SCICOS - 力学系の構築およびシミュレータ ユーザーズ・ガイド*

R. Nikoukhah　　S. Steer

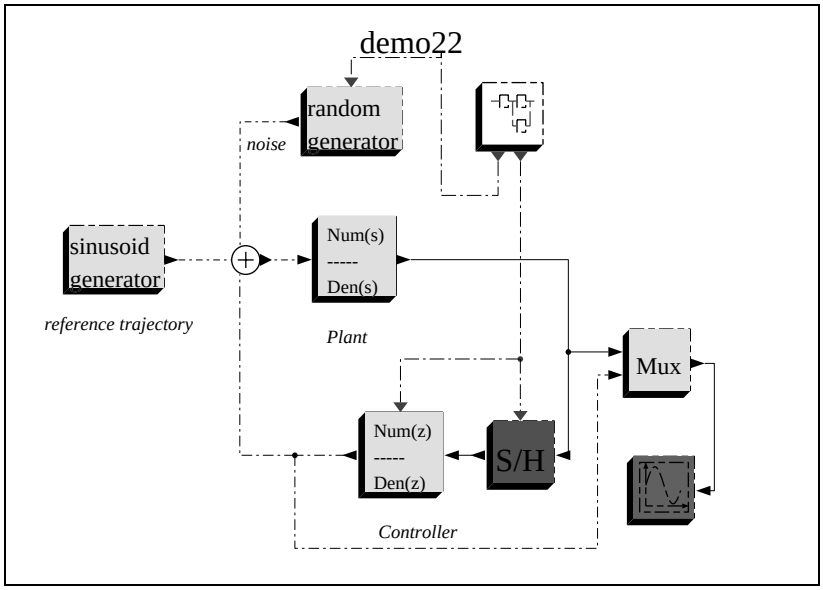

図 1: Scicos ダイアグラムの一例

1 Introduction

Scicos (Scilab Connected Object Simulator) は、連続系および離散系を含む力学系のモデリングおよびシミュレーション用の Scilab パッケージです。Scicos は、(定義済みの基本関数やユーザー定義の関数を表す) ブロックを相互結合することによりモデルを構築するためのグラフィカルエディターを備えています。

Scicos において各信号は信号を評価する時間である起動時間という名前の時間配列と関連づけられています。起動時間以外では、Scicos の信号は一定です。(図 2 参照) 起動時間は、一連の時間間隔とイベントと呼ばれる独立した点の組からなります。

Scicos における信号は、起動信号に駆動されるブロックにより生成されます。起動信号により入力および内部状態量の関数としてブロックの出力が評価されます。出力信号は、信号生成ブロックからの起動時間を継承しており、他のブロックを駆動するために使用することができます。

ブロックは、ブロック最上部にある入力起動ポートに入力された起動信号により起動されます。入

*Scicos は、Scilab ツールボックスです。Scicos のこの版は、Scilab-2.4 に含まれています。Scilab に関する詳細は、`http://www-rocq.inria.fr/scilab/`を参照下さい。

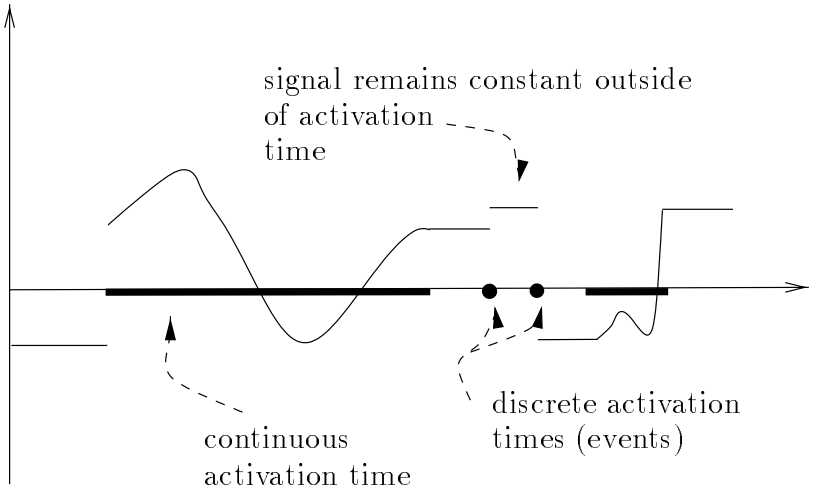

図 2: Scicos の信号と起動時間

力起動ポートがないブロックは、常にアクティブ (時間依存と呼ばれます) であり、一連の入力信号の起動時間に起動時間を同期します。

ブロックの最下部にあるポートは、出力起動ポートです。出力される信号は、ブロックにより生成された起動信号です。例えば、`Clock` ブロックは、等時間間隔で一連の起動信号を生成します。この出力を (`MScope` ブロックのような) スコープブロックの入力起動ポートに接続することにより、スコープに入力される値を表示するタイミングを指定することができます。

2 Scicos エディタ

Scicos において系は、ブロックおよびサブブロック (スーパーブロック) によりモデル化されます。サブブロックは、種々のパレットまたはユーザーにより定義することができます。Scicos は、ダイアグラムを容易に編集できる GUI を備えています。エディタを起動するには、Scilab で `scicos();` と入力して下さい。これにより Scicos のメインウインドウが表示されます。

Scicos におけるモデル構築は通常以下の手順からなります。

- (`Edit` メニューの `Palettes` ボタンを用いて) 一つ以上のパレットをオープンします。
- パレットからブロックを Scicos のメインウインドウにコピーします。コピーするには、`Edit` メニューの `copy` ボタンを選択し、続いてコピーするブロックをクリック、最後に Scicos メインウインドウのコピーする場所をクリックします。
- `Edit` メニューの `link` メニューを選択、出力ポートと入力ポート (または中間点) をクリックすることにより、入力および出力ポートを接続します。

(リンクを分岐するために) 既存のリンクから他のリンクを作成するには、まず `Link` ボタンをクリック、続いて既存のリンクの分岐する場所をクリック、最後に入力ポート (または中間点) をクリックします。マウスの右ボタンをクリックすることによりリンク作成のプロセスは停止し、現在のリンクは削除されます。

シミュレーション結果を可視化または保存するために、Scicos ダイアグラムに少なくとも一個のスコープまたは 「write to file」ブロックを置く必要があることにも注意して下さい。例については、Scicos デモを参照下さい。

2.1 パラメータ調整

ブロックのパラメータは、ブロックダイアログをオープンすることにより修正することができます。これは、Open/set により行うことができます。多くのブロックはダイアログメニューを備えており、ブロックのパラメータを設定・修正するために使用することができます。これらのパラメータは、有効なScilab 式を用いて定義することができます。Scilab 変数をこれらの式を定義する際に使用することが出来ます。これらの式は数式で記憶され、評価されます。

ダイアログの内容は、Context ボタンを選択することにより編集することができます。その内容は、Eval ボタンにより評価されます。内容修正により既存のブロックパラメータの定義に用いられている変数の値が変更される場合にのみ、評価を行う必要があります。

2.2 シミュレーション

完成したダイアグラムは、Simulate メニューの Run によりシミュレーションを行うことができます。このボタンを選択することにより、(まだコンパイルされていない場合は) ダイアグラムがコンパイルされ、シミュレーションが行われます。シミュレーションはScicos メインウインドウの最上部にある stop ボタンをクリックすることにより中断することができます。

コンパイルされたScicos ダイアグラムは、*.cos ファイルとして保存されます。保存されたファイルは、コンパイルの結果を保持しており、次回ロードされた際に再度コンパイルをする必要はありません。シミュレーション実行に必要なデータのみを抽出することも可能です。これにより、Scicos 環境に入らずにシミュレーションを行うことが可能です。これは、scicosim 関数により行うことができます。

2.3 その他の機能

その他、エディタは次のような多くの機能を提供します。

- さまざまなフォーマットでダイアグラムを保存・ロードする
- 視野のズームおよび変更
- ブロックの外観や色の変更
- ダイアグラムの背景や線の色の変更
- ダイアグラムにテキストを表示
- Scicos ダイアグラムを印刷およびエクスポートする
- その他の多くの標準的 GUI 機能

Help ボタンは、Scicos のさまざまな場面でヘルプを得るために使用することができます。Help を選択してから、ブロックをクリックすることにより、そのブロックのマニュアルページが表示されます。Help を選択してから、他のボタンを選択することにより、ボタンのマニュアルページを表示します。

最後に、Scicos の重要な機能は、サブシステム (Super Blocks) を作成できることです。一つのScicos ダイアグラムに数百もの要素を有する複雑な系を構築することは明らかに好ましくありません。このため、Scicos はブロックをグループ化し、Super Blocks という名前のサブダイアグラムを定義する機能を提供しています。これらのブロックは、他のブロックのように動作しますが、含むことができるブロックの数は無制限であり、他の Super Blocks をを含むこともできます。

3 基本ブロック

Scicos には、標準基本ブロック、ゼロクロス基本ブロック、同期基本ブロックの 3 種類の基本ブロックがあります。これらのブロックは、2 種類の入力ポート、標準入力、起動入力と 2 種類の出力ポート、標準出力と起動出力を有することができます。標準入力および出力は標準リンクと、起動入力および出力は起動リンクにより相互結合されます。起動入力ポートは、ブロックの最上部に、起動出力ポートは最下部にあることにあることに注意して下さい。

3.1 標準基本ブロック

標準基本ブロックは、**連続状態量** x および **離散状態量** z を有しています。連続状態量 x を有し、u が標準入力を定義すると、時間間隔を通過しブロックがアクティブになった際に、x は次の式に基づき連続量として更新されます。

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

ただし、f はベクトルを返す関数、p は定数パラメータのベクトルです。n_e は **起動コード** であり、ブロックを起動するポートを指定する整数です。例えば、起動入力ポートが $i_1, i_2, \ldots, i_n$ の場合、次のようになります。

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

一方、イベントが起動すると、状態変数 x および z は次の方程式に基づき瞬間的にジャンプします。

$$x(t_e) = g_c(t_e, x(t_e^-), z(t_e^-), u(t_e), p, n_e) \tag{2}$$

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

ただし、t_e はイベント時間を定義します。 離散状態量 z は二つの連続的イベントの間で一定であり、このため、$z(t_e^-)$ は z の値として解釈することができます。

起動時間の間、ブロックの標準出力は次の式により定義されます。

$$y(t) = h(t, x(t^-), z(t^-), u(t), p, n_e) \tag{4}$$

そして、ブロックがアクティブでない時は一定です。

標準基本ブロックはイベント型の起動信号を生成することが可能です。時間 t_e におけるイベントにより起動された場合、各出力イベントの起動時間は次のように定義されます。

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

ただし、t_{evo} は時間ベクトルであり、その各要素は各起動出力ポートに対応します。特定のイベントが無効の場合、現在時間よりも小さな時間となります。イベント生成は事前にスケジュール化することも可能です。イベント出力ポートを有するブロックの初期活性化変数を設定することによりイベントを事前にスケジュール化することができます。

3.2 ゼロ交差基本ブロック

ゼロ交差基本ブロックは、少なくとも標準入力の一つがゼロと交差する (符号を変える) 場合にのみイベント出力を生成することが可能です。この場合、イベントの生成とタイミングは、ゼロと交差する前後の入力の符号の組み合わせに依存させることができます。

ゼロ交差基本ブロックの例は、`Threshold` パレットにあります。

3.3 同期基本ブロック

同期基本ブロックは、入力起動信号と同期した出力起動信号を生成します。これらのブロックは、起動入力ポートを一つだけ有しています。入力起動信号は、起動出力信号の一つに接続されています。出力の選択は、標準入力の値によります。例としては、`event select` ブロックおよび `Branching` パレットの `If-then-else` ブロックがあります。

4 継承の時間依存性

Scicos ダイアグラムで全ての起動信号を陽に描くことを避けるために Scicos により継承という機能が提供されています。

ブロックに起動入力ポートがない場合、標準入力信号からの起動信号が継承されます。常にアクティブなブロックについては、「時間依存」と定義することが出来、入力起動ポートを必要としません。時間依存ブロックは継承されないということに注意して下さい。

5 ブロックの構築

新規ブロックは、(基本ブロックを相互結合することにより) Super Block として構築することが可能です。これは、新規の基本ブロックと同様に次の関数により定義されます。

- ユーザーインターフェースを処理する **インタフェイス** 関数
- 力学的運動を指定する **演算** 関数

インタフェイス 関数は常に Scilab 関数として書かれます。例としては、`<SCIDIR>/macros/scicos_blocks` の Scilab 関数を参照下さい。**演算** 関数は、C または Fortran で記述することも可能です。例えば、`<SCIDIR>/routines/scicos` を参照下さい。しかし、Scilab 言語で書くことも可能です。動的または Scilab に静的に結合されたにリンクされた C および Fortran ルーチンは、シミュレーション実行時間に関する限り、より良い結果を与えます。

`Scifunc`, `GENERIC`, `C_block`, `Fortran_block` ブロックは、**インタフェイス** 関数の雛形で、ユーザー開発の **演算** 関数をプロトタイプを素早く作成、試験する際には非常に便利です。

5.1 インタフェイス 関数

インタフェイス 関数は、状態量およびパラメータの初期値に加えて、ジオメトリ、カラー、ポートの数および大きさ、アイコン、等... を定義します。この関数は、ブロックのユーザーダイアログも処理します。

インターフェース関数の動作と返り値は、入力フラグ `job` に依存します。構文は、次のようになります。

5.1.1 構文

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

パラメータ

- job=='plot': ブロックを描画する関数
  - arg1 はブロックのデータ構造です。
  - arg2 は使用されていません。
  - x,y,typ は使用されていません。

  通常、長方形ブロックおよび入出力ポートを描画する standard_draw 関数を使用することができます。ブロックのサイズ、アイコン、カラーも処理します。

- job=='getinputs': この関数は位置及び入力ポートの型 (標準または起動) を返します。
  - arg1 はブロックのデータ構造です。
  - arg2 は使用されていません。
  - x は入力ポートの x 座標ベクトルです。
  - y は入力ポートの y 座標ベクトルです
  - typ は入力ポート型のベクトルです。(1：標準、2：起動)

  通常、standard_input 関数を使用することが可能です。

- job=='getoutputs': 位置及び出力ポートの型 (標準および起動) を返します。
  - arg1 は、ブロックのデータ構造です。
  - arg2 は使用されていません。.
  - x は出力ポートの x 座標ベクトルです。
  - y は出力ポートの y 座標ベクトルです。
  - typ は出力ポート型のベクトルです。

  通常、standard_output 関数を使用することが可能です。

- job=='getorigin': はブロックの輪郭を含む矩形の左下の座標を返します。
  - arg1 はブロックのデータ構造です。
  - arg2 は使用されていません。.
  - x は出力ポートの x 座標ベクトルです。
  - y は出力ポートの y 座標ベクトルです。
  - typ は使用されていません。.

  通常、standard_origin 関数を使用することが可能です。

- job=='set': ブロックパラメータ取得用にダイアログをオープンします。
  - arg1 はブロックのデータ構造です。
  - arg2 は使用されていません。.
  - x はブロックの新規データ構造です。

- y は使用されていません。.
- typ は使用されていません。.

- job=='define': ブロックのデータ構造 (対応する**演算** 関数の名前、入出力の型、数、サイズ、等) の初期化。
  - arg1, arg2 は使用されていません。.
  - x はブロックのデータ構造です。
  - y は使用されていません。.
  - typ は使用されていません。.

5.1.2 ブロックデータ構造の定義

各Scicos ブロックは、次のようなScilab データ構造により定義されます。

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

ただし、 GUI_function は対応する **インタフェイス** 関数の名前です。graphics はグラフィック用データを有する構造体です。

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

- **xo**: ブロック原点の x 座標
- **yo**: ブロック原点の y 座標
- **l**: ブロックの幅
- **h**: ブロックの高さ
- **orient**: ブロックが左右反転 (標準入力が左または右にある) しているかどうかを指定する論理値。
- **dlg**: 文字列 ベクトルであり、ブロックの数式パラメータを含みます。
- **pin**: ベクトル pin(i) は、i 番目の標準入力ポートに接続されたリンクの数、または、ポートに接続されていない場合に 0。
- **pout**: ベクトル pout(i) は、i 番目の番目の標準出力ポートに接続されたリンクの数、または、ポートに接続されていない場合に 0。
- **pcin**: ベクトル pcin(i) は、i 番目の起動入力ポートに接続されたリンクの数、または、ポートに接続されていない場合に 0。
- **pcout**: ベクトル pcout(i) は、i 番目の起動出力ポートに接続されたリンクの数、または、ポートに接続されていない場合に 0。
- **gr_i**: アイコンを描くために用いるScilab 命令を含む文字列ベクトル

model はシミュレーション情報を含むデータ構造であり、次のようになります。

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

- **eqns**: 二つの要素を有する リスト。最初の要素は、**演算** 関数 (fortran, C, Scilab 関数) の名前を有する文字列です。二番目の要素は、**演算** 関数の型を指定する整数です。**演算** 関数の型は、コールの基本手順を指定します。詳細は後述します。
- **#input**: ブロックの標準入力ポートの数に等しい大きさのベクトル。各エントリは、対応する入力ポートの大きさを指定します。負の整数は、「コンパイラにより定義される」ことを意味します。同じ負の整数を一つ以上の入出力ポートに指定することにより、コンパイラにこれらのポートが同じ大きさを有していることを伝えます。
- **#output**: ブロックの標準出力ポートの数に等しい大きさのベクトル。各エントリは、対応する出力ポートの大きさを指定します。同じ負の整数を一つ以上の入出力ポートに指定することにより、コンパイラにこれらのポートが同じ大きさを有していることを伝えます。
- **#clk_input**: ブロックの起動入力ポートの数に等しい大きさのベクトル。全てのエントリは 1 に等しい必要があります。Scicos は、ベクトル化された起動リンクをサポートしません。
- **#clk_output**: ブロックの起動出力ポートの数に等しい大きさのベクトル。全てのエントリは 1 に等しい必要があります。Scicos は、ベクトル化された起動リンクをサポートしません。
- **state**: 連続状態量の初期値の列ベクトル。
- **dstate**: 離散状態量の初期値の列ベクトル。
- **rpar**: 対応する **演算** 関数に渡された実数のパラメータを有する列ベクトル。
- **ipar**: 対応する **演算** 関数に渡された整数のパラメータを有する列ベクトル。
- **typ**: 基本ブロック型を表す文字列。ゼロ交差基本ブロックの場合は'z'、同期基本ブロックの場合は'l'、標準基本ブロックのようなそれ以外のブロックは's'。
- **firing**: 初期起動時間の列ベクトルであり、ブロックの起動出力ポートの数に等しい大きさを有しています。これは、事前に設定したイベント起動時間を有しています。(起動しない場合は <0)
- **deps**: [udep timedep]
  - **udep**: 論理値。系が直達項、つまり、少なくとも出力の一つが陽に入力の一つに依存している場合に TRUE。
  - **timedep**: 論理値。ブロックが時間に依存する場合に TRUE.
- **label**: ブロック ID に用いる文字列。このフィールドは Block メニューの label ボタンにより設定されます。

5.2 演算 関数

演算 関数は、出力、新規の状態変数、連続状態変数の微係数およびブロックの型およびシミュレータからの呼び出し方依存する出力イベントタイミングベクトルを評価します。

5.2.1 動作

シミュレータは、異なったタスクを実行するために**演算** 関数をコールします。

- **初期化**シミュレータは、スタート時に状態変数および (入力はこの時点で利用可能ではないため、) 出力を初期化するために **演算** 関数を一度コールします。ファイルのオープン、グラフィックウインドウの初期化、等といった他のタスクもこの時点で行うことが可能です。
- **再初期化**シミュレータは、再初期化用に複数回ブロックをコールすることができます。これにより、再び状態変数や出力を初期化することが可能です。この時には入力も初期化することができます。
- **出力の更新**シミュレータは、出力の値を呼び出します。このため、**演算** 関数は (4) を評価する必要があります。
- **状態変数の更新** 一つ以上のイベントが生じた場合、シミュレータが**演算** 関数をコールし、(2) および (3) に基づき状態変数 x および z を更新します。
- **状態変数の微係数の計算**シミュレータが連続系の場合、ソルバーは $\dot{x}$ を必要とします。このため、**演算** 関数は (1) を評価する必要があります。
- **出力イベントのタイミング**シミュレータは、出力イベントが生じたタイミングで**演算** 関数をコールします。**演算** 関数は、 (5) を評価する必要があります。
- **終了処理**シミュレータは、終了時に一度 **演算** 関数をコールします。(ファイルを閉じたり、確保したメモリーを解放したりといった用途に有用です)

シミュレータは、計算するべきタスクを指定するためにフラグを使用します。(表 1 を参照下さい)

フラグ	タスク
0	状態変数の微係数の計算
1	出力の更新
2	状態変数の更新
3	出力イベントのタイミング
4	初期化
5	終了処理
6	再初期化

表 1: **演算** 関数のタスクと対応するフラグ

5.2.2 演算 関数の型

Scicos では、**演算** 関数は同じダイアグラムに異なった型を置き、共生させることができます。現在定義されている型を表 2 に示します。**演算** 関数の型は `eqns` の二番目のフィールドに保存されています。(5.1.2 節を参照下さい)

関数の型	Scilab	Fortran	C	コメント
0	yes	yes	yes	固定コール手順
1	no	yes	yes	可変コール手順
2	no	no	yes	固定コール手順
3	yes	no	no	入出力を Scilab リストとする

表 2: **演算** 関数の型一覧。0 型は古いです。

演算 関数: 0 型　0 型のブロックは、シミュレータは全ての入力ベクトルを集めて一つの入力ベクトルを作成し、同様に単一のベクトルとして出力を求めます。この型は下位互換性のためのみにサポートされています。

コール手順は、一つの標準入力と一つの標準出力を有する 1 型の **演算** 関数の手順と同じです。

演算 関数: 1 型　この型を説明する最も簡単な方法として 2 つの標準入力ベクトルと 4 つのブロックの標準出力ベクトルを有するブロックについて考えてみましょう。**演算** 関数は、ほぼ次のようになります。

Fortran の場合

```
      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
```

引数の説明について表 3 を参照下さい。

C の場合　同様に 1 型の **演算** 関数を C 言語でも書く事ができます。引数はポインタとして渡す必要があることに注意して下さい。

この関数を書き方を学ぶ最良の方法は、Scilab ディレクトリ SCIDIR/routines/scicos のルーチンを見ることです。ここには、全ての Scicos ブロックの **演算** 関数があります。その多くは、fortran 0 型および 1 型です。

演算 関数: 2 型　この **演算** 関数の型は C でプログラムする必要があります。概要を以下に示します。

```
#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;
```

I/O	引数	説明
I	flag	0,1,2,3,4,5,6, (表 1 参照)
I	nevprt	起動コード
I	t	時間
O	xdot	連続状態変数の微係数
I/O	x	連続状態変数
I	nx	x の大きさ
I/O	z	離散状態変数
I	nz	z の大きさ
O	tvec	(flag=3 の) 出力イベントの時間
I	ntvec	起動出力ポートの数
I	rpar	パラメータ
I	nrpar	rpar の大きさ
I	ipar	パラメータ
I	nipar	ipar の大きさ
I	ui	i 番目の入力 (標準) i=1,2,...
I	nui	i 番目の入力の大きさ
O	yj	j 番目の出力 (標準) j=1,2,...
I	nyj	j 番目の出力の大きさ

表 3: 1 型の **演算** 関数の引数。I: 入力, O:出力

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

引数の説明については表 4 を参照下さい。

演算 関数 3 型　この **演算** 関数の型は、Scilab でプログラミングする必要があります。コール手順は次のようになります。

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

引数の説明については表 5 を参照下さい。

例　以下は、イベントを受ける度毎に Scilab ウインドウに現在までに受けたイベントの数と二つの入力の値を表示するブロックに関連する**演算** 関数の例です。

```
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)
```

I/O	引数	説明
I	*flag	0,1,2,3,4,5,6, (表 1 を参照下さい)
I	*nevprt	起動コード
I	*t	時間
O	xd	連続状態変数の微係数 (flag= 0)
I/O	x	連続状態変数
I	*nx	x の大きさ
I/O	z	離散状態変数
I	*nz	z の大きさ
O	tvec	出力イベントの時間 (flag=3)
I	*ntvec	起動出力ポート数
I	rpar	パラメータ
I	*nrpar	rpar の大きさ
I	ipar	パラメータ
I	*nipar	ipar の大きさ
I	inptr	inptr[i] は i 番目の入力の先頭へのポインタです。
I	insz	insz[i] は、i 番目の入力の大きさです。
I	*nin	入力ポートの数
I	outptr	outptr[j] は j 番目の出力の先頭へのポインタです。
I	outsz	outsz[j] は j 番目の出力の大きさです。
I	*nout	出力ポートの数

表 4: 2 型の **演算** 関数の引数 I: 入力, O: 出力

```
    write(%io(2),'first input');disp(u1)
    write(%io(2),'second input');disp(u2)
end
```

例　入出力をリストとしてコード化する利点は、陽に入出力の数を指定する必要がないことです。この例では、出力は入力の数によらず全ての入力ベクトルの要素毎の積です。

```
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　結論

このドキュメントでは、Scicos とその使用法のみを簡単に説明しています。詳細な情報は、Scicos 関数のマニュアルページ (Scicos ライブラリにある Scilab ヘルプ) にあります。Scilab で提供される

I/O	引数	説明
I	flag	0,1,2,3,4,5,6 (表 1 参照)
I	nevprt	起動コード (スカラー)
I	t	時間 (スカラー)
I	x	連続状態変数 (ベクトル)
I	z	離散状態変数 (ベクトル)
I	rpar	パラメータ (scilabtt 変数の型)
I	ipar	パラメータ (ベクトル)
I	u	u(i) は、i 番目の標準入力 (リスト) のベクトルです。
O	y	y(j) は、j 番目の標準出力 (リスト) のベクトルです。
O	x	flag=2, 4, 5 , 6 の場合、新規の x
O	z	flag=2, 4, 5 , 6 の場合、新規の z
O	xd	flag= 0 (ベクトル) の場合 x の微係数、それ以外は、[]
O	tvec	flag=3 (ベクトル) の場合、出力イベントの時間、それ以外は []

表 5: 3 型の **演算** 関数の引数 I: 入力, O: 出力

Scicos デモは、有益な情報源でもあります。空のダイアグラムから始めるよりも Scicos でもから始めて編集していく方が有利なことが多いです。

目 次