中文 Scilab

SCICOS -動態系統建構及模擬器 使用介紹 (Scicos–Scilab 功能類似 Simulink–Matlab) *

R. Nikoukhah S. Steer Scilab 中文開發群 (翻譯)

目 錄

1 中文版前言 **1**

2 介紹 **2**

3 Scicos 方塊圖編輯器 **4**
3.1 設定功能塊參數 4
3.2 模擬 4
3.3 其他功能 4

4 基本功能塊 (Blocks) 類型 **6**
4.1 連續功能塊: CBB (Continuous Basic Block) 6
4.2 離散功能塊:DBB (Discrete Basic Block) 6
4.3 跨零偵測功能塊: ZBB (Zero Crossing Basic Block) 7
4.4 同步功能塊: SBB (Synchro Basic Block) 7

5 時間參數的相依及繼承 **8**

6 建構功能塊 **9**
6.1 介面函數 9
6.1.1 語法 9
6.1.1.1 Parameters 9
6.1.2 功能塊之資料結構及定義 11
6.2 運算函數 12
6.2.1 行爲 (Behavior) 12

*Scicos 內附於 Scilab 內，以開放軟體形式一起散發。有關 Scilab 請參考: http://www-rocq.inria.fr/scilab/ 或 ADE 中文 Scilab 主網頁

6.2.2 運算函數的類型 13
6.2.2.1 運算函數:型態 0 14
6.2.2.2 運算函數:型態 1 14
6.2.2.3 Fortran 例 14
6.2.2.4 C 例 14
6.2.2.5 運算函數:型態 2 14
6.2.2.6 運算函數:型態 3 15
6.2.2.7 例 15
6.2.2.8 例 16
6.2.2.9 運算函數:型態 4 16
6.2.2.10 運算函數:型態 5 18
6.2.2.11 例 18
6.2.2.12 例 19

7 結語 **20**

1 中文版前言

本文以英文版: SCICOS-A Dynamic System Builder and Simulator 之 LaTeX原始文檔爲本進行中文翻譯。

英文原稿中常規功能塊 (Regular Basic Blocks) 在新版之 Scicos 已爲連續功能塊 (Continuous Basic Blocks) 及離散功能塊 (Discrete Basic Block) 所取代，另外 Scicos 新版也定義更多之運算函數型態。因此本文參考較新之公開文獻對這類章節過時之內容，進行改寫以符現況。

2 介紹

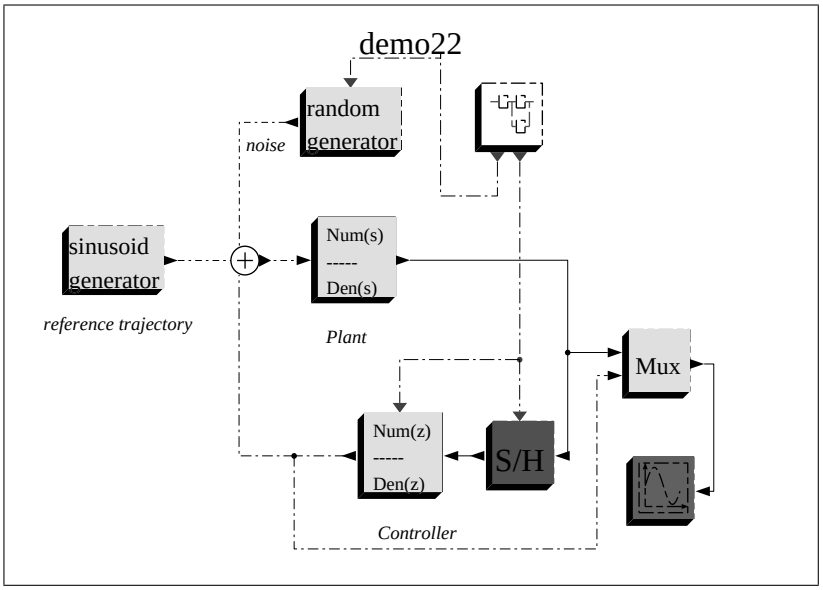

圖 1: 典型之 Scicos 方塊圖

Sicos (Scilab Connected Object Simulator) 是一 Scilab 套件，主要應用於動態系統 (包含連續或離散之次系統) 之建模 (modeling) 及模擬 (simulation)。 Scicos 內含圖形編輯器可藉聯結次系統之功能方塊以建立模型，而次系統功能方塊則爲系統內定或使用者自訂之函數 (程序)。

Scicos 中每一訊號 (signal) 都有一組時間之指標，稱爲活化時間，在活化時間內訊號持續變化，但在活化時間之外則保持常數 (請參考圖 2)。活化時間是時間區間及個別分離時間點 (稱爲事件， events) 之聯集。

Scicos 內的訊號是由活化訊號所驅動的功能塊所產生。一活化之訊號驅使功能塊內之函數以內部狀態 (state) 爲輸入參數並將功能塊計算結果當作輸出訊號。輸出訊號則由功能塊繼承其活化時間，可用以驅動另外之功能塊。

功能塊由上方之活化輸入埠 (port) 之訊號驅動。沒有活化輸入埠的功能塊代表一直在活化狀態 (時間相依)，否則它的活化時間埠的活化時間的聯集。

功能塊下方爲活化訊號的輸出埠，所輸出的訊號爲功能塊所產生的活化訊號。例如， **Clock** 功能塊產生一列等距之時間或化事件。如果將 **Clock** 之輸出訊號連接上一觀測 (scope) 方塊 (例如**MScope** 方塊) 之活化輸入埠，則觀測方塊所顯示之輸入 (左方輸入埠) 數值的時間將由上方之活化輸入埠來決定。

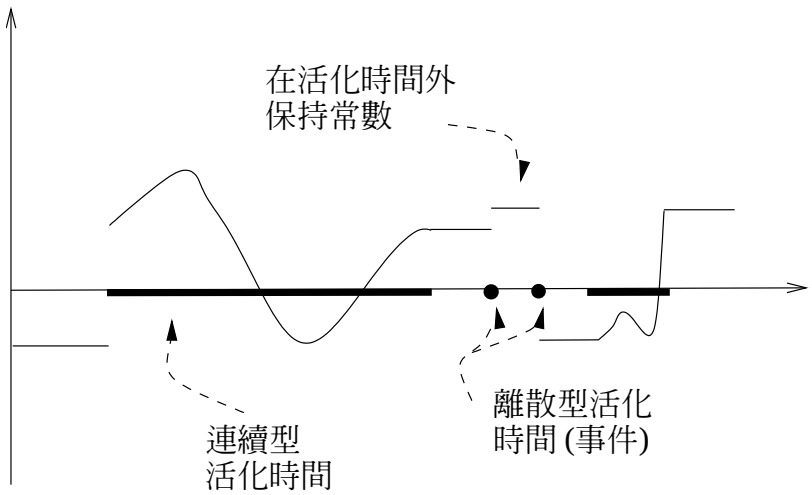

圖 2: Scicos 之訊號及其活化時間

3 Scicos 方塊圖編輯器

Scicos 藉由連結功能塊及次系統 (或稱 Super 功能塊) 而建立動態系統模型；功能塊可以在個別的圖盤 (palettes) 中找到也可以由使用者自行定義。使用方塊圖編輯器， Scicos 的使用者很容易編輯功能塊之聯結關係。在 Scilab 中，鍵入 **scicos();** 即能開啓 Scicos 方塊圖編輯器，這也是 Scicos 的主視窗。

建構 Scicos 模型一般之步驟爲

- 開啓一個或更多圖板 (使用**Edit**選單之**Palettes**按鍵),
- 從圖板中選取功能塊複製到 Scicos 主視窗；只要點選圖板中的功能塊，將游標移到主視窗後，在選定的位置上點擊一次即能完成複製。
- 利用 **Edit** 選單的 **link** 指令連接各功能塊的輸入及輸出埠；常用的方法是在將游標移到輸出訊號的功能塊之後，按右鍵後點選 **link** 指令，再移到輸入功能塊的輸入埠即能完成連結。

注意，若要從一連結線上分歧出另一連結，使用者只需要先點選已有的連結線，再按右鍵選取 **link** 指令及輸入埠即可。已有之連結可以選取 **Delete** 指令加以刪除。

同時要注意，至少要在主視窗內加入一個觀測方塊 (scope block)，或 “write to file” (寫入檔案) 方塊才能觀測或存起模擬的結果。請參考 Scicos 範例。

3.1 設定功能塊參數

功能塊的內部參數可以藉開啓功能塊的對話盒 (dialogs) 加以編輯；使用 **Open/set** 指令或直接雙擊主視窗的功能塊即可。大部分功能塊都有一對話盒可用以變更內部參數，這些參數能使用 Scialb 運算式來設定。而運算式內可以出現定義在主視窗方塊圖語境 (Context, 語言環境) 之變數。

Scicos 語言環境 (context) 是指每一 Scicos 方塊圖所具有的局部軟體環境，此環境可以利用 **Context** 按鍵設定內部變數之數學運算。**Eval** 指令能夠用來強迫這些環境參數重算。

3.2 模擬

編輯完成的 Scicos 方塊圖可以藉由 **Simulate** 選單的 **Run** 指令來執行模擬。指令 **Run** 會先編譯 (必要時) 再執行，模擬過程間可以點選主視窗上方的 **stop** 指令以強迫終止。

編譯過的 Scicos 方塊圖存成 ***.cos** 檔，此檔在下次執行會直接載入系統，不會再編譯。有經驗的使用者也可以直接執行 **scicosim** 函數以模擬 Scicos 系統方塊圖不須要進入 Scicos 環境。

3.3 其他功能

方塊圖編輯器還提供以下功能:

- 以不同各式存檔及載入方塊圖
- 放大、縮小及改變視點 (point of view)

- 改變功能塊長寬及顏色
- 改變方塊圖之背景及前景顏色
- 在方塊圖上加上文字
- 列印及輸出 Scicos 方塊圖
- 其他標準 GUI 函數。

Help 按鍵點擊功能方塊，可以叫出此功能方塊之線上手冊。 **Help** 按鍵點擊任意選單之按鍵，可以叫出按鍵之使用說明。

最後， Scicos 一個很重要的功能就是能夠產生次系統方塊 (sub-systems,Super Blocks, 超方塊)。 Scicos 能夠將一個複雜系統分割成幾個次系統，每一次系統以一 Super Blocks (超方塊) 代表。這些超方塊之行爲與一般功能方塊無異，但內部卻可由其他功能塊甚至其他超方塊共同組成。

4 基本功能塊 (Blocks) 類型

Scicos 內有四種基本功能塊:連續功能塊 (Continuous Basic Blocks)、離散功能塊 (Discrete Basic Block)、跨零偵測功能塊 (Zero Crossing Basic Blocks) 及同步功能塊 (Synchro Basic Blocks)。

這些功能塊有兩類輸出、輸入埠：常規輸出、入埠及活化輸出、入埠。常規之輸出、入埠位於功能塊之左右方，由常規連結 (link) 接連兩功能塊。而活化輸出、入埠位於功能塊之上下方，由活化連結接合兩功能塊。

4.1 連續功能塊: CBB (Continuous Basic Block)

連續功能塊 (Continuous Basic Block, CBB) 可以擁有一*連續狀態* x 和一 *離散狀態* z。如果連續狀態變數爲 x 且 u 代表常規輸入、y代表正規輸出，則， x 及 y 滿足下式

$$\dot{x} = f(t, x, z, u, p, n_e) \tag{1}$$
$$y(t) = h(t, x, z, u, p, n_e) \tag{2}$$

其中 f, h 是向量函數，p 是一向量參數，而 n_e 代表活發化時段的活化輸入埠之編碼。若 $i_1, i_2, \ldots, i_n$, 爲觸發之活化輸入埠，則

$$n_e = \sum_{j=1}^{n} 2^{i_j - 1}.$$

另一方面，如果是由事件 (離散型活化) 所觸發，狀態 x 及 z 依下式瞬間改變：

$$x(t_e) = g_c(t_e, x(t_e^-), z(t_e^-), u(t_e), p, n_e) \tag{3}$$
$$z(t_e) = g_d(t_e, x(t_e^-), z(t_e^-), u(t_e), p, n_e) \tag{4}$$

其中 t_e 代表事件發生之時間，離散狀態 z 在兩事件時之間保持常數，因此 $z(t_e^-)$ 可以解釋成狀態 z 在前一刻之數值。

連續功能塊 (CBB) 也可以產生離散型之活化輸出訊號。如果在時間 t_e 之事件觸發了活化輸出訊號，則

$$t_{evo} = k(t_e, z(t_e), u(t_e), p, n_e) \tag{5}$$

其中 t_{evo} 爲一時間向量，向量內每一項對應一個活化輸出埠之事件時間。如果事件時間早於目前時間 (t_e)，代表在對應的活化輸出埠不產生事件。事件之產生也可以事先排程，這項功能是從功能塊的參數設定功能來改變事件初始觸發時間參數 ("initial firing variables ")。

4.2 離散功能塊:DBB (Discrete Basic Block)

離散功能塊 (Discrete Basic Block, DBB) 與連續功能塊主要的不同在於只有離散狀態參數 z 而沒有連續狀態，因此可視爲連續功能塊之退化型式。 Scicos 加上此類功能塊能讓編譯器更有效率處理特定問題。

離散功能塊 (Discrete Basic Block, DBB) 之狀態 z 及輸出 y 滿足下式

$$z = f_d(t_e, z(t_e^-), u(t_e), p, n_e) \quad (6)$$
$$y = h_d(t, z, u, p, n_e) \quad (7)$$

其中 f_d, h_d 爲向量函數。與連續功能塊相同地，t_e 代表事件發生之時間，離散狀態 z 在兩事件時之間保持常數，$z(t_e^-)$ 可以解釋成狀態 z 在前一刻之數值。

離散功能塊 (DBB) 也能產生離散型之活化輸出訊號，因此 (5) 式也適用用於離散功能塊。

4.3 跨零偵測功能塊: ZBB (Zero Crossing Basic Block)

跨零偵測功能塊 (Zero Crossing Basic Block ,ZBB) 只在其中一個常規輸入正負變號時產觸發輸出事件。在這種狀況，輸出事件以及事件的時間是由產生變號的輸入埠及其變號前的正負號值來決定。

Scicos 系統中，典型的跨零偵測功能塊 (ZBB) 置放在 Threshold 圖板上。

4.4 同步功能塊: SBB (Synchro Basic Block)

同步功能塊將活化輸出訊號和活化輸入訊號同步。這類功能塊只有單一的活化輸入埠；功能塊將活化輸入訊號轉接至其中一個活化輸出埠。至於選擇哪一個活化輸出埠則是由常規輸入訊號決定。

Scicos 系統中，同步功能塊如 event select 及 If-then-else 皆置於 Branching 圖板中。

5 時間參數的相依及繼承

爲避免在 Scicos 方塊圖中繪出太多活化連結， Scicos 提供繼承之特性。如果功能快沒有任何活化輸入，那麼它會藉由常規輸入埠而繼承上游之活化特性。另外，有些功能塊永遠在活化狀態，那麼可以宣告爲 (“time dependent”, 隨時變化)，這類功能塊不需要活化輸入埠，但也不繼承上游之活化特性。

6 建構功能塊

新的功能塊可以藉由超功能塊 (Super Block) 建構、編譯而成。作爲一個新的功能塊，它必須定義一組函數:

- 介面 (*Interfacing*) 函數，負責處理與使用者之介面
- 運算 (*Computational*) 函數，負責計算特定動態行爲。

介面函數必須以 Scilab 函數定義。請參考 <SCIDIR>/macros/scicos_blocks 內之 Scilab 函數。運算函數則可以使用 C、 Fortran 或 Scilab 轉寫，請參考 <SCIDIR>/routines/scicos 。 C 及 Fortran 函數使用動態聯結 (dynamically linked) 與 Sculab 系統結合，但也可以爲了效率因素此用靜態連結。

Scifunc, GENERIC, C_block 及 Fortran_block 功能塊提供一般化之介面函數，是測試自訂之運算函數的快速功能塊原型。

6.1 介面函數

介面函數決定功能塊之幾何外觀、顏色、輸出埠數目、輸入埠數目、圖示 (icon) ..., 等等，也包含初始狀態及參數。介面函數同時負責控制使用者之對話盒 (dialog)。

介面函數到底要負責哪項功能，是由函數輸入的旗標 job 來決定。介面函數之語法如下：

6.1.1 語法

```
[x,y,typ]=block(job,arg1,arg2)
```

6.1.1.1 Parameters

- job==’plot’:此旗標指示函數繪出功能塊外觀。
  - ▷ **arg1** :功能塊的資料結構。
  - ▷ **arg2** 未用。
  - ▷ **x,y,typ**未用。

  一般而言，可使用 **standard_draw** 繪出矩形方塊及輸入輸出埠。也可用來控制功能塊之大小、圖示顏色及長寬比。
- job==’getinputs’:此旗標指示函數輸出輸入埠之位置及型態 (常規或活化輸入)。
  - ▷ **arg1** :功能塊的資料結構。
  - ▷ **arg2** 未用。
  - ▷ **x** :輸入埠的 x 座標向量。
  - ▷ **y** :輸入埠的 y 座標向量。
  - ▷ **typ** :輸入埠的型態向量 (1 爲常規輸入， 2 爲活化輸入)。

一般而言，可使用 standard_input 函數提供此項功能。

- job==’getoutputs’:此旗標指示函數輸出輸出埠之位置及型態 (常規或活化輸出)。
  - ▷ **arg1** :功能塊的資料結構。
  - ▷ **arg2** 未用。
  - ▷ **x** :輸出埠的 x 座標向量。
  - ▷ **y** :輸出埠的 y 座標向量。
  - ▷ **typ** 輸出埠的型態向量 (1 爲常規輸出， 2 爲活化輸出)。

  一般而言，可使用 standard_output 函數提供此項功能。
- job==’getorigin’:此旗標指示函數傳回功能快左下方之座標值。
  - ▷ **arg1** :功能塊的資料結構。
  - ▷ **arg2** 未用。
  - ▷ **x** :功能塊左下方之 x 座標值。
  - ▷ **y** :功能塊左下方之 y 座標值。
  - ▷ **typ** 未用。

  一般而言，可使用 standard_origin 函數提供此項功能。
- job==’set’:此旗標指示函數開啓對話盒以取得功能塊之內部參數。
  - ▷ **arg1** :功能塊的資料結構。
  - ▷ **arg2** 未用。
  - ▷ **x** :功能塊的新的資料結構。
  - ▷ **y** 未用。
  - ▷ **typ** 未用。
- job==’define’:此旗標指示函數傳回功能塊之初始值及運算函數之結構 (名稱、型態、輸出入埠之數目及形態，... 等)
  - ▷ **arg1, arg2**未用。
  - ▷ **x** :功能塊的資料結構。
  - ▷ **y** 未用。
  - ▷ **typ** 未用。

6.1.2 功能塊之資料結構及定義

每一 Scicos 功能塊是由 Scilab 之串列 (list) 資料結構來定義，如下：

```
list('Block',graphics,model,unused,GUI_function)
```

其中 GUI_function 爲介面函數的名稱 (字串)，graphics 是圖形之資料結構

```
graphics=..
  list([xo,yo],[l,h],orient,dlg,pin,pout,pcin,pcout,gr_i)
```

- **xo**:方塊原點的 x 座標
- **yo**:方塊原點的 y 座標
- **l**:方塊寬
- **h**:方塊高
- **orient**:布林值, 指定是否左右翻轉 (正常輸入在左方，輸出在右方)
- **dlg**: 字串向量，內含功能塊的參數名稱
- **pin**:向量， pin(i) 爲連結到第 i 常規輸入埠的數目， 0 代表未連結
- **pout**:向量， pout(i) 爲連結到第 i 常規輸出埠的數目， 0 代表未連結
- **pcin**:向量， pcin(i) 爲連結到第 i 活化輸入埠的數目， 0 代表未連結
- **pcout**:向量， pcout(i) 爲連結到第 i 活化輸出埠的數目， 0 代表未連結
- **gr_i**:字串向量，繪出圖示 (icon) 的 Scilab 指令

模擬所需的資訊內含在 model 結構內：

```
model=list(eqns,#input,#output,#clk_input,#clk_output,...
        state,dstate,rpar,ipar,typ,firing,deps,label,unused)
```

- **eqns**: 串列包含兩元素。第一元素爲運算函數的名稱 (fortran, C, 或 Scilab 函數)。第二元素爲一整數，代表運算函數的類型。運算函數類型基本上選定了函數參數的傳呼規則，此主題容後再敘。
- **#input**:大小爲常規輸入埠總數的向量，每一項指定出各常規輸入埠的大小。若爲負整數時，代表由 Scicos 編譯器決定輸入埠的大小。若不同輸入埠對應到相同的負整數時，代表 Scicos 編譯器所決定之輸入埠大小也必須一樣。
- **#output**:大小爲常規輸出埠總數的向量，每一項指定出各常規輸出埠的大小。若爲負整數時，代表由 Scicos 編譯器決定輸出埠的大小。若不同輸出埠對應到相同的負整數時，代表 Scicos 編譯器所決定之輸出埠大小也必須一樣。

- **#clk_input**:大小爲活化輸入埠總數的向量，每一項數值都必爲 1，代表活化訊號必爲純量， Scicos 未支持向量之活化連結 (link)
- **#clk_output**:大小爲活化輸出埠總數的向量，每一項數值都必爲 1，代表活化訊號必爲純量， Scicos 未支持向量之活化連結 (link)
- **state**:連續狀態向量參數之初值 (列向量)。
- **dstate**:離散狀態向量參數之初值 (列向量)。
- **rpar**:傳送到運算函數內的實數參數向量 (列向量)。
- **ipar**:傳送到運算函數內的整數參數向量 (列向量)。
- **typ**:字串，代表功能塊之型態：'c' 代表連續功能塊 (CBB， Continuous Basic Block) , 'd' 代表離散功能塊 (DBB， Discrete Basic Block) , 'z' 代表跨零偵測功能塊 (ZBB， Zero Crossing Basic Block)， 'l' (小寫 L，注意不是's') 代表同步功能塊 (SBB ,Synchro Basic Block)
- **firing**:大小與活化輸出埠相同的列向量，代表活化開始的觸發時間，事先排程好的觸發狀態可以藉此參數設定 (<0 代表不觸發)
- **deps**: [udep timedep]
  - **udep**:布林數，眞 (true) 時，代表功能塊內部有前饋機制 (feed-through)，也就是至少一個輸出埠直接與輸入埠相關聯。
  - **timedep**:布林數，眞 (true) 時代表此功能塊爲隨時變化 (time dependent)。
- **label**:字串，功能塊名稱。這欄可以利用 Block 選單的 label 按鈕來設定。

6.2 運算函數

運算函數負責輸出訊號、新狀態值、連續狀態的導數量以及輸出事件的延時 (delay time) (timing) 向量。功能塊的類型以及 Scicos 模擬器呼叫它的方式會影響運算函數之計算結果。

6.2.1 行爲 (Behavior)

模擬器依照不同工作需求呼叫運算函數：

- **設定初值 (Initialization)** 每次開始模擬前，模擬器要求運算函數提供狀態及輸出訊號初始值給功能塊 (但輸入訊號還未得到)。另外，開啓檔案、圖形視窗啓動，... 等，也在這個階段進行。
- **重設初值 (Re-initialization)** 模擬器能夠直接重設初值，這是另一個改變狀態及輸出的機會。但此時輸入訊號也可能得到。
- **更新輸出訊號 (Outputs update)** 模擬器要求得到一新的輸出訊號。因此運算函數會被要求執行 (2) 式。

- **狀態更新 (States update)** 因應一個或多個事件產生，模擬器要求依照 (3) 及 (4) 式更新狀態 x 及 z。
- **計算狀態導數 (State derivative computation)** 功能塊在連續型活化狀態而外界要求取得 $\dot{x}$，因此運算函數必須計算 (1) 式。
- **輸出事件及延時 (delay time)(Output events timing)** 模擬器要求運算函數依照 (5) 式計算輸出事件及其延時 (delay time)。
- **結束 (Ending)** 模擬器在結束前呼叫運算函數一次 (通常進行關檔，清除記憶體,... 等)。

模擬器以設定旗標的方式，要球運算函數執行不同工作 (參考表 1)

Flag	Task
0	計算狀態導數 (State derivative computation)
1	更新輸出訊號 (Outputs update)
2	狀態更新 (States update)
3	輸出事件及延時 (delay time)(Output events timing)
4	設定初值 (Initialization)
5	結束 (Ending)
6	重設初值 (Re-initialization)

表 1: 運算函數功能及對應旗標編號

6.2.2 運算函數的類型

在 Scicos 中運算函數有幾種不同型態，但這幾種型態可以出現在同一方塊圖中。函數型態如表 2 所示。運算函數的型態設定資訊是存在 `eqns` 參數的的第二欄中， (請參考 6.1.2 節)。 Scicos 系統中編號大的爲較新定義的函數類型，因此定義運算函數時，最好使用型態 4, 5 而避免使用型態 0 之運算函數。

函數型態	Scilab	Fortran	C	說明
0	允許	允許	允許	函數參數數目固定
1	不允許	允許	允許	函數參數數目可變
2	不允許	不允許	允許	函數參數數目固定
3	允許	不允許	不允許	輸入、輸出是 Scilab 串列 (lists)
4	不允許	不允許	允許	輸入爲 C 之單一資料結構
5	允許	不允許	不允許	輸入爲與 C 資料結構對應的 Scilab 串列 (lists)

表 2: 不同型態運算函數。型態 0 不再建議使用。

6.2.2.1 運算函數:型態 0 型態 0 之功能塊中，模擬器建立單一之輸入儲存向量，並將實際上之輸入向量都堆到此單一向量中，同樣的所有的輸出向量，也推疊到單一之輸出儲存向量內。此型態目前只提供作爲向過去系統軟體相容之用。

此型態之函數相當於只有單一常規輸入及輸出之型態 1 函數。

6.2.2.2 運算函數:型態 1 介紹此型態最簡單的方法是以範例解釋：

已有兩個常規輸入及四個常規輸出埠之功能塊爲例，所需之函數原型如下：

6.2.2.3 Fortran 例

```
      subroutine myfun(flag,nevprt,t,xd,x,nx,z,nz,tvec,
     &     ntvec,rpar,nrpar,ipar,nipar,u1,nu1,u2,nu2,
     &     y1,ny1,y2,ny2,y3,ny3,y4,ny4)
c
      double precision t,xd(*),x(*),z(*),tvec(*),rpar(*)
      double precision u1(*),u2(*),y1(*),y2(*),y3(*),y4(*)
      integer flag,nevprt,nx,nz,ntvec,nrpar,ipar(*)
      integer nipar,nu1,nu2,ny1,ny2,ny3,ny4
```

函數各參數之說明可參考表 3。

6.2.2.4 C 例 型態 1 運算函數也可以以 C 撰寫，注意參數必須以指標方式傳遞 (passed as pointers)

學習撰寫運算函數最好的方式是透過研究 Scilab 次目錄 routines/scicos，裡面有完整的 Scicos 功能塊的程式碼，大部爲 Fortran 型態 0, 及型態 1。

6.2.2.5 運算函數:型態 2 型態 2 之運算函數只用於 C 語言。語法爲：

```
#include "<SCIDIR>/routines/machine.h"
void selector(flag,nevprt,t,xd,x,nx,z,nz,tvec,ntvec,
   rpar,nrpar,ipar,nipar,inptr,insz,nin,outptr,outsz,nout)

integer *flag,*nevprt,*nx,*nz,*ntvec,*nrpar;
integer ipar[],*nipar,insz[],*nin,outsz[],*nout;

double x[],xd[],z[],tvec[],rpar[];
double *inptr[],*outptr[],*t;
```

各參數之說明可參考表 4。

I/O	參數	說明
I	flag	0,1,2,3,4,5 or 6, (參考表 1)
I	nevprt	活化輸入埠編碼
I	t	時間
O	xdot	連續狀態之導數
I/O	x	連續狀態
I	nx	x 之大小
I/O	z	離散狀態
I	nz	z之大小
O	tvec	輸出事件之延時 (delay time) (for flag=3)
I	ntvec	活化輸出埠數目
I	rpar	實數參數向量
I	nrpar	rpar 大小
I	ipar	整數參數
I	nipar	ipar大小
I	ui	第 i 輸入 (常規), i=1,2,...
I	nui	第 i 輸入之大小
O	yj	第 j 輸出 (常規) , j=1,2,...
I	nyj	第 j 輸出之大小

表 3: 型態 1 運算函數之參數， I:輸入, O:輸出.

6.2.2.6 運算函數:型態 3

此型運算函數只適用於 Scilab。傳呼語法爲：

```
[y,x,z,tvec,xd]=test(flag,nevprt,t,x,z,rpar,ipar,u)
```

各參數之說明可參考表 5。

6.2.2.7 例

以下是一個簡單的功能塊之運算函數，此功能塊之離散狀態 z 記錄了被呼叫的次數，而每次呼叫時輸出 z 及兩個輸入參數值

```
function [y,x,z,tvec,xd]=test(flag,nevprt,t,x,z,rpar,ipar,u)
y=list();tvec=[];xd=[]
if flag==4 then
    z=0
elseif flag==2 then
    z=z+1
    write(%io(2),'Number of calls:'+string(z))
    [u1,u2]=u(1:2)
    write(%io(2),'first input');disp(u1)
    write(%io(2),'second input');disp(u2)
end
```

I/O	參數	說明
I	*flag	0,1,2,3,4,5 或 6, (參考表 1)
I	*nevprt	活化輸入埠編碼
I	*t	時間
O	xd	連續狀態之導數 (flag= 0)
I/O	x	連續狀態
I	*nx	x 之大小
I/O	z	離散狀態
I	*nz	z之大小
O	tvec	輸出事件之延時 (delay time) (flag=3))
I	*ntvec	活化輸出埠數目
I	rpar	實數參數向量
I	*nrpar	rpar 大小
I	ipar	整數參數
I	*nipar	ipar大小
I	inptr	inptr[i] 爲指向第 i 輸入向量開始位置之指標
I	insz	insz[i] 爲第i 輸入向量之大小
I	*nin	輸入埠 (常規) 總數
I	outptr	outptr[j] 爲指向第 j 輸出向量開始位置之指標
I	outsz	outsz[j] 爲第j 輸出向量之大小
I	*nout	輸出埠 (常規) 總數

表 4: 型態 2 運算函數之參數， I:輸入, O:輸出.

6.2.2.8 例 使用串列作爲輸入及輸出參數的好處是，輸入向量及輸出向量之數目不必事先固定。此例，輸出向量是對所有輸入向量進行一逐元素相乘，不必管實際數入多少向量。

```
function [y,x,z,tvec,xd]=elemprod(flag,nevprt,t,x,z,rpar,ipar,u)
tvec=[];xd=[]
y=u(1)
for i=2:length(u)
    y=y.*u(i)
end
y=list(y)
```

6.2.2.9 運算函數:型態 4 型態 4 之運算函數只用於 C 語言，爲型態 2 之改良型，語法爲：

```
#include "scicos_block.h"
#include <math.h>
void my_block(scicos_block *block,int flag)
```

I/O	參數	說明
I	flag	0,1,2,3,4,5 或 6, (參考表 1)
I	nevprt	活化輸入埠編碼 (純量)
I	t	時間 (純量)
I	x	連續狀態 (向量)
I	z	離散狀態 (向量)
I	rpar	參數向量 (任意 Scilab 變數)
I	ipar	參數 (向量)
I	u	u(i) 是第 i 個常規輸入向量 (串列)
O	y	y(j) 是第 i 個常規輸出向量 (串列)
O	x	更新的連續狀態 x (若 flag=2, 4, 5 或 6)
O	z	更新的離散狀態 z (若 flag=2, 4, 5 或 6)
O	xd	x 的導數若 flag=0 (vector), =[ ] 其他狀況
O	tvec	輸出事件之延時 (delay time)，若 flag=3 (vector), =[ ] 其他狀況

表 5: 型態 3 運算函數之參數， I:輸入, O:輸出.

```
{
...
}
```

其中 flag 爲 0,1,2,3,4,5 或 6, (參考表 1)。 scicos_block 爲 C 之資料結構，內含下列各欄 (field)

```
typedef struct {
 int nevprt;    /*活化輸入埠編碼, -1 代表內部活化(internally activated) */
 voidg funpt;   /* 運算函數指標 */
 int type;      /* 函數類型, 目前固定為 4 */
 int scsptr;    /* C 介面函數未使用 */
 int nz;        /* 離散狀態之大小*/
 double *z;     /* 離散狀態向量指標 */
 int nx;        /* 連續狀態之大小 */
 double *x;     /* 連續狀態向量指標*/
 double *xd;    /* 連續狀態導數向量指標 */
 double *res;   /* 只用於內部隱式功能塊(implicit blocks)，大小為 nx */
 int nin;       /* 輸入埠(常規)總數 */
 int *insz;     /* 各輸入埠(常規)大小 */
 double **inptr; /* 輸入埠(常規)指標表 */
 int nout;      /* 輸出埠(常規)總數 */
 int *outsz;    /* 各輸出埠(常規)大小 */
 double **outptr;/* 輸出埠(常規)總數 */
```

```
 int nevout;  /* 活化輸出埠數目 */
 double *evout; /* 輸出事件之延時(delay time) */
 int nrpar;  /* 實數參數數目 */
 double *rpar; /* 實數參數向量 */
 int nipar;  /* 整數參數數目 */
 int *ipar;  /* 整數參數向量 */
 int ng;      /* 跨零變號數目(zero-crossing surfaces)*/
 double *g;  /* 跨零變號向量(zero-crossing surfaces) */
 int ztyp;   /* 布林數, 若功能塊中有跨零變號條件向量時為真 */
 int *jroot; /* 大小為 ng 之向量，代表跨零變號之方向(direction of crossings) */
 char *label; /* 功能塊名稱 */
 void **work; /* 指向被此功能塊動態配置(allocation)的工作區 */
 int nmode;  /* number of modes ?*/
 int *mode;  /* mode vector of size nmode ? */
} scicos_block;
```

以下 C 函數可以在型態 4 之運算函數內呼叫，以取得其他資訊：

double get_scicos_time() 傳回目前時間。

void set_block_error(int) 通知模擬器，功能塊產生錯誤。

int get_phase_simulation() 傳回模擬器運作階段 (phase, 1 或 2)。

int get_block_number() 傳回功能塊編號。

6.2.2.10 運算函數:型態 5

型態 5 之運算函數只用於 Scilb 語言，語法爲：

```
block=func_name(block,flag)
```

其中串列block 對應到型態 4 之 C 資料結構 scicos$_b lock(field)$

以下 Scilab 函數可以在型態 5 之運算函數內呼叫，以取得其他資訊：

scicos_time() 傳回目前時間。

set_blockerror(i) 通知模擬器，功能塊產生錯誤。

phase_simulation() 傳回模擬器運作階段 (phase, 1 或 2)。

curblock() 傳回功能塊編號。

6.2.2.11 例

以下爲型態 4 之運算函數範例，功能爲進行累加運算。

```
#include "scicos_block.h"
#include <math.h>
```

```
void summation(scicos_block *block,int flag)
{
 int j,k;
 if(flag==1){
 if (block->nin==1){
  block->outptr[0][0]=0.0;
  for (j=0;j<block->insz[0];j++) {
   block->outptr[0][0]=block->outptr[0][0]+block->inptr[0][j];
  }
 }
 else {
  for (j=0;j<block->insz[0];j++) {
   block->outptr[0][j]=0.0;
   for (k=0;k<block->nin;k++) {
    if(block->ipar[k]>0){
     block->outptr[0][j]=block->outptr[0][j]+block->inptr[k][j];
    }else{
     block->outptr[0][j]=block->outptr[0][j]-block->inptr[k][j];
    }
   }
  }
 }
 }
}
```

6.2.2.12 例 以下爲型態 5 之運算函數範例，功能爲數學 *sin* 運算。

```
function block=sin5(block,flag)
 if flag==1 then
  for j=1:block.insz(1)
   block.outptr(1)(j)=sin(block.inptr(1)(j));
  end
 end
endfunction
```

7 結語

本文簡單介紹 Scicos 及其用法， Scicos 之線上文件提供其他詳細資訊。 Scicos 套件所提供的範例也是有趣的研究資料。通常，啓動現成的範例並編輯改寫、測試是比從一完全空白的方塊圖從頭開始來的容易些。