/F77 函數，最後一行設定輸出變數的名稱。

將 ex01fi.desc 作爲 intersci 指令之參數如下：

```
SCIDIR/bin/intersci-n ex01fi
```

產生兩個檔案：ex01fi.c 及 ex01fi_builder.sce 。 ex01fi.c 是 C 版本之介面函數檔，內容如下 (可參考 7.2.2) ：

```
#include "stack-c.h"

int intsext1c(fname)
    char *fname;
{
 int m1,n1,l1,mn1,m2,n2,l2,mn2,un=1,mn3,l3;
 CheckRhs(2,2);
 CheckLhs(1,1);
```

```
  /*   checking variable a */
  GetRhsVar(1,"d",&m1,&n1,&l1);
  CheckVector(1,m1,n1);
  mn1=m1*n1;
  /*   checking variable b */
  GetRhsVar(2,"d",&m2,&n2,&l2);
  CheckVector(2,m2,n2);
  mn2=m2*n2;
  /* cross variable size checking */
  CheckDimProp(1,2,m1*n1 != m2*n2);
  CreateVar(3,"d",(un=1,&un),(mn3=mn1,&mn3),&l3);/* named: c */
  C2F(ext1c)(&mn1,stk(l1),stk(l2),stk(l3));
  LhsVar(1)= 3;
  return 0;
}
```

檔案 ex01fi_builder.sce 內容如下：

```
// generated with intersci
ilib_name = 'libex01fi'                    // interface library name

table =["ext1c","intsext1c"];
ilib_build(ilib_name,table,files,libs);
```

這是一個動態程式庫建構檔，可以在設定 files 及 libs 變數後執行此檔：

```
-->files = ['ex01fi.o' , 'ex01c.o'];
-->libs   = [] ;
-->exec ex01fi_builder.sce
```

除了編繹連結出動態程式庫之外，建構成續頁自動產生檔案 loader.sce ，以載入動態程式庫。(參考 7.2.2) 例如：

```
-->exec loader.sce
-->a=[1,2,3];b=[4,5,6]; c=ext1c(a,b);
```

如果要產生 Fortran 版本之介面程式，可以使用 intersci 而非 intersci-n。實際負責執行計算的函數以及介面函數不一定要使用同一語言，程式開發者可以選擇自己較熟析之語言作爲介面函數語言。

7.4.函數參數 (Argument functions)

Scilab 部份內設之非線性數值工具，如 ode (常微分方程組) 或 optim (非線性最佳化)，均要求輸入一函數做以描述所求解的方程式。例如指令 ode(x0,t0,t,fydot), fydot 是一函數代表一爲微分方程組。這類函數可以是一 Scilab 函數或是一外部函數，例如以 C 或 Fortran 語言撰寫的函數。以下以 Fortran 語言爲例，解釋如何提供函數讓 Scilab 的常微分工具能夠於內部呼叫據以求解。

考慮以下微分方程組：
examples/misc-examples/wfex.f

$$\dot{\mathbf{y}} = \begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \end{bmatrix} = \begin{bmatrix} -0.04y_1 + 10^4 y_2 y_3 \\ -\dot{y}_1 - \dot{y}_3 \\ 3 \times 10^7 * y_2^2 \end{bmatrix}$$

在目錄 examples/misc-examples 中有一 Fortran 檔 wfex.f 內含計算此向量方程式之函數，內容如下：

```
c     檔案 : wfex.f,  位於  examples/misc-examples
      subroutine wfex (neq, t, y, ydot)
      double precision t, y, ydot
      dimension y(3), ydot(3)
      ydot(1) = -.0400d+0*y(1) + 1.0d+4*y(2)*y(3)
      ydot(3) = 3.0d+7*y(2)*y(2)
      ydot(2) = -ydot(1) - ydot(3)
      return
      end
```

要在 Scilab 中設定此 Fortran 函數爲 ode 指令所需的函數可以如下執行：

- -使用 G_make 編譯

```
//Copyright INRIA
Eps=1.e-2
//
// 時間區間 :  t = 0.0 到 t = 4.e10 之間
// 初值      : y1 = 1.0, y2 = y3 = 0.
y0=[1;0;0];t0=0;t1=[0.4,4];nt=size(t1,'*');
//     參考值:
yref=[0.9851721 0.9055180;0.0000339 0.0000224;0.0147940 0.0944596];
//
//  1. 編譯 wfex.f 並連結成 wfex.dll 動態程式庫
files=G_make(["wfex.o"],"wfex.dll");
//  2. 載入 wfex.dll 讓函數名稱 wfex 為 Scilab 環境認識
iwfex=link(files,"wfex");
//  3. 輸入 wfex 函數名稱到 ode 指令, 並將模擬結果存在 y1 中
y1=ode(y0,t0,t1,'wfex')
```

G_make 能夠先產生與系統相關的建構檔 (Makefile) 再執行此建構檔編繹、連結出動態連結檔 wfex.dll。而 link 指令則將此動態連結檔載入 Scilab 環境。 ode 指令能夠直接使用此 Fortran 函數，並不需要進行輸出入參數之轉換。

反覆使用此微分方程組時， G_make 僅需執行一次。

- -使用 ilib_for_link 編譯

```
Eps=1.e-2
y0=[1;0;0];t0=0;t1=[0.4,4];nt=size(t1,'*');
//      參考值:
yref=[0.9851721  0.9055180;0.0000339  0.0000224;0.0147940  0.0944596];
//
ilib_for_link('wfex','wfex.o',[],"f")
// 載入動態程式庫
exec  loader.sce
y2=ode(y0,t0,t1,'wfex');
```

  ilib_for_link 是更高階的用法，此例會產生動態連結檔 libwfex.dll 及載入程序檔 loader.sce。用於 ode 指令前，先載入 libwfex.dll 即可使用。

- -更動 routines/default 次目錄內之檔案 Ex-ode.f，將 Fortran 函數 wfex 加入 ‘, 再重新編譯一版 Scilab 即可在 ode 指令中直接輸入’wfex’。

routines/default 目錄內還有其他之範例，使用者可參考學習。

7.5.Matlab Mex 檔

Scilab 也模擬 Matlab 與 C/Fortran 介面溝通方式，提供 Mex 程式介面格式之呼叫方式。次目錄 SCIDIR/examples/mex-examples 內含許多使用 Matlab Mex 介面之範例。使用者原先在 Matlab 已有利用 Mex 介面設計的 C/Fortran 函數，可以參考此目錄內範例轉移到 Scilab 環境使用。