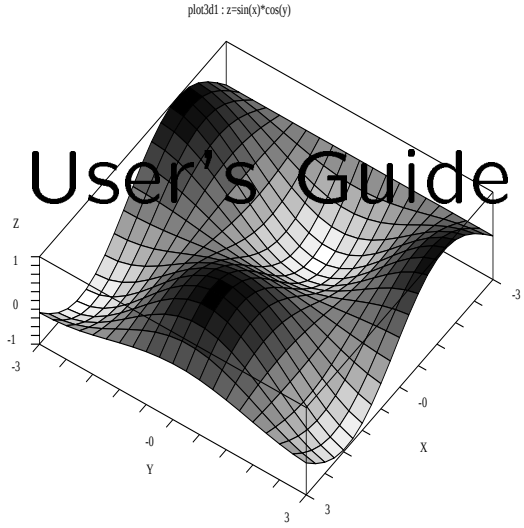
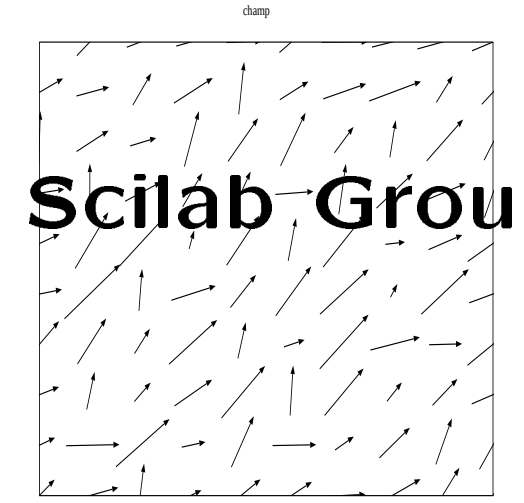
Introduction To Scilab

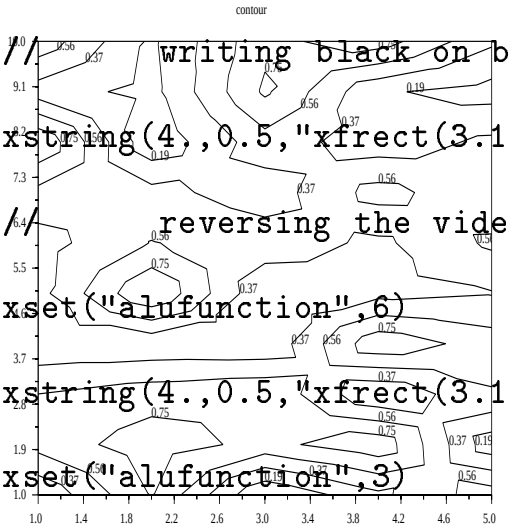
User's Guide



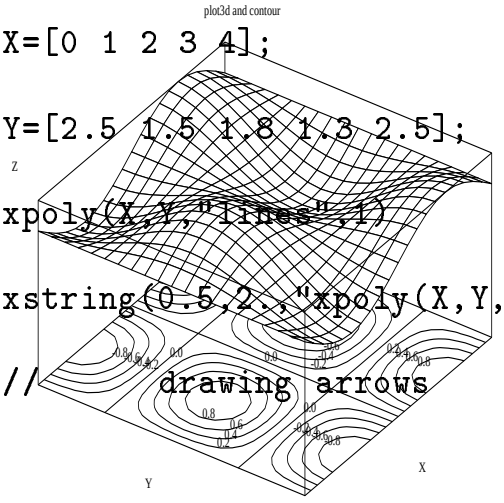


Scilab Group

```
-->plot(1:10)

-->xbasc()

-->//      simple rectangle

-->xrect(0,1,3,1)

-->//      filling a rectangle

-->xfrect(3.1,1,3,1)

-->//      writing in the rectangle

-->xstring(0.5,0.5,"xrect(0,1,3,1)")
```




```
-->//      writing black on black !

-->xstring(4.,0.5,"xfrect(3.1,1,3,1)")

-->//      reversing the video

-->xset("alufunction",6)

-->xstring(4.,0.5,"xfrect(3.1,1,3,1)")

-->xset("alufunction",3)

-->//      drawing a polyline

-->X=[0 1 2 3 4];

-->Y=[2.5 1.5 1.8 1.3 2.5];

-->xpoly(X,Y,"lines",1)

-->xstring(0.5,2.,"xpoly(X,Y,""lines""

-->//      drawing arrows
```



ΨLab 入門

Scilab Group
INRIA Meta2 Project/ENPC Cergrene

INRIA - Unité de recherche de Rocquencourt - Projet Meta2
Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)
E-mail : scilab@inria.fr

目 次

第1章　導入

1.1　Scilab とは何か

INRIA により開発された Scilab は、システムの制御および信号処理への適用を目的として開発されてきました。Scilab は、ソースコードの形式で自由に配布可能です。(ファイル `notice.tex` を参照して下さい。)

Scilab は、3 つの部分から構成されています。それらは、インタプリタ、関数ライブラリ (Scilab プロシージャ) 、Fortran と C のサブルーチンライブラリです。これらのサブルーチン (細かい話をすると、Scilab に属さずインタプリタから対話的に呼ばれるもの) はそれ自体が有益であり、ほとんどのものは Netlib から手にいれることができます。それらのごく一部は、Scilab インタプリタとの互換性を向上させるために一部を修正してあります。

MATLAB 構文の主な特徴は、その行列の演算能力です。つまり、結合、展開、転置のような基本行列操作は、加算や乗算といった基本演算と同様に簡易に実行可能であることです。Scilab の目的は次のようなものです。まず、数値行列より複雑なオブジェクトについて MATLAB 構文を使用することです。(例えば、自動制御系の設計者は、分数または多項式伝達行列に関する演算を行なうことを望むでしょう。) 第二に数値演算ライブラリ (特定のルーチンを Scilab から動的にコールしたり、新しい基本関数としてパッケージに含めることができます。) へのオープンなインターフェースとなることです。

Scilab は、(MATLAB に似た構文を有する) 対話的なインタープリタソフトウエアパッケージであり、次のような強力な機能を有しています。

- リスト
- 多項式および多項式行列の数式処理
- 線形または非線型系の数式処理
- 非線型計算: シミュレーション及び最適化
- Fortran または C 言語との簡便なインターフェース

リスト構造により、伝達関数や線形系といった複雑な数式オブジェクトの自然な数式表現が可能となります。(2.6 節を参照して下さい。)

多項式、多項式行列、伝達行列も定義されており、自然な数式処理風にこれらのオブジェクトの定義や操作が可能となります。(2.4 節を参照して下さい。) これらの行列を操作するための構文は、定数ベクトルや行列を操作するためのものと同一です。

Scilab は、非線形システムの解析用に広範で強力な基礎関数を提供します。陽的システムおよび陰的システムの積分は、数値的に実行することができます。数値最適化の機能としては、非線形最適化 (微分演算を行なえない系の最適化を含みます。)、2 次系最適化、線形系最適化があります。

Scilab は、オープンなプログラミング環境を有しており、関数とライブラリの作成は完全にユーザーの手で行なわれます。(3 章を参照して下さい。) 関数は、Scilab におけるデータオブジェクトとして認識されます。このため、他のデータオブジェクトのように演算を行なったり作成することが可能です。例えば、関数を引数として他の関数に渡すことが可能です。

加えて、Scilab は、文字列データ型をサポートします。この型は、関数の自動生成を可能にします。文字列の行列は通常の行列と同じ構文により操作することが可能です。また、Scilab は、Fortran または C のサブプログラムと容易にインターフェースを構築できます。これにより、Scilab のインタープリタ環境において標準化されたパッケージおよびライブラリの使用が可能となります。

Scilab の大局的な目的は、次のような計算環境を提供することです。

- 可変で柔軟なデータ型を有すること。
- 自然で使用が簡単な構文を有すること。
- 広範な計算の基礎関数として用いることが可能な関数群を提供すること。
- 新しい基礎関数を容易に付加できるようなオープンなプログラミング環境を有すること。
- 特定の応用分野 (線形制御、信号処理、ネットワーク解析、非線形制御、等) 向けに提供される「ツールボックス」によりライブラリ開発をサポートすること。

この入門マニュアルの目的は、Scilab により何ができるかをユーザに示すことです。全ての Scilab 関数についてオンラインドキュメントを見ることができます。(`help` コマンド)

1.2　ソフトウエアの構成

Scilab は、一連のディレクトリに分けられています。メインディレクトリ `SCIDIR` には、`scilab.star` (スタートアップ用ファイル) と copyright ファイル `notice.tex`、`configure` ((1.3) を参照) があります。次のようなサブディレクトリがあります。

- `bin` は実行ファイルのディレクトリです。Scilab の実行ファイルである `scilex` がここにあります。このディレクトリには、Scilab が作成した Postscript/LaTeX ファイルを管理したり印刷したりするためのシェルスクリプトがあります。
- `demos` は Scilab のデモがあるディレクトリです。`alldems.dem` ファイルには、“demo” をクリックすることにより実行可能な新しいデモを付加することが可能です。このディレクトリには、様々なデモに対応するコードがあります。これらは、新しいユーザーを元気づけるのに、しばしば役に立ちます。ほとんどのプロットコマンドが、簡単な例により説明されます。引数パラメータなしでグラフィック関数を実行すると、この関数の使用例が表示されます。(例えば、`plot2d()` は `plot2d` 関数の使用例を表示します。)
- `examples` には、動的リンクまたは `intersci` を使用して外部プログラムを scilab にリンクする方法の有用な例があります。
- `doc` は、Scilab ドキュメントのディレクトリです。LaTeX、dvi 、Postscript ファイルがあります。このドキュメントは、`SCIDIR/doc/intro/intro.tex` です。このディレクトリ `SCIDIR/man` にあるマニュアル (オンラインヘルプ) も参照して下さい。

- geci には、GeCI のソースコードと実行ファイルがあります。Geci は、リモート実行を管理し、ソフトウエア間でメッセージを交換することを可能にするために作られた対話型のコミュニケーションマネージャです。この Geci は、ネットワーク上にある多くのマシンを、分配された独立したソフトウエアのグループを作成することにより、仮想コンピューターとして活用する可能性を提供します。(詳細な説明については、help communications を参照して下さい。) GeCI は、Xmetanet と Scilab のリンクに用いられます。

- imp は、プリント用 Postscript ファイルを管理するルーチンのあるディレクトリです。

- libs には、Scilab ライブラリ (コンパイル済コード) があります。

- macros には、オンラインで入手可能な Scilab 関数のライブラリがあります。新しいライブラリは簡単に加えることが可能です。(Makefile を参照して下さい。) このディレクトリは、複数のサブディレクトリに分けられています。ここには、制御、信号処理、等に用いる “ツールボックス” があります。厳密に言うと、Scilab はツールボックスを系統立てていません。つまり、特定のサブディレクトリの関数は、他のディレクトリの関数をコールすることができます。よって、例えば、サブディレクトリ “signal” は自己完結せず、ここに含まれる関数は信号処理全般に用いられます。

- man は、マニュアル (Unix マニュアル) のあるディレクトリであり、サブマニュアルに分割されています。これらは、オンラインヘルプおよび LaTeX フォーマットの Scilab リファレンスマニュアルと同じものです。LaTeX コードは、Unix フォーマットの Scilab マニュアル (サブディレクトリ SCIDIR/man を参照して下さい。) から変換することにより作成されます。ある項目について情報を得たい場合には、Scilab において help 項目 と入力するか、ヘルプボタンにより使用可能なヘルプウインドウ機能を使用して下さい。あるキーワードに一致する関数を得たい場合には、apropos キーワード と入力するか、ヘルプウインドウの中で apropos を使用して下さい。help と apropos コマンドにより認識される全ての item と キーワード は、サブディレクトリ man にある .cat と whatis ファイルの中にあります。

  新しい項目を help と apropos コマンドに加えるためにユーザーは、ファイル SCIDIR/man/Chapters を編集することによりヘルプブラウザで利用可能なディレクトリのリストを拡張することが可能です。README ファイルを参照して下さい。

- maple ディレクトリには、Maple 関数のソースコードがあります。この関数により Maple オブジェクトを Scilab 関数に変換することが可能です。効率のために、変換の際には Scilab に動的にリンクされる Fortran コードが作成されます。

- routines ディレクトリには、全ての数値計算ルーチンのソースコードがあります。サブディレクトリ default には、Scilab をカスタマイズするのに有用なルーチンのソースコードがあり、重要なサブディレクトリです。特に、ODE/DAE シミュレーションまたは最適化用の C または Fortran ルーチンは、ここでインクルードすることが可能です。(これらも動的にリンクされます。)

- intersci は、新しい Fortran または C のプリミティブを Scilab に加える際に必要なインターフェースプログラムを作成するために提供されたプログラムを含んでいます。このプログラムは、bin/intersci ディレクトリにある intersci スクリプトにより実行されます。

- scripts ディレクトリには、シェルスクリプトファイルのソースコードがあります。Scilab により使用可能なプリンター名のリスト は、環境変数によりそこで定義されています。
- tests：このディレクトリには、Scilab のマシンへのインストールをテストするための評価プログラムがあります。ファイル “demos.tst” は、全てのデモをテストします。
- tmp:ユーザーによって書かれたいくつかの例がこのディレクトリに加えられています。
- util には、fortran ルーチンとして Scilab をコールしたり、ドキュメントを作成するための幾つかのユーティリティ関数があります。
- xless は、Berkeley 大学において開発されたファイル閲覧用ツールです。
- xmetanet ディレクトリには、xmetanet があります。これは、ネットワーク用のグラフィックディスプレイです。使用に際しては、Scilab 上で metanet() とタイプして下さい。

1.3　Scilab のインストール、要求されるシステム

Scilab は、ソースコードフォーマットで配布されます。幾つかの主要な Unix-XWindow システム用の実行ファイルも入手可能です。

Dec Alpha (OSF 3.0)、Dec Mips (ULTRIX 4.2)、Sun Sparc stations (Sun OS 4.1.3)、Sun Sparc stations (Sun Solaris 2.3)、 HP9000 (HP-UX 9.01)、SGI Mips Irix 5.2、IBM-RS6000 (AIX 3.2)、PC 486 Linux

インストール上の要求は、次のようになります。:

- ソースコード版について: Scilab は、unpack してインストールするために約 75Mb のディスク記憶と必要とします。(全てのソースコードが含まれています。) X Window (X11R4 または X11R5)、C コンパイラと Fortran コンパイラ (または、f2c) が必要です。X11R4 を実行している場合には Athena Widgets ライブラリ libXaw.a と libXmu.a. を必要とします。

- バイナリ版について: Scilab を実行するために最小限必要な容量は (ソース無しで) 圧縮しない場合で約 20 Mb です。Dec Alpha、Dec Mips、Sun OS、HP9000、IBM-RS6000 版は静的にリンクされており、基本的には fortran コンパイラを必要としません。Sun Solaris、SGI、PC Linux 版は動的にリンクされます。Scilab におけるメモリーの主な部分は、通常の Fortran の仕様と同じく積層化されています。Scilab の幾つかの部分 (特に、疎行列に関して) は、動的なアロケーションを用いています。積層 (パイル) のサイズとして 2Mega ワード (倍精度浮動小数) が選択されています。もちろん、ソースコード版では、ユーザーは簡単にこの大きさを変更し、(減らしたり、あるいは) コンピューターのメモリ量 (ファイル routines/stack.h のパラメータ vsiz) まで増加させたりすることが可能です。

1.4　Scilab の概要。チュートリアル

1.4.1　始めに

Scilab は、ディレクトリ SCIDIR/bin において scilab と入力することにより、実行されます。(SCIDIR は Scilab がインストールされたディレクトリを定義します。) このシェルスクリプトは、Scilab を Xwindow 環境で実行します。(このスクリプトファイルは、“ウインドウ無し” とするため

の-nw といった特定のパラメータを付けて呼び出すことができます。) すぐに Scilab ウインドウが現れ、次のバナーと --> で表されるプロンプトが表示されます。:

```
                            ===========
                            S c i l a b
                            ===========


                     Scilab-2.x  ( 10 February 1995 )
                     Copyright (C) 1989-95 INRIA


Startup execution:
  loading initial environment

 -->
```

Scilab との最初のコンタクトは、左マウスボタンで Demos をクリックし、つづいて Introduction to SCILAB をクリックすることにより行なわれます。セッションの実行はリターンを入力することにより行なわれ、Stop または Abort ボタンにより中断することが可能です。

いくつかのライブラリ (SCIDIR/scilab.star ファイルを参照して下さい。) は自動的にロードされます。

Scilab の幾つかの機能に関する考え方をユーザーに提供するために、Scilab のサンプルセッションを後に示します。

1.4.2　コマンド行の編集

サンプルセッションの前に、コマンド行の編集の仕方について簡単にまとめます。プロンプトの後に入力したり、ウインドウの一部をマウスでクリックすることによりコマンド行を入力することが可能です。また、Scilab ウインドウのプロンプトにおいてこのコマンド行を記憶させることが可能です。

このとき、コマンドを修正するために自由に古くからある Emacs コマンドを使用することができます。(Ctrl-<chr> は、CONTROL キーを押しつつ、文字 <chr> を入力することを意味します。) 例えば、

- Ctrl-p 前のコマンドを再コールする。
- Ctrl-n 次の行を再コールする。
- Ctrl-b １文字後ろへ移動する。
- Ctrl-f １文字前方へ移動する。
- Delete 前の文字を削除する。

- Ctrl-h 前の文字を削除する。
- Ctrl-d (カーソル上の) 一文字を削除する。
- Ctrl-a 行頭に移動する。
- Ctrl-e 行末に移動する。
- Ctrl-k 行末まで削除する。
- Ctrl-u 現在の行を取り消す。
- Ctrl-y 前に削除したテキストを張り付ける。
- !prev prev で始まる最後に実行したコマンド行を再びコールする。
- Ctrl-c Scilab を中断し、復帰後、ポーズします。(関数のみ中断できます。) stop ボタンをクリックすることにより、Ctrl-c が入力されます。

先に述べたようにマウスによりカットアンドペーストも可能です。この方法は、エディタで Scilab コマンドを入力する場合に便利でしょう。Scilab 文を含むファイルを “load” するには、`File Operations` ボタンを利用することも可能です。

1.4.3　ボタン

Scilab ウインドウには、次のようなボタンがあります。

- Stop 実行を中断し、ポーズモードに入ります。
- Resume コマンドまたは `Stop` ボタンにより行なわれたポーズの後に実行を継続します。
- Abort 1 回の (あるいは複数の)`pause` の後に実行を終了し、トップレベルプロンプトに戻ります。
- Restart 全ての変数を消去し、スタートアップファイルを実行します。
- Quit Scilab を終了します。
- Kill Scilab シェルスクリプトを kill します。
- Demos いくつかのデモを対話的に実行します。
- File Operations 関数やデータを Scilab にロードしたり、スクリプトファイルを実行する機能です。従来のリリースより次のような点が変更されていることに注意して下さい。従来は、このボタンを使用するとロードしたファイルのあるディレクトリにワーキングディレクトリが変更されました。この動作は混乱を生じる可能性があります。このボタンを使用してももはやワーキングディレクトリは変更されません。
- Help : ツリー状のマニュアルと対応する項目を有するオンラインヘルプを呼び出します。Scilab ウインドウで直接 `help <項目>` を入力することも可能です。

- +- : アクティブウインドウの数を増加、または減らします。
- Raise Window : 指定された数字に対応するウインドウを上にします。また、必要ならば一つまたは複数のウインドウを作成します。
- Set Window : 指定された数字に対応するウインドウをアクティブにします。(必要ならば、一つあるいは複数のウインドウを作成します。)

コマンド `SCIDIR/bin/scilab -nw` は、Scilab を “no-window” モードで起動します。

1.4.4 Scilab のカスタマイズ

多くのソフトウエアと同様に、Scilab により開かれた異なったウインドウのパラメータは簡単に変更できます。これを行なう方法は、X11-defaults サブディレクトリにあるファイルを編集することです。最初の方法は、これらのファイルを直接変更しますが、今後のリリースにおいても同様の修正を必要とすることになります。正しい方法は、修正を行なった行を自分のホームディレクトリの `.Xdefaults` ファイルにコピーすることです。この修正は Xwindow を再起動するか、コマンド `xrdb .Xdefaults` を実行することにより有効となります。Scilab は、`.Xdefaults` ファイルを読み込みます。このファイルの内容は、X11-defaults の対応する行をキャンセルしたり置き換えたりします。

簡単な例 :

```
Xscilab.color*Scrollbar.background:red
Xscilab*vpane.height: 500
Xscilab*vpane.width:  500
```

`.Xdefaults` において以上のように記述することにより、500x650 ウインドウは 500x500 の正方形ウインドウに変更され、スクロールバーの背景色は、緑から赤になります。

1.4.5 初心者用のサンプルセッション

続いて、いくつかの簡単なコマンドについて説明します。コマンドはセミコロンまたはリターン (CR) で終わります。リターンを入力すると、前の入力行が解釈された後の全てのコマンドが出力されます。入力行につけるセミコロンは任意です。

```
-->a=1;

-->A=2;

-->a+A
 ans  =

    3.

-->//Two commands on the same line
```

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

    1.5

-->//A command on several lines

-->u=1000000.000000*(a*sin(A))**2+2000000.000000*a*b*sin(A)*cos(A)+1000000.000000*(b*cos(A))**
 u  =

    81268.994

-->u=1000000.000000*(a*sin(A))**2+...
     2000000.000000*a*b*sin(A)*cos(A)+...
     1000000.000000*(b*cos(A))**2
 u  =

    81268.994
```

値 1 と 2 を変数 a と　A に与えます。コマンドの最後のセミコロンは、結果の出力を抑制します。Scilab は大文字小文字を区別するということに注意して下さい。2 つのコマンドが実行され、セミコロンが後ろについていない第 2 のコマンドの結果が表示されます。最後のコマンドは、“...” を用いてコマンドを複数行にわたって記述する方法を示しています。この記号は、オンライン入力時に改行の効果を避けるためにのみ必要です。

// に続く一連の文字列は、Scilab に解釈されません。(コメント行になります。)

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

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

    4.25

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

-->who
your variables are...

 ans       b         a         bugmes    %F        %T        TMPDIR
 SCI       scicoslib           xdesslib  utillib   tdcslib   siglib
```

```
 s2flib    roblib    percentlib          optlib    metalib   elemlib
 polylib   autolib   armalib   alglib    %z        %s        %nan
 %inf      %t        %f        %eps      %io       %i        %e
 %pi
 using    3065 elements  out of 1000000.
          and      34 variables out of     499
```

先に定義した変数 a b c A と別のライブラリといくつかの特定の “永続” 変数から構成される初期環境のリストを同時に表示します。

定数を既存の変数と混用する式の例を以下に示します。結果は、標準のデフォルト変数 ans に保持されます。

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

!   2.    2.i !
```

ベクトルの引数を付けて関数 (あるいは、基関数) を呼び出します。返り値は、複素数ベクトルです。

```
-->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
```

多項式を作成する更に複雑なコマンドを示します。

```
-->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
```

多項式行列の定義です。多項式行列用の構文は、定数行列用の構文と同じです。多項式行列のデターミナントの計算は、det 関数により行ないます。

.....

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

-->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      !
```

分数多項式行列の定義です。f の内部表現は、リスト list('r',num,den) です。ただし、num と den は共に多項式行列です。

.....

```
-->pause

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

-->pt
 pt  =

          2   3
    s + 2s + s
```

次に、コマンド pause を用いて新しい環境に移動します。新しいプロンプト -1-> が現れます。これは、新しい環境のレベル (レベル 1) を示しています。最初の環境で使用可能だった全ての変数は、新しい環境においても使用可能です。

新しい環境において作成された変数は、return を用いて元の環境に返すことが可能です。引数なしで return を使用すると新しい環境で作成された全ての変数は元の環境に戻る前に破棄されます。pause 機能は、デバッグの際に非常に便利です。

.....

```
-->f21=f(2,1);v=0:0.01:%pi;frequencies=exp(%i*v);

-->response=freq(f21(2),f21(3),frequencies);

-->plot2d(v',abs(response)',[-1],'011',' ',[0,0,3.5,0.7],[5,4,5,7]);

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

分数多項式の定義が上に定義した行列 f の要素を展開することにより行なわれています。続いて、frequencies により定義された複素数の周波数値ベクトルにおいて分数多項式の評価が行なわれます。多項式の評価は、基関数 freq により行なわれます。numer(f21) は分子の多項式であり、denom(f21) は分母の多項式です。評価結果の可視化は、コマンド plot2d により行なわれます。(図 1.1 を参照して下さい。)

.....

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

        1
    ---------
```

```
              2
    1 + 2s + s

-->horner(f,w)
 ans  =

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

関数 horner により、ユーザーは多項式の変数 (あるいは記号) を変更することが可能です。(例えば、上記のように双 2 次変換を行なうような場合。)

```
-->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  !
```

線形システムの定義を状態空間表現で行なっています。ここで、関数 syslin は、連続時間 ('c') システム Sl を状態空間行列 (A,B,C) で定義しています。関数 ss2tf は、Sl を伝達関数表現に変換します。

```
-->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

       ans(2) = A matrix =

! - 0.5  - 0.5 !
! - 0.5  - 0.5 !

       ans(3) = B matrix =

! - 0.7071068    0.7071068    0. !
!   0.7071068    0.7071068    0. !

       ans(4) = C matrix =

! - 1.4142136    0. !

       ans(5) = D matrix =

!               2 !
!   0    1     s  !

       ans(6) = X0 (initial state) =

!   0. !
!   0. !

       ans(7) = Time domain =

 c
```

分数行列 R の定義を行なっています。

Sl は、(特異) 伝達行列 R で定義される連続時間線形システムです。tf2ss は、Sl を多項式行列 D で定義される多項式状態空間表現へと変換します。

線形システムは、(7 個のエントリを有する) 特別なリストで表現されることに注意して下さい。

```
-->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. !
```

sl1 は、Sl と 2*Sl +eye の並列結合より得られた伝達行列表現の線形システムです。状態空間表現の Sl についても同じ構文が有効です。

```
-->deff('[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=[Id     ,0*Ko; Id  ,-Ko   ];';
       'C1=[C ,0*C];Cl=syslin(''c'',A1,B1,C1)' ])

-->comp(compen)
```

ゲイン Ko のオブザーバーとゲイン Kr のコントローラにより制御される線形システムの状態空間表現 (Cl) を計算する compen という名前の関数のオンライン定義を示しています。

他の行列を用いてブロック形式で行列が構築されていることに注意して下さい。関数 compen は、comp によりコンパイルされます。

```
-->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])');

-->f=Cl(2),spec(f)
 f  =

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

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

上で定義された関数 compen をコールしています。ただし、ゲインは極配置を行なう基本関数 ppol のコールにより計算しています。結果として得られる f 行列は、表示されます。そして、その極の位置を行列の固有値を計算する基本関数 spec を用いて確認しています。(ここで、関数 compen は、線形システム (Sl) を入力とし受け、線形 (Cl) を出力として返す関数の例として deff によりオンラインで定義されています。一般に Scilab 関数は、ファイルで定義され、getf により Scilab にロードされます。)

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

-->save('myfile')

-->//Request to the host system to perform a system command

-->unix_s('rm myfile')

-->//Request to the host system with output in this Scilab window
```

```
-->unix_w('date')
Sun Nov 17 10:48:13 JST 1996

-->
```

Unix 環境とエラーメッセージの関係を示しています。: 変数 q が未知であるためコマンドはシステムにより解釈することができません。

```
-->foo=['      subroutine foo(a,b,c)';
    '      c=a+b';
    '      end'  ];

-->unix_s('\rm foo.f')
x
 !--error 10000
unix_s: rm: foo.f: No such file or directory
at line      23 of function unix_s                    called by :
unix_s('\rm foo.f')


-->write('foo.f',foo);

-->unix_s('make foo.o')

-->link('foo.o','foo')

-->deff('[c]=myplus(a,b)',...
   'c=fort(''foo'',a,1,''r'',b,2,''r'',''out'',[1,1],3,''r'')')

-->myplus(5,7)
 ans  =

    12.
```

Fortran サブルーチンを定義する、文字列の列ベクトルを定義しています。このルーチンはコンパイルされ (コンパイラが必要です。)、動的に Scilab にリンクされます。そして、関数 myplus により対話的にコールされます。

```
-->deff('[ydot]=f(t,y)','ydot=[a-y(2)*y(2) -1;1 0]*y')
```

```
-->a=1;comp(f);y0=[1;0];t0=0;instants=0:0.02:20;

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

-->plot2d(y(1,:)',y(2,:)',[-1],'011',' ',[-3,-3,3,3],[10,2,10,2])

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

1 次微分ベクトル f(t,y) を計算する関数を定義します。続いて、この関数で使用される定数 a の定義が行なわれ、関数がコンパイルされます。

基本関数 ode が f(t,y) により定義される微分方程式を $t = 0$ において $y(0) = \langle 1; 0 \rangle$ の範囲で、時間 $t = 0, .02, .04, \ldots, 20$ において解が与えられた範囲について積分します。結果を図 1.2 にプロットします。ただし、積分ベクトルの最初の要素をこのベクトルの 2 番目の要素に対してプロットしています。

.....

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

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

-->m*m'
     !--error    43
not implemented in scilab....


-->deff('[x]=%cmc(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,'])

-->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)  !
```

文字列を保持する行列を定義しています。デフォルトでは、文字列を要素とする 2 つの行列のシンボリックな積の計算は、Scilab において定義されていません。%cmc に関する (オンライン) 関数の定義は、文字列行列の積算を定義します。(deff の内容が引用符の中に引用符を含んでいるため 2 重引用符が必要であるということに注意して下さい。) %cmc に関する関数定義の始めにある % は、Scilab にこれまで存在しない演算の定義を許可します。そして、cmc という名前で、“chain multiply chain” (鎖かける鎖) を意味します。この例は、あまり有益ではありません。: 単に複雑な構造において演算子を定義する方法を示しているだけです。

```
-->deff('[y]=calcul(x,method)','z=method(x),y=poly(z,''x'')')

-->deff('[z]=meth1(x)','z=x')

-->deff('[z]=meth2(x)','z=2*x')

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

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

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

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

ある関数を他の関数に対する引数として渡すことを示す簡単な例です。Scilab 関数は、定義されたり、ロードされたり、行列やリストのような他のオブジェクトとして操作されたりするオブジェクトです。

```
-->quit
```

Scilab を終了します。

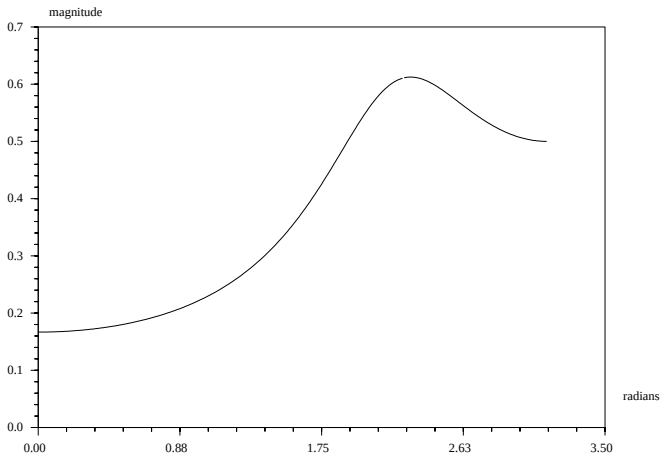


図 1.1: 簡単な応答

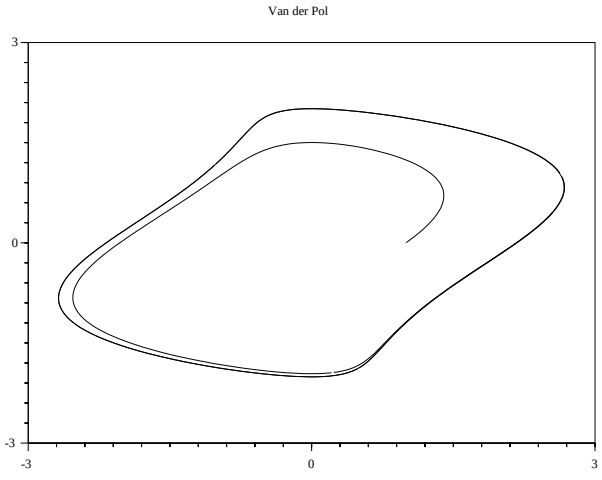


図 1.2: 位相プロット

第2章　データ型

Scilab は、複数の基本的なデータ型を認識します。スカラーオブジェクトは、定数、論理変数、多項式、文字列、分数式 (多項式の商) です。これらのオブジェクトは、これらのスカラーをエントリーとして許容する行列を定義することも可能です。その他の基本的なオブジェクトは、リストと関数です。定数と論理変数のみ、疎行列を定義することができます。本章の目的は、これらの型の各々の使用法を説明することです。

2.1　特別な定数

Scilab は、特別な定数 %i, %pi, %e, and %eps を基礎変数として提供します。定数 %i は $\sqrt{-1}$ を表します。%pi は $\pi = 3.1415927\cdots$、%e は、三角法の定数 $e = 2.7182818\cdots$、%eps は、マシンの精度を表す定数 (%eps は、$1 + $ %eps $= 1$ となる最大の数です。) です。%inf と %nan は、それぞれ "infinity" と "NotANumber" を表します。

最後に、論理定数は%t と %f で、それぞれ "真 (true)" と "偽 (false)" を表します。%t は、1==1 と同じであり、%f は、~%t と同じであることに注意してください。

これらの変数は、"あらかじめ定義された" 変数とみなされます。これらは保護されており、削除することはできません。また、save コマンドにより保存されません。各ユーザーは、predef コマンドを使用することにより、自分用の "あらかじめ定義された" 変数を持つことができます。最良の方法は、恐らく、これらの特別な変数を自分用のスタートアップファイル <home dir>/.scilab にセットすることでしょう。

2.2　定数行列

Scilab は、一連のデータオブジェクトを行列として扱います。実数または複素数のエントリ有するスカラー、ベクトル、行列は、全て行列として扱われます。

オブジェクトの使用法の詳細は、次の Scilab セッションで明らかになります。

スカラー　スカラーは、実数または複素数のどちらかです。スカラーの値は、ユーザーにより選択された名前の変数に割り付けられます。

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

    5. + 2.i

--> B=-2+%i;
```

```
--> b=4-3*%i
 b  =

    4. - 3.i

--> a*b
 ans  =

    26. - 7.i

-->a*B
 ans  =

  - 12. + i

--> c=a+b;

-->c
 c  =

    9. - i
```

Scilab は、キャリッジリターンで終る行をただちに評価するということに注意して下さい。セミコロンで終る命令は、評価されますが、画面に表示されません。現在の Scilab は、大文字小文字を区別します。(2.0 版は大文字小文字を区別しませんでした。)

ベクトル　ベクトルを作成する通常の方法は、次のようなものです。

```
--> 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. !
```

コンマ (あるいは空白) により区切られたベクトル要素の組は行ベクトルを作成し、セミコロンにより区切られたベクトル要素の組は列ベクトルを作成します。また、シングルクオートは、ベクトルの転置 を行うために使用されます。(複素数のエントリでは、複素共役が与えられます。) 同じ次元のベクトルは、加算および減算ができます。行ベクトルと列ベクトルのスカラー積の例が上で示されています。例示されているように要素毎の掛け算 (`.*`) と割算 (`./`) も可能です。

次の例で空白の位置の役割を示します。:

```
-->v=[1 +3]
 v  =

!   1.    3. !

-->w=[1 + 3]
 w  =

!   1.    3. !

-->w=[1+ 3]
 w  =

    4.
```

```
-->u=[1, + 8- 7]
 u  =

!   1.    1. !
```

要素が等間隔で増加、減少するベクトル は次のように作成することができます。

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

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

得られたベクトルは、最初の値から始まり、2 番目の値を増分として、3 番目の値で終っています。指定しない場合、デフォルトの増分は 1 となります。ones と zeros 機能を使用して定数ベクトルを作ることができます。

```
--> 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. !
```

```
-->zeros(v)
 ans  =

!   0.    0.    0. !

-->zeros(1:5)
 ans  =

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

ones または zeros はその引数のベクトルを同じ次元を有する 1 または 0 を要素とするベクトルで置き換えるということに注意して下さい。

行列　行の要素は、コンマあるいスペースにより区切られており、列の要素はセミコロンにより区切られています。行列とスカラー、ベクトル、他の行列との積は、通常の感覚で行うことができます。行列の加算と減算は要素毎行われます。そして、要素毎の掛け算、割算は演算子.*および ./ により実行することが可能です。

```
--> 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       =

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

--> a*b'
 ans       =

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

上の例に示したように、カンマで区切られた 2 つの実数を引数とした ones 演算子は、引数を次元とした行列を作成します。(zeros と同様。) 行列は、より大きな行列 matrices の要素として使用することが可能です。更に、行列の次元は変更可能です。

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

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

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

!   5.    6. !
!   7.    8. !

--> c=[9 10;11 12]
 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. !
```

行列 d は、他の行列の要素を用いて作成されていることに注意して下さい。matrix 基関数は、2 つの引数で指定された次元を用いて行列 d を要素とする新しい行列を e 作成します。行列 d における要素の順序は、e において再構成された行列の順番となり、上から下へ、左から右へとなります。

関数 eye は、対角要素に 1 を有する $m \times n$ 行列を作成します。(引数が行列 e である場合は、m と n は e の次元となります。)

定数要素の疎行列は、非ゼロ要素により定義することが可能です。(更に、詳細な情報は、help sparse と入力してみてください。) 一度定義されると、完全な行列と同様に操作可能です。

2.3　文字列の行列

文字列は、単引用符を用いて作成することができます。文字列の結合は、+ 演算により行うことができます。文字列の行列は、通常の行列と同様に、つまり、括弧を用いて作成されます。文字列の行列の非常に重要な機能は、関数を操作し作成する能力です。更に、数学的オブジェクトの数式処理が、文字列の行列を用いて実行可能です。次の例は、これらの機能を説明するものです。

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

--> 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)  !

--> evstr(at)
 ans       =

!   3.    9. !
```

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

上の Scilab セッションにおいて、関数 trianfml は行列 a の数式処理による 3 角化を実行します。得られた数式表現の行列の値は、evstr により得ることができます。

文字列行列の非常に重要な面は、自動的に新しい関数を作成するために使用することができるということです。(この関数に関する更に詳細な情報は、 3.2 節を参照して下さい。) 自動的に関数を作成する例を次の Scilab セッションに示します。この例は、2 つの変数 s と t の多項式を評価するために望ましいものです。2 つの独立変数を有する多項式は、Scilab により直接にサポートされないため、新しいデータ構造を構築する必要があります。(2.6 節を参照して下さい。)

評価する多項式は、$(t^2+2t^3)-(t+t^2)s+ts^2+s^3$ です。

```
-->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       =

[p]=pst(t)

-->pst
 pst       =

[p]=pst(t)

-->pst(1)
 ans       =

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

ここで、多項式は、リスト p により変数 s の係数を設定するコマンドを用いて作成されます。リスト p は、新しい関数 pst を作成する関数 makefunction により解釈されます。新しい関数の中身は表示可能であり、この関数は t の値を用いて評価できます。新しい関数 pst の作成は次のように行うことができます。

```
function [newfunction]=makefunction(p)
   n=size(p);
   num=mulf(makestr(p(1)),'1');
   for k=2:n,
      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,
      ck=c(k),
      if ck<>0 then,
      str=addf(str,mulf(string(c(k)),(xstar+string(k-1))));
      end;
   end,
```

ここで、関数 makefunction は、リスト p を取り、関数 pst を作成します。makefunction の内部で、新しい 2 変数多項式の各項を表す文字列を作成する別の関数 makestr をコールします。addf は、文字列を加えるための関数であり、mulf は、文字列を掛けるための関数です。(例えば、addf(x,y) は、文字列 x+y を作成します。) 最後に、新しい関数を作成するための基本的なコマンドは、基関数 deff です。基関数 deff は、2 つの文字列行列で定義された関数を作成します。ここで、関数 p は、2 つの文字列'[p]=newfunction(t)' と text により定義されます。ただし、文字列 text は、2 変数の多項式を評価します。

2.4　多項式と多項式行列

Scilab において、多項式は、簡単に作成し、操作することができます。多項式行列の操作は、本質的に定数行列の操作と同様です。Scilab の基関数 poly は、多項式の係数または多項式の根を指定するために使用されます。

```
--> 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

--> q/p
 ans       =

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

多項式 p は、**根** 1 と 2 を持ち、一方、多項式 q は、**係数** 1 と 2 を持つことに注意して下さい。基関数 poly の 3 番目の引数は、係数フラグオプションを指定するものです。poly の最初の係数が正方行列であり、かつ、根のオプションが有効である場合、結果は行列の多項式の係数になります。

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

              2
  - 2 - 5s + s
```

多項式は、通常の演算と同様に、加えたり、引いたり、掛けたり、割ったりすることが可能です。しかし、同じ変数で表される多項式の間でのみ可能です。

　実数や複素数の定数のように多項式は、行列の要素として使用することができます。これは、システム理論に関して Scilab の非常に便利な機能の一つです。

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

    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      !
```

上の例から、行列を多項式と分数から構築することが可能であることがわかります。

2.5　論理値行列

論理定数は、%t と %f です。これらは、論理行列の中で使用することができます。構文は、通常の行列と同じです。すなわち、結合したり、転置したり、といったこと等を行うことが可能です。

論理行列に関して使用されたり、論理行列を作成するために使用される演算子は、== と ~ です。

B が論理行列の場合、or(B) と and(B) 論理演算 or と and を行います。

```
-->%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 !
```

論理疎行列は、例えば、2 つの定数疎行列が比較された場合に作成されます。行列は、通常の論理行列と同様に取り扱われます。

2.6　リスト、線形システム

Scilab はリストデータ型を有しています。リストは、データオブジェクトの集合であり、同じデータ型である必要はありません。一つのリストは、他のリスト、関数、ライブラリと同様にこれまで説明した全てのデータ型を含むことができます。リストは、データオブジェクトの構造を定義するのに便利です。例えば、Scilab において線形システムは、リストとして取り扱われます。線形システムを定義するための基本関数は、`syslin` です。この関数は、パラメータとして状態空間形式で線形システムを定義する定数行列をとります。システムが伝達関数形式の場合、入力は分数行列である必要があります。より明確に定義すると、`syslin` の呼び出し手順は、`Sl=syslin('dom',A,B,C,D,x0)` または `Sl=syslin('dom',trmat)` のどちらかです。`dom` は、文字 `'c'` または `'d'` のどちらかです。それぞれ、連続時間系、離散時間系を表します。`D` は、多項式行列 (プロパーでない系) とすることが可能であることを知っておくと良いでしょう。ここで、`D` と `x0` はオプションの引数です。`trmat` は、分数行列、つまり、分数 (多項式の比) の行列として定義されます。ある形式から他の形式への変換は、`ss2tf` または `tf2ss` により行われます。プロパーでない系も処理することが可能です。

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

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

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


       h(1)   (state-space system:)

 lss

       h(2) = A matrix =

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

       h(3) = B matrix =

!   0. !
!   1. !

       h(4) = C matrix =

! - 1.    0. !

       h(5) = D matrix =

    0.

       h(6) = X0 (initial state) =

!   0. !
!   0. !

       h(7) = Time domain =

 c

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

-->hs=ss2tf(h)
 hs  =

        1
    ---------
             2
```

```
    1 + 3s + s

-->size(hs)
 ans  =

!   1.    1. !

-->hs(1)
 ans  =

 r

-->hs(2)
 ans  =

    1

-->hs(3)
 ans  =

              2
    1 + 3s + s

-->hs(4)
 ans  =

 c

-->typeof(hs)
 ans  =

 rational

-->//inversion of transfer matrix

-->inv(hs)
 ans  =

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

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

-->inv(h)
 ans  =


       ans(1)   (state-space system:)

 lss

       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

-->//conversion of this inverse

-->ss2tf(ans)
 ans  =

              2
    1 + 3s + s
```

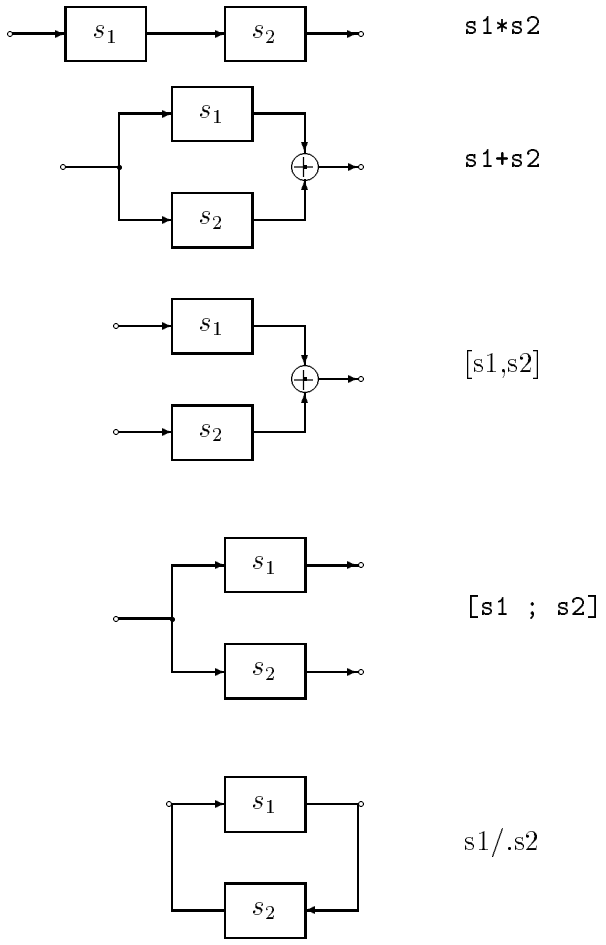


図 2.1: 線形システムの相互結合

上の Scilab セッションでわかるように、リスト h は、文字列 'lss' で始まっています。この場合、この文字列は、このリストが線形系を表すことを示しています。次の 5 つのリストの要素は、線形系の状態空間表現とその初期値を与える行列です。($\dot{x} = Ax + Bu$, $y = Cx + Du$, $x(0) = x_0$) 最後の要素 'c' は、リストが連続線形系を表すことを示しています。

リスト表現は、線形系を抽象的なデータオブジェクトとして操作することを可能にします。例えば、線形系は、他の線形系と結合することが可能です。また、線形系の伝達関数表現を ss2tf を使用して先に行ったように得ることが可能です。

線形系の伝達関数表現は、それ自体リストであることに注意して下さい。リストは、4 つの要素を有しています。最初の要素は、リストの要素が分数多項式行列であることを示す文字 'r' です。2 番目と 3 番目は、分子と分母の多項式です。最後の 4 番目の要素は、伝達関数が連続系であることを示す文字 'c' です。Scilab における系の操作をする上で便利な点は、一つのシステムを一つのデータオブジェクトとして扱うことができることです。線形系は、相互結合 することができ、それらの表現は、状態空間から伝達関数、またはこの逆に簡単に換えることができます。

線形系の相互結合は、図 2.1 のように説明することができます。2 つの系 s1 と s2 の間で可能な相互結合の各々について、相互結合を作るコマンドを図 2.1 において対応するブロック図の右側に示しています。フィードバック結合は、s1/.s2 で行われることに注意して下さい。

線形系の表現は、状態空間表現または伝達関数表現です。2 つの表現は、関数 tf2ss と ss2tf により相互に変換することが可能です。この関数は、それぞれ伝達関数表現から状態空間表現へ、状態空間表現から伝達関数表現へ変換を行います。

作成、表現形式の変換、線形系の相互結合の例を次の Scilab セッションで示します。

```
-->//system connecting

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

    s

-->ft=1/(s-1)
 ft  =

      1
    -----
  - 1 + s

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

      1
    -----
  - 2 + s

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

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

-->gls=tf2ss(gt);

-->ssprint(gls)

x = | 2 |x + | 1 |u

y = | 1 |x

-->hls=gls*ft;

-->ssprint(hls)

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

```
y = | 1   0 |x

-->ht=ss2tf(hls)
 ht  =

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

-->gt*ft
 ans  =

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

上のセッションは少し長いですが、線形系の取り扱いに関していくつかの重要な面を示しています。まず、基関数 `syslin` を使用して 2 つの線形系が伝達関数表現で作成されています。この基関数は、この例において系を連続系として (離散系の逆として) 定義するために使用されています。2 つの伝達関数の内の一つをリスト形式の等価な状態空間表現に変換するために基関数 `tf2ss` が使用されています。(関数 `ssprint` は、状態空間表現の線形系に関する更に可読性の高いフォーマットを作成することに注意して下さい。) 次の 2 つの系の操作は、これらの積結合を作成し ます。2 つの系の相互結合は、系の内の一つが状態空間表現でもう一つが伝達関数表現である場合でも行うことができることに注意して下さい。相互結合の結果は、状態空間表現で与えられます。最後に、基関数 `ss2tf` は、相互結合の結果の系を等価な伝達関数表現に変換するために使用されています。

2.7　関数 (マクロ)

関数 (マクロとも呼ばれる) は、Scilab の非常に便利な面です。関数は、新しい環境、つまり、元の環境の変数から関数の変数が隔離されている環境、で実行されるコマンド の集合です。関数は、複数の異なった手法で作成され、実行されます。更に、関数は引数を取り、条件分岐やループのようなプログラミング機能を有しており、再帰的に呼び出すことができます。関数は、他の関数の引数したり、リストの要素とすることが可能です。関数を作成するための最も便利な手法は、テキストエディタを使用したやり方です。しかし、基関数 `deff` を用いて関数を Scilab 環境で直接作成することも可能です。

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

```
--> foo(5)
 ans       =

    1.

--> foo(-3)
 ans       =

  - 1.
```

通常、関数は、エディタを用いてファイルで定義され、getf('filename') または getf('filename','c') により Scilab に読み込まれます。これは、File operation ボタンをクリックすることによっても可能です。この後の構文は、filename の関数をロードし、これらをコンパイルします。filename の最初の行は、次のようになります。:

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

ただし、yi は出力変数であり、xi は入力変数です。
関数の使用と作成の詳細に関しては、3.2 節を参照して下さい。

2.8　ライブラリ

ライブラリは関数の集合であり、Scilab が読み込まれた時に自動的に Scilab 環境にロードされるか、ユーザーにより要求された際にロードされます。ライブラリは、lib コマンドにより作成されます。ライブラリの例は、SCIDIR/macros ディレクトリに提供されています。このディレクトリの中には、ライブラリの各関数の名前を保持するアスキーファイル “names”、関数のソースコードである一連の .sci ファイル、関数のコンパイルされたコードである一連の .bin ファイルがあります。Makefile は、関数をコンパイルし、 .bin ファイルを作成するために scilab を呼びます。ライブラリのコンパイルされた関数は、その関数が最初に呼ばれた時に自動的に Scilab にロードされます。

2.9　オブジェクト

関数 typeof が様々な Scilab オブジェクトの型を返すことを述べて本章を完結することにします。次のようなオブジェクトが定義されています。

- usual 実数または複素数のエントリを有する行列。
- polynomial 多項式行列: 係数は、実数または複素数とすることが可能。
- boolean 論理行列。
- character 文字列行列。
- uncompiled function コンパイルされていない関数用。
- function コンパイルされた関数。

- rational 分数行列。(または、線形系の伝達関数表現 (syslin リスト))
- state-space 線形系の状態空間表現 (syslin リスト)。
- sparse 疎行列。
- list 通常のリスト、すなわち、線形系 (syslin リスト) 以外のリスト。
- library ライブラリ定義。

第3章　プログラミング

Scilab の最も便利な機能の一つは、関数を作成し使用できることです。これにより、ライブラリの使用のような簡易なモジュールを用いた手法で Scilab パッケージに組み込むことが可能な特定のプログラムの開発が可能になります。本章では、次のような題目を取り扱います。:

- プログラミングツール
- 関数の定義と使用
- 新しいデータ型における演算子の定義
- デバッグ法

ライブラリの作成は、後の章で議論します。

3.1　プログラミングツール

Scilab は、ループ、条件文、ケース選択、新しい環境の作成を含むプログラミングツールを完全にサポートします。殆んどのプログラミング作業は、関数の環境で行われるべきです。以下に、使用可能なプログラミングツールについて説明します。

3.1.1　比較演算子

Scilab のデータオブジェクトの値を比較する方法は、5 つあります。これらの比較法を以下の表に示します。

== or =	equal to
<	smaller than
>	greater than
<=	smaller or equal to
>=	greater or equal to
<> or ~=	not equal to

これらの比較演算子は、条件節の評価の際に使用されます。

3.1.2　ループ

Scilab には、2 種類のループがあります。`for` ループと、`while` ループです。`for` ループは、インデックスベクトルを使用した各ステップ毎に `end` までのコマンドを実行します。

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

    1.
 x         =

    2.
 x         =

    6.
 x         =

    24.
```

for ループは、ベクトルまたは行列の列の要素を値としてとる行列についてループを行うことができます。

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

    0.
 x         =

    3.
 x         =

    3.
```

for ループは、リストに関してもループを行うことが可能です。構文は、行列と同じです。

while ループは、ある条件が満たされるまで一連のコマンドを繰り返し実行します。

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

    2.
 x         =

    4.
 x         =

    8.
 x         =
```

```
    16.
```

for または while ループは、コマンド break で終えることが可能です。:

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

-->a
 a  =

    3.
```

3.1.3　条件文

Scilab には 2 つのタイプの条件文があります。: if-then-else 条件文と select-case 条件文です。if-then-else 条件文は式を評価し、真 (true) である場合、then 命令と else 命令 (または end 命令) の間の命令を実行し ます。偽 (false) の場合、else と end の間の命令が実行されます。else が常に必要なわけではありません。elseif は、通常の意味を有しており、これも、インタープリターにより認識されるキーワードです。

```
--> 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 条件文は、ある式を可能な複数の式と比較し、始めの式に等しい最初のケースに続く命令を実行し ます。

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

  - 1.

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

    i
```

条件が何にも一致しない場合のために else 文 を含めることが可能です。

3.2　関数の定義と使用

Scilab 環境において直接関数を定義することが可能です。しかし、関数を含むファイルをテキストエディタで作成する方法が最も簡単でしょう。本節では、関数の構造とほとんど関数環境でのみ使用される幾つかの Scilab コマンドについて説明します。

3.2.1　関数の構造

関数の構造は、次のようなフォーマットに従う必要があります。

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

ただし、foo は関数名であり、xi は関数の m 番目の入力引数、yj は関数からの n 番目の出力引数です。そして、3 つの縦に並んだ点は関数で実行される命令のリストを表しています。$k!$ を計算する関数の例を次に示します。

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

この関数が fact.sci という名前のファイルに含まれている場合、この関数は、次のようにして Scilab 環境に “ロードされ”、使用されます。

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

    0.

--> getf('../macros/fact.sci')

--> exists('fact')
 ans       =

    1.

--> x=fact(5)
 x         =

    120.

--> comp(fact)
```

上の Scilab セッションにおいて、コマンド exists は、(exist に 0 で答えることにより) fact が環境に存在しないことを示しています。getf を用いて環境に関数をロードした後は、exists は、関数が存在することを示します。(答えは、1) 例では、5! を計算しています。より高速に実行するために関数を comp によりコンパイルすることができます。fact のコンパイルは、コマンド getf('../macros/fact.sci','c') により直接実行可能であることに注意して下さい。

3.2.2　関数のロード

関数は通常ファイル中で定義されます。関数を含むファイルは、次のようなフォーマットに従う必要があります。

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

ただし、foo は関数名です。xi は、入力パラメータ、yj は出力パラメータです。そして、3 つの縦に並んだ点は関数で実行される命令のリストを表します。入出力パラメータは、あらゆる Scilab オブジェクトとすることができます。(関数自体も含みます。)

関数は、Scilab オブジェクトであり、ファイルとして考えるべきではありません。Scilab で使用するために、ファイルで定義された関数は、コマンド getf(filename,'c') により**ロードされなければなりません。**ファイル filename が関数 foo を含んでいる場合、関数 foo は、コマンド getf(filename,'c') (ただし、'c' はオプション) により事前にロードされている場合にのみ実行することができます。ファイルは複数の関数を含むことができます。関数は、コマンド deff により“オンライン”で定義することも可能です。これは、ある他の関数の出力パラメータにより関数を定義したい場合に便利です。

関数の集合をライブラリとして統合することができます。(lib コマンドを参照して下さい。) 標準 Scilab ライブラリ (線形計算、制御、...) はサブディレクトリ SCIDIR/macros/で定義されています。

3.2.3　グローバル変数と局所変数

関数中のある変数が定義されていない場合 (そして入力パラメータで定義されない場合)、呼び出し側の環境において同じ名前の変数の値を取得します。しかし、この変数は、resume を使用しない限り、関数の内部で行った修正が呼び出した環境におけるその変数の値を変更しないという意味で局所変数のままです。(以下を参照して下さい。) 関数は、入出力パラメータ無しで呼ぶことが可能です。以下に例を示します。

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


-->[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.
```

呼び出しの際のパラメータの一つが未定義の場合、関数を呼び出すことはできないということに注意して下さい。

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



-->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
```

3.2.4　特別な関数コマンド

Scilab は、ほとんどの場合、関数の内部で使用される幾つかの特別なコマンドを有しています。これらのコマンドを以下に示します。

- `argn`: 関数の入力および出力の引数の数を返します。
- `error`: 関数の実行を中断し、エラーメッセージを出力し、エラーが検知された際に直前の環境に戻るために使用されます。
- `warning`,
- `pause`: 一時的に関数の実行を中断します。

- break: ループを強制的に終了します。
- return または resume : 呼び出し側の環境に戻すため、そして、呼び出し側の環境に関数の環境における局所変数を引き渡すために使用されます。

次の例では、これらのコマンドを説明する foo という名前の関数を Scilab にロードします。

```
-->getf('../macros/foo.sci')

-->foo
 foo       =

[z]=foo(x,y)

--> 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
```

この例では、foo.sci をロードし、関数の内容を表示しています。最初の foo のコールでは、関数の計算に使用できない引数を渡しています。関数は、動作を終了し、ユーザーにエラーの場所を表示します。2 番目の関数コールでは、slope の計算の後、動作を中断します。この時、ユーザーは、関数の内部で計算された値を評価することができます。また、プロットを実行したり、実際、Scilab で可能な全ての演算を行うことができます。-1-> プロンプトは、pause コマンドにより作成された現在の環境が関数の環境であり、呼び出し側の環境でないことを示しています。制御は、コマンド return により関数に返されます。関数の実行は、コマンド quit または abort により中断することが可能です。最後に関数は、z の値を返して計算を終了します。関数の局所変数である変数 s は、グローバル環境に引き渡されており、呼び出し側の環境においても使用可能です。

3.3　新しいデータ型における演算子の定義

Scilab における新しいデータ型について組込みの演算子と等価な演算子を定義することが可能です。これにより、ユーザーは、Scilab に存在するあらゆる 2 つのデータ型に関して乗算、割算、加算等を定義することが可能です。例として、(リストで表現された)2 つの線形系は、並列結合を表す加算を行ったり、直列結合を表す乗算を行ったりすることが可能です。Scilab は、以下に記述された特別な名前の付け方に従う (ユーザーにより書かれた) 関数を探索することによりユーザーが定義した演算を実行します。

ユーザーにより定義された演算子を識別するために使用するこの Scilab の表記法は、次のような取り決めの元で定義されます。ユーザー定義関数の名前は、4 つ (あるいは 3 つ) のフィールドより構成されます。最初のフィールドは、常に記号 `%` です。3 番目のフィールドは、次の表の中の文字の一つであり、2 つのデータ型の間で実行される演算子の型を表します。

3 番目のフィールド	
SYMBOL	OPERATION
a	+
b	; (row separator)
c	[ ] (matrix definition)
d	./
e	() extraction: m=a(k)
i	() insertion: a(k)=m
k	.*.
l	\ left division
m	*
p	^ exponent
q	.\
r	/ right division
s	-
t	' (transpose)
u	*.
v	/.
w	\.
x	.*
y	./.
z	.\.

2 番目と 4 番目のフィールドは、最初と 2 番目のデータオブジェクトをそれぞれ表します。これらは、関数により処理され、次の表の中の記号により表されます。

2、4 番目のフィールド	
SYMBOL	VARIABLE TYPE
s	scalar
p	polynomial
l	list (untyped)
c	character string
m	function
xxx	list (typed)

型のある (typed) リストとは、リストの最初の要素が文字列であり、その文字列の最初の 3 文字が上の表における xxx で表されるリストです。

例えば、線形系を表すリストは、list('lss',a,b,c,d,x0,'c') であり、この場合、上の xxx は lss です。

2 つの線形系を乗算する (直列結合を表す) 関数名の例は、%lssmlss です。最初のフィールドは、% であり、2 番目のフィールドは lss (線形系)、3 番目のフィールドは m "乗算 (multiply)"、4 番目のフィールドは lss です。この乗算を実行するユーザー関数は、以下のようになります。

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

(例えば、getf またはライブラリに収録することにより) Scilab にこの関数をロードした後として、この関数の使用例は、次の Scilab セッションで説明されます。

```
-->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=%lssmlss(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
```

%lssmss の使用は、完全に透過的に 2 つのリスト s1 と s2 の乗算が通常の乗算演算子 * を用いて実行されたことと同じなります。

ディレクトリ SCIDIR/macros/percent は、(非常に多くの...) 線形系と伝達行列に関する演算を行う全ての関数を含んでいます。変換は、自動的に実行されます。例えば、関数 %lssmlss のコードは、そこにあります。(ここで与えられたコードよりかなり複雑であるこ t に注意して下さい。)

3.4　デバッグ法

Scilab 関数をデバッグする最も簡単な手法は、その関数に pause コマンドを導入することです。実行時に、この関数はこの点で中断し、異なった “レベル” を示すプロンプト -1-> を表示します。そして、別の pause は、-2-> を表示します... レベル 1 において、Scilab のコマンドは、他のセッションと同一です。しかし、ユーザーは、関数の内部あるいは外部、すなわち、関数の局所側であるか、呼び出し側に属するかを問わず、Scilab において存在する全ての変数を表示することが可能です。関数の実行は、コマンド return または resume により再開されます。(より上位のレベルで使用された変数は消去されます。) 関数の実行は、abort により終了されます。

関数にブレークポイントを挿入することも可能です。コマンド setbpt、delbpt、disbpt を参照して下さい。関数の実行中にエラーをトラップすることも可能であることに注意して下さい。コマンド errclear と errcatch を参照して下さい。Scilab の熟練者は、デバッグレベルを示す関数 debug(i) (ただし、i=0,..,4) を使用することができます。

第4章　基本的なプリミティブ

本章では、いくつかの Scilab の基本的な関数について簡単に説明します。更に詳細な情報はマニュアル (ディレクトリ SCIDIR/man/LaTex-doc を参照してください。) で提供されます。

4.1　環境と入出力

この章で Scilab 環境に関する次のような重要な特徴を説明します。: Scilab の実行を開始した際にある種の操作を自動的に実行する方法、そして、Scilab 環境から、あるいは Scilab 環境へデータを読み書きする方法。

4.1.1　環境

Scilab は、多くの変数と基関数と共にロードされます。コマンド who は、使用可能な変数の一覧を表示します。who コマンドは、使用可能な要素と変数の数も示します。ユーザーは、help <function-name> と入力することにより、一覧にある関数全てについてオンラインヘルプを得ることができます。

変数は、save を使用して外部バイナリファイルに保存することができます。同様に、以前保存した変数、は load を使用して Scilab にロードすることができます。

コマンド clear x y を実行した後では、変数 x と y はもはやその環境に存在しないことに注意して下さい。変数の引数を何も指定しない場合、コマンド save は Scilab 環境全体を保存します。同様に、コマンド clear を引数無しで使用した場合、その環境における全ての変数、関数、ライブラリをクリアします。

ファイルに存在する関数は、disp を用いて見ることが出来ます。そして、getf を使用してロードすることが出来ます。

関数のライブラリは、lib を用いてロードされます。

使用可能なライブラリ中の関数の一覧は、disp を使用して得ることができます。

4.1.2　ユーザー毎のスタートアップコマンド

Scilab がコールされた時にユーザーはホームディレクトリにある .scilab を使用して自動的に環境に関数、ライブラリ、変数を読み込み、コマンドを実行することができます。これは、ユーザーが Scilab プログラムを (バッチモードのように) バックグラウンドで実行したい場合に特に便利です。その他の場合で .scilab ファイルが有効なのは、いくつかの関数あるいはライブラリを頻繁に使用する場合です。この場合、Scilab が実行された時に欲しい関数とライブラリを自動的にロードするために .scilab ファイルの中でコマンド getf を使用することが出来ます。

4.1.3　入出力

コマンド save と load は便利ですが、コマンド read と write を使用してファイルと Scilab 間のデータ転送に関して更に多くの制御が行われます。2 つのコマンドは、Fortran の read、write コマンドと同様な動作をします。このコマンドの動作は次のようになります。

```
--> 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.        !
```

read では、行列 x の行と列の数を指定していることに注意して下さい。複雑なフォーマットを指定することも可能です。

4.2　Help

help ボタンをクリックするか、help item と入力することにより、オンラインヘルプが利用可能です。(ただし、通常、item は関数または基関数の名前です。) ヘルプ機能は、Unix xman コマンドに基づいています。apropos item は、whatis ファイルにおいて item を探します。この機能は、Unix の whatis コマンドと同等です。マニュアルで新しい項目を加えるのは簡単です。: SCIDIR/man サブディレクトリを少し見てみて下さい。Scilab LaTeX マニュアルは、オンラインマニュアル項目から Makefile により自動的に得ることができます。ディレクトリ SCIDIR/man/Latex-doc を参照して下さい。コマンド manedit は、ヘルプファイルをエディタで開くといういうことに注意して下さい。(デフォルトのエディタは emacs です。)

4.3　非線形計算

Scilab は、シミュレーションや最適化用の多くの強力な非線形基関数を有しています。

4.3.1　外部関数 (external)

“external” は、(ode、optim... のような、) いくつかの高レベル基関数の引数として使用される関数または Fortran ルーチンです。外部関数 (またはルーチン) の呼び出し手続きは、外部関数の引数をセットする高レベル基礎関数により構成されます。例えば、外部関数 costfunc は、optim 基関数の引数です。呼び出し手続きは、次のようになります。: optim 基関数により必要とされる [f,g,ind]=costfunc(x,ind)。次の Scilab の非線形基関数は外部関数を必要とします。: ode、optim、impl、dassl、intg、fsolve。計算時間が重要な問題では、外部関数を Fortran サブルーチンとしてコード化することが推奨されます。そのようなサブルーチンの例は、ディレクトリ SCIDIR/routines/default に与えられています。このようなサブルーチンが書かれた場合には、Scilab とリンクする必要があります。このリンク操作は、“動的に” 行うことができます。(link を参照して下さい。) 特定のインターフェースに挿入することにより、より永続的な手法でそのコードを導入することも可能です。(例えば、SCIDIR/routines/default にある fydot.f を参照して下さい。そして、make で新しい Scilab を再構築して下さい。)

4.3.2　非線形基関数

Scilab における主なシミュレーション用の基関数は、陽的または陰的な解法により広範にわたる非線形系の関数を積分することが可能です。停止時間を有する微分方程式の系を積分することも可能です。: 積分は、軌道が指定された曲面に達するまで行われます。以下に、例を用いて基本的な ode の構文を説明します。

この例は、ode の基本的な使用法を説明します。ここでは、2 重振り子の運動をシミュレーションします。θ_1 と θ_2 を最初と 2 番目の振り子が垂直を基準としてなす角度と定義します。そして、r_1、r_2、m_1、m_2 をそれぞれの振り子の長さと重りの質量とします。

振り子の運動を記述する非線形微分方程式は、:

$$\begin{bmatrix} mr_1 & m_2 r_2 \cos\theta \\ r_1 \cos\theta_1 & r_2 \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} = \begin{bmatrix} -m_2 r_2 \dot{\theta}_2^2 \sin\theta \\ r_1 \dot{\theta}_1^2 \sin\theta \end{bmatrix} - \begin{bmatrix} m \sin\theta_1 \\ \sin\theta_2 \end{bmatrix} g \tag{4.1}$$

ただし、g は重力加速度、$m = m_1 + m_2$、$\theta = \theta_1 + \theta_2$ とします。次の Scilab セッションは、振り子の運動を 1.5 秒間シミュレーションします。ただし、$g = 9.8m/s$、$r_1 = r_2 = 1m$、$m1 = m2 = 1kg$ とします。初期条件は、$\theta_1 = \pi/2$、$\theta_2 = \pi/2$、$\dot{\theta}_1 = 0$、$\dot{\theta}_2 = 0$ です。

```
--> m1=1;m2=1;r1=1;r2=1;

--> g=9.8;

--> t0=0;

--> t=0:.1:1.5;

--> z0=[%pi/2;%pi/2;0;0];

--> getf('../macros/dpend.sci','c');
```

```
--> z=ode(z0,t0,t,dpend);

--> pp(z)
```

上のセッションの結果を、図 4.1 にプロットします。ode は以下の 4 個の引数をとることに注意して下さい。: 初期条件ベクトル z0、初期時間 t0、積分値 z が欲しいところでのベクトル時間 t 、Scilab 環境に存在し、指定された時間 t とその時間における z の値より微係数 zd の値を計算する関数 dpend。

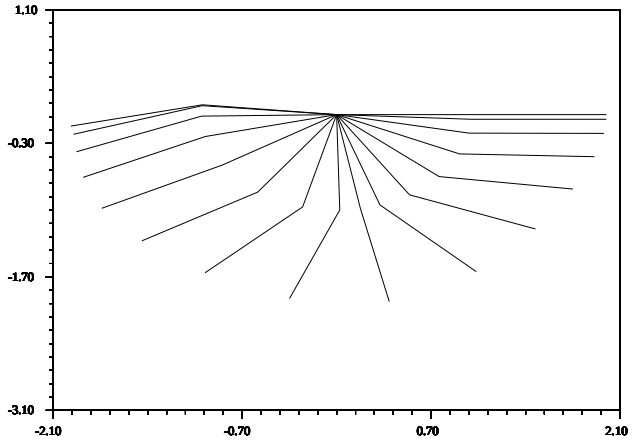


図 4.1: 2 重振子のシミュレーション

dpend 関数は、以下のように 2 重振り子の状態変数の微係数を計算します。(4.1)

```
function [zdot]=dpend(time,z)
   th1=z(1);th2=z(2);th1d=z(3);th2d=z(4);
   s12=sin(th1-th2);c12=cos(th1-th2);
   m12=m1+m2;s1=sin(th1);s2=sin(th2);
   mat=[m12*r1 m2*r2*c12;...
        r1*c12 r2];
   vec=-[m12*g*s1+m2*r2*th2d*th2d*s12;...
         g*s2-r1*th1d*th1d*s12];
   res=mat\vec;
   th1dd=res(1);th2dd=res(2);
   zdot=[th1d;th2d;th1dd;th2dd];

function []=pp(z)
   th1=z(1,:);th2=z(2,:);
   rc1=r1*cos(th1);rc2=r2*cos(th2);
   rs1=r1*sin(th1);rs2=r2*sin(th2);
   rs12=rs1+rs2;rc12=rc1+rc2;
```

```
rect=[-2.1,-3.1,2.1,1.1];
for k=1:maxi(size(th1));
   plot2d([0 rs1(k) rs12(k)]',-[0 rc1(k) rc12(k)]',...
             [-1],"011",' ',rect,[10,3,10,3]);
end,
```

微係数を計算する Scilab “external” 関数 (この例では、dpend) は、あるフォーマットを有している必要があります。このフォーマットは、次のようなものです。

```
function [xdot]=f(time,x)
   .
   .
   .
```

フォーマットは、2 つの変数を含む引数のリストを指定します。最初の変数 time は、時間を表すスカラーです。2 番目の変数 x は、状態変数の値を有するベクトル (または行列) です。返り値 xdot は計算された微係数であり、x と同じ次元を有しています。2 次以上の微分方程式については、状態変数の付加が行われます。Scilab における最適化用の基関数 optim は、広範な非線形問題において局所最適解を求める能力を有しています。以下に、optim の基本的な構文を例を用いて説明します。optim の使用法を説明するために使用される例は、ある物質から他の物質に伝播する平面波に関する古典的な面上のレイトレーシング問題です。ここでは問題を少し改変して構成します。ビーチにライフガードがいて、助けを求める人が水の中にいるとしましょう。ライフガードの陸上でのスピードは、水中とは異なっています。結果的に、彼は呼んでいる人のところまで行く時間を最小にするビーチと水中の境界線上の点を選択したいと思うでしょう。

ビーチと水の境界線が直線である場合、この問題は、解析的に解くことが可能です。境界線の関数が直線よりも複雑である時には、この問題を解析的に解くことは不可能になります。

この問題は次のように構成されます。ライフガードを直交座標系の原点に、救おうとする人を座標 (x, y) に置きます。そして、ビーチと水の境界を関数 $d(x)$ により記述します。この場合、ライフガードの移動時間は、以下のようになります。

$$T(p) = \frac{\sqrt{p^2 + d^2(p)}}{v_b} + \frac{\sqrt{(x-p)^2 + (y-d(p))^2}}{v_w} \tag{4.2}$$

ただし、p はライフガードの経路が境界と交わる点の x 座標であり、v_s と v_w はそれぞれビーチ上と水中におけるライフガードの速度です。ここで、p により与えられる x 座標はビーチと水面との境界線上のある点をユニークに定義すると仮定します。

局所最適解は、$dT(p)/dt = 0$ を満たす必要があります。T の微分は、d の微分の関数として計算することができます。

$$\frac{d}{dt}T(p) = \frac{p + dd_p}{v_s\sqrt{p^2 + d^2}} + \frac{(p-x) + (d-y)d_p}{v_w\sqrt{(x-p)^2 + (y-d)^2}} \tag{4.3}$$

ただし、d_p は、d の p に関する偏微分です。次の Scilab 関数は、(4.2) と (4.3) について計算を行います。

```
function [t,t_p,ind]=swim(p,ind)
```

```
   [d,d_p]=dpfun(p);
   as=sqrt(d^2+p^2);
   aw=sqrt((x-p)^2+(y-d)^2);
   t=as/vs+aw/vw;
   t_p=(p+d*d_p)/(vs*as)+((p-x)+(d-y)*d_p)/(vw*aw);


function [d,d_p]=dpfun(p);
   d=1.75-exp(log(1.55)*p);
   d_p=-log(1.55)*exp(log(1.55)*p);
```

ただし、関数 d と d_p は、関数 dpfun により計算されます。これら 2 つの関数は、次の SCILAB セッションにおいて最適点 p を見付けるために基関数 optim により使用されます。

```
--> x=1;y=1;vs=5;vw=1;

--> getf('../macros/swim.sci');

--> getf('../macros/plotopt.sci');

--> [ts,ps,tps]=optim(swim,0)
 end of optimization

 tps       =

    4.580D-16
 ps        =

    0.6144065
 ts        =

    0.8303513

--> popt(ps);
```

上の例から、基関数 optim は 2 つの引数をとり、3 つの値を返すことがわかります。最初の引数は、時間 (コスト) と時間の p に関する微係数を計算する Scilab 関数の名前です。2 番目の引数は、p の値に関する初期推定値です。返り値は、それぞれ、最適コスト ts 、その時の p の値が ps に、そして、最後に、この点における傾斜の値が tps となります。

ライフガードの最適経路を図 4.2 に示します。曲線は、海岸線を表し、点線は、ライフガードの最適経路を表します。

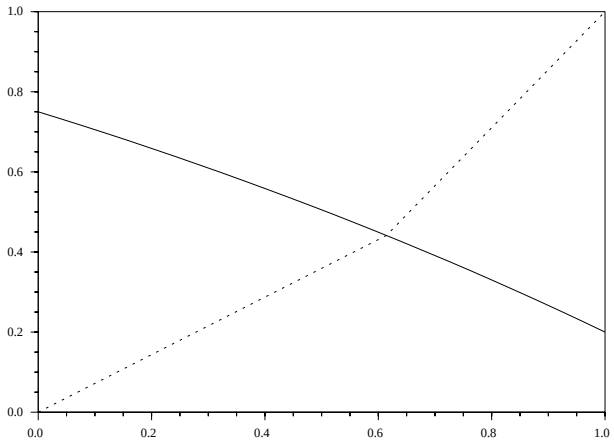


図 4.2: 最適なライフガードの経路

4.4　Fortran または C インターフェース

Scilab は、容易に Fortran または C のサブルーチンとのインターフェースを構築できます。最も簡単なやり方は、動的リンク基関数 `link` とサブルーチンコール基関数 `fort` を使用するものです。詳細は、Scilab マニュアルに記述されています。

外部 Fortran プログラム `interf` をコールすることにより、Fortran 副プログラムを Scilab にリンクすることも可能です。例題がある場所 `SCIDIR/default` にあるプログラム `interf.f` を参照して下さい。これらの簡便性は、非線形系のシミュレーションと最適化を行う場合に、また、数式処理システム Maple により自動的に生成されたプログラムを導入する場合に、特に便利です。

Fortran と Scilab で書かれたサブルーチンのインターフェースを動的に構築するには、2 つの手順を踏みます。これら、2 つの手順を以下に示します。

fortran サブルーチンを Scilab にリンクするためには、サブルーチンの実行オブジェクトファイルを含むファイルを有することがまず必要です。ここで、n 次のフィナボッチ数 f_n (すなわち、$0, 1, 1, 2, 3, 5, 8, \ldots$ の連続。ただし、$f_n = f_{n-1} + f_{n-2}$) を計算する Fortran サブルーチン `fibb` を導入します。

```
      subroutine fibb(fn,n)
      n0=0
      n1=1
      if(n.ge.3)then
      do 10 k=1,n-2
         fn=n0+n1
         n0=n1
         n1=fn
 10   continue
      else
         fn=n-1
```

```
      endif
      end
```

fibb 用のオブジェクトファイルがファイル fibb.o の中で利用可能であると仮定すると、次の Scilab セッションは、Scilab からのサブルーチンの利用法を示します。

```
-->unix("make fibb.o");

-->link('fibb.o','fibb')

-->link()
 ans       =

 fibb

-->n=6
 n         =

    6.

-->fn=fort('fibb',n,2,'i','out',[1,1],1,'r')
 fn        =

    5.
```

基関数 link は、通常 2 つの引数をとります。最初の引数はオブジェクトファイルの名前であり、2 番目の引数はサブルーチンの名前です。基関数 c_link は、これまでリンクされたサブルーチンを全て知ることができることに注意して下さい。サブルーチン fibb をコールするための fort の使用法は、少々複雑です。引数は、4 つのグループに分けられます。最初の引数 'fibb' はコールされるサブルーチンの名前です。引数 'out' は、残りの引数を 2 つのグループに分割します。'fibb' と 'out' の間の引数のグループは、入力引数、その fibb のコール時の位置、データ型のリストです。'out' の右側の引数のグループは、出力変数の次元、fibb コール時の位置、データ型です。

利用可能なデータ型は単精度実数 (real)、整数 (integer)、倍精度実数 (double) であり、それぞれ、文字列 'r'、'i'、'd' で表されます。fibb の 2 つの引数には、n については位置 2 であり、fn 1 の位置にあることが上記の引数で示されています。ベクトル [1,1] は、出力 fn が 1×1 の行列であることを示しています。

Scilab にリンクされたルーチンは、内部 Scilab 変数もアクセスすることが可能です。(ディレクトリ SCIDIR/routines/system2 の中のルーチン matz.f を参照して下さい。)

4.5　XWindow ダイアログ

関数の内部で対話的にパラメータを入力するためやデモ用に特定の XWindow ウインドウを開くことができると便利です。このような機能は、例えば、関数 x_dialog、x_choose、x_mdialog、x_matrix 、x_message により可能となります。demo ボタンをクリックすることにより実行可能なデモは、これらの関数の使用法に関する簡単な例を提供します。

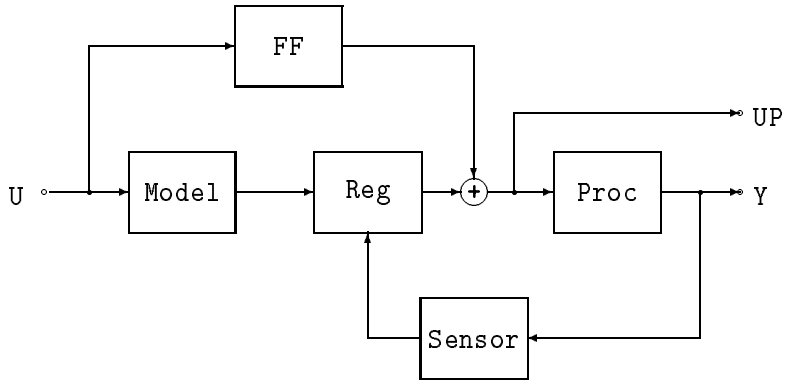


図 4.3: 相互結合された系

4.6　Maple インターフェース

Maple プロシージャ maple2scilab は、全ての Maple 式を Scilab において数値的に評価することを可能にします。入力時にこのプロシージャは、数式で表現された Maple 行列をとります。maple2scilab が Maple に呼ばれた際に、Fortran プログラムは自動的に Maple により作成されます。(Maple の Macrofort パッケージによります。) 同時にこのサブルーチンを (動的に) コールすることを可能にする Scilab 関数も作成されます。関数のパラメータは、Maple シンボルです。Maple 行列は、Scilab において数値的に評価されます。Fortran の使用は、Maple の数式は極端に複雑なので、速度の向上を目的 としています。

4.7　システムの相互結合

この節の目的は、Scilab の更に洗練された面を例を示すことにより説明することです。例では、Scilab が複数の系の相互結合を数式で表現する方法を示します。これは、逆に相互接続された系の性能を数値的に評価するために使用されます。相互接続された系の数式表現は、bloc2exp という関数を用いて行われます。得られたシステムの評価は、evstr により行われます。

以下の例は、図 4.3 に示されたように 数式表現によるシステムの相互結合を説明するものです。図 4.3 は、古典的なレギュレータ問題 を説明しています。ここで、Proc というラベルのブロックは、Sensor ブロックと Reg ブロックによるフィードバックを用いて制御されます。

Reg ブロックは、Proc ブロックをレギュレートするための方法を決めるために、Model ブロックからの出力を Sensor ブロックからの出力と比較します。入力信号 U から Proc ブロックへのフィルターとなるフィードフォワードブロックもあります。系の出力は、Y と UP です。図 4.3 に示された系は、Scilab において関数 bloc2exp を用いて表現されます。bloc2exp の使用法は、次の Scilab セッションで説明されています。2 種類のオブジェクトがあります。これらは、“transfer” と “links” です。ここで考慮される例では、5 つの伝達関数と 7 つの相互結合を行います。最初の伝達関数は、数式により定義されます。続いて相互結合が定義され、“相互結合された系” が特定なリストとして定

義されます。関数 bloc2exp はグローバルな伝達関数を数式により評価します。
　また、evstr は、その系が “値”、すなわち、Scilab 線形系として定義された値を与えられると、グローバルな伝達関数を数値的に評価します。

```
-->model=2;reg=3;proc=4;sensor=5;ff=6;somm=7;

-->tm=list('transfer','model');

-->tr=list('transfer',['reg(:,1)','reg(:,2)']);

-->tp=list('transfer','proc');

-->ts=list('transfer','sensor');

-->tf=list('transfer','ff');

-->tsum=list('transfer',['1','1']);

-->lum=list('link','input',[-1],[model,1],[ff,1]);

-->lmr=list('link','model output',[model,1],[reg,1]);

-->lrs=list('link','regulator output',[reg,1],[somm,1]);

-->lfs=list('link','feed-forward output',[ff,1],[somm,1]);

-->lsp=list('link','proc input',[somm,1],[proc,1],[-2]);

-->lpy=list('link','proc output',[proc,1],[sensor,1],[-1]);

-->lsup=list('link','sensor output',[sensor,1],[reg,2]);

-->syst=...
 list('blocd',tm,tr,tp,ts,tf,tsum,lum,lmr,lrs,lfs,lsp,lpy,lsup);

-->[sysf,names]=bloc2exp(syst)
 names     =


       names>1

 input
```

```
       names>2

!proc output  !
!             !
!proc input   !
 sysf      =

!proc*((eye-reg(:,2)*sensor*proc)\(-(-ff-reg(:,1)*model)))  !
!                                                           !
!(eye-reg(:,2)*sensor*proc)\(-(-ff-reg(:,1)*model))         !
```

bloc2exp の引数はリストのリストであることに注意して下さい。リストの最初の要素 syst は (実際は) 系の相互結合のブロック図を表すリストを示す文字列 'blocd' です。リスト syst の各リストエントリは、図 4.3 におけるブロックまたは相互結合を表します。ブロックを表すリストの形式は、文字列 'transfer' ではじまり、ブロック名を与える文字列の行列が続きます。ブロックが多入力多出力である場合、文字列の行列は、ブロック Reg により図示されたように表される必要があります。

ブロック間の相互結合もリストで表されます。リストの最初の要素は、文字列 'link' です。相互結合の 2 番目の要素は、その記号名です。相互結合の 3 番目の要素は、その結合への入力です。残りの要素は、全てその結合の出力です。相互結合への各入出力は、最初の要素にブロック番号を有するベクトルです。(例えば、model ブロックは、番号 2 に割り付けられています。) ベクトルの 2 番目の要素は、そのブロックのポート番号です。(多入力多出力ブロックの場合。) ある相互結合が何にも結合していない場合、ブロック番号の値は、負となります。(例えば、'input' というラベルのついた相互結合)

bloc2exp 関数の結果は、割り付けられていない系の入出力と sysf により与えられる系の伝達関数の数式表現を与える名前のリストです。sysf における記号名は、多項式と関連づけたり、関数 evstr を用いて評価することができます。このことを次の Scilab セッションで説明します。

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

-->ff=1;sensor=1;model=1;

-->proc=s/(s^2+3*s+2);

-->reg=[1/s 1/s];

-->sys=evstr(sysf)
 sys       =

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

```
!              2   3 !
!   2 + 5s + 4s + s  !
!   ---------------- !
!           2   3    !
!     s + 3s + s     !
```

得られた多項式伝達関数は、ブロック系の入力から 2 つの出力にリンクしている。evstr の出力は分数多項式行列 sys ですが、bloc2exp の出力は文字列の行列であることに注意して下さい。

ここで行われた数式の評価は、大規模な相互結合系ではあまり効率的ではありません。関数 bloc2ss は、以前の計算を状態空間形式で計算します。ここで、各系は、syslin リストまたは単なるゲイン (定数行列) として状態空間で与えられます。bloc2ss は、ユーザーが必要な変換を行わなかった場合、これを行います。各系には bloc2ss がコールされる前に値を与えられる必要があります。線形系が伝達関数形式で与えられる場合でも、全ての計算は、状態空間表現で行われます。

4.8　Scilab 関数の Fortran ルーチンへの変換

Scilab は、いくつかの Scilab 関数を Fortran サブルーチンに変換するコンパイラ (開発中) を提供します。得られたルーチンは、Fortran ライブラリ中のルーチンを利用します。全ての基本行列演算が利用可能です。

次のような Scilab 関数を考えてみましょう。:

```
function [x]=macr(a,b,n)
z=n+m+n,
c(1,1)=z,
c(2,1)=z+1,
c(1,2)=2,
c(2,2)=0,
if n=1 then,
 x=a+b+a,
else,
x=a+b-a'+b,
end,
y=a(3,z+1)-x(z,5),
x=2*x*x*2.21,
sel=1:5,
t=a*b,
for k=1:n,
 z1=z*a(k+1,k)+3,
end,
t(sel,5)=a(2:4,7),
x=[a b;-b' a']
```

このルーチンは、関数 mac2for を用いて Fortran に変換することが可能です。サブルーチンの各入力パラメータは、その型と次元を有するリストにより記述されます。ここで、3 つの入力変数、ただし、整数、倍精度実数、倍精度実数があり、その次元は (m,m), (m,m), (1,1) であるとしましょう。この情報は、次のようなリストで集められます。:

```
l=list();
l(1)=list('1','m','m');
l(2)=list('1','m','m');
l(3)=list('0','1','1');
```

The call to mac2for is made as follows:

```
comp(macr);
mac2for(macr2lst(macr),l)
```

このコマンドの出力は、スタンドアローンの Fortran サブルーチンを含む文字列です。

```
       subroutine macr(a,b,n,x,m,work,iwork)
c!
c  automatic translation
.
.
.
       double precision a(m,m),b(m,m),x(m+m,m+m),y,z1,24(m,m),work(*)
       integer n,m,z,c(2,2),sel(5),k,iwork(*)
.
.
.
       call dmcopy(b,m,x(1,m+1),m+m,m,m)
       call dmcopy(work(iw1),m,x(m+1,1),m+m,m,m)
       call dmcopy(work(iw1),m,x(m+1,m+1),m+m,m,m)
       return
c
       end
```

このルーチンは、Scilab にリンクし対話的にコールすることが可能です。

第5章　グラフィックス

この節では、Scilab のグラフィックスを紹介します。

5.1　グラフィクスウインドウ

Delete ボタンの右側にあるチェッカーは、グラフィックウインドウの 番号 wn です。この番号は、ウインドウのタイトルの最後に付けられています。例えば、ScilabGraphic1 です。ボタン Raise (Create) Window は、存在する場合は、ウインドウ wn を表に出します。存在しない場合は、これを作成します。ボタン Set (Create) Window は、ウインドウ wn が存在する場合、これをアクティブにします。また、存在しない場合は同時にこれを作成します。Delete ボタンは、ウインドウ wn が存在する場合、これを閉じます。プロットコマンドの実行時には、必要な応じて自動的にウインドウが作成されます。

グラフィックウインドウには、次の 4 つのボタンがあります。

- 3D Rot.: 3D プロットにおいてマウスで回転を行います。このボタンは、2D プロットでは使用できない。
- 2D Zoom: 2D プロットのズームを行います。このコマンドは、再帰的に実行可能です。このボタンは、3D プロット用ではありません。
- UnZoomx: 最初のプロットに戻します。(複数のズームを行った場合は、プロットが直前のズームに一致するわけではありません。)
- File: このボタンは、別のコマンドとメニューを開きます。

  最初のは簡単です。: Clear は、単にウインドウ上のプロットを消去します。

  次のコマンド Print... は、プロットの紙への出力を得るための選択パネルを開きます。

  Export コマンドは、指定されたフォーマット (Postscript,Latex) ファイルでファイルにプロットのコピーを作成するための選択パネルを開きます。

  save コマンドは、指定された名前でファイル上にプロットを直接保存します。このファイルは、後に Scilab で再プロットのためにロードすることが可能です。

  Delete は、(close) と同じコマンドです。しかし、単にそのウインドウに用いられます。

5.2　メディア

Scilab には、グラフィックスをウインドウや紙に送るために使用できる幾つかのグラフィックスデバイスがあります。デフォルトは、ScilabGraphic0 ウインドウです。

基本的な Scilab グラフィックスコマンドは、次のようなものです。

- driver: グラフィックスドライバを選択します。
- xinit: グラフィックスドライバを初期化します。
- xclear: 一つ以上のグラフィックスウインドウを消去します。
- xpause: ミリ秒単位で動作を中断します。
- xselect: カレントのグラフィックウインドウを上に持って来ます。
- xclick: マウスクリックを待ちます。
- xend: グラフィックセッションを閉じます。

デバイスの種類を以下に示します。

- X11 : X11 ウインドウシステム用グラフィックデバイス
- Rec : 全てのグラフィックコマンドも記録する X ウインドウドライバ (X11) 。デフォルトのドライバです。
- Wdp : グラフィックスの記録を行わない X11 ドライバグラフィックは、pixmap に行われ、コマンド xset("wshow") によりグラフィックウインドウに送られます。pixmap は、コマンド xset("wwpc") または、通常のコマンド xbasc() により消去されます。
- Pos : Postscript プリンタ用グラフィックスデバイス
- Fig : Xfig システム用プリンタ用グラフィックスデバイス

実際、多くの場合、ドライバの存在は無視することができ、Xfig システムに関してグラフィックをプリンターあるいはファイルに送るために関数 xbasimp、xs2fig を使用することができます。以下に例を示します。

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

-->xinit('foo.ps')

-->plot(1:10)

-->xend()

-->driver('Rec')

-->plot(1:10)

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

2 つの同じポストスクリプトファイル、'foo.ps' と 'foo1.ps.0' が作成されます。(付加された 0 は、プロットが行われたアクティブウインドウの番号です。)

プロットのデフォルトは、重ね合わせモードです。これは、ウインドウ window-number に関連する記録された Scilab グラフィックスコマンドを消去し、このウインドウを消去するコマンド xbasc(window-number) により避けることができます。(xbasc と xclear に関する以下の警告を参照して下さい。)

グラフィックウインドウを拡大した場合、コマンド xbasr(window-number) が Scilab により実行されます。このコマンドは、グラフィックスウインドウ window-number を消去し、このウインドウに関連するグラフィックコマンドを再実行します。関連する記録されたグラフィックコマンドを確かめるためにこの関数を手動で呼ぶことが可能です。

ボタンまたはコマンド xset、xselect により、グラフィックウインドウはいくつでも作成できます。環境変数 DISPLAY は、X11 ディスプレイを指定するために使用可能です。また、指定したディスプレイ上にグラフィックウインドウを開くために xinit 関数を使用することも可能です。

5.3　2D プロット

5.3.1　基本的な 2D プロット法

最も簡単な 2D プロットは、plot(x,y) または plot(y) です。これは、x の関数 y のプロットです。ただし、x と y は、共にベクトルです。x が与えられない場合、ベクトル (1,...,size(y)) で代用されます。y が行列の場合、各行がプロットされます。オプションの引数があります。

最初の例では、次のようなコマンドを与えます。結果を図 5.1 に示します。

```
-->//first example of plotting

-->  t=(0:0.05:1)';

-->  ct=cos(2*%pi*t);

-->  plot2d(t,ct)
```

一般的な複数の 2 次元プロットは、以下のようなものです。

plot2di(str,x,y,[style,strf,leg,rect,nax])

ただし、i=missing,1,2,3,4

異なった i の値に関して、次のように分かれます。

i=missing : 区分線形プロット

i=1 : 上と同じだが、対数軸でプロット

i=2 : 区分定数描画スタイル

i=3 : 垂直バー

i=4 : 矢印スタイル (例えば、位相空間における常微分方程式)

-パラメータ str : 文字列 "abc" :

i がない場合、str は空です。

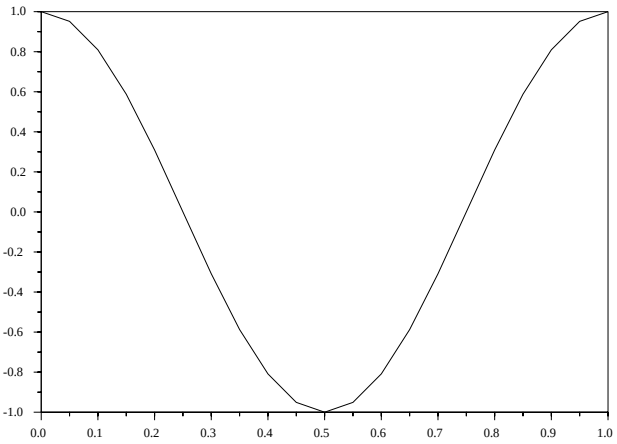


図 5.1: プロットの最初の例

a=e : 空 (empty) を意味します。x の値は使用されません。(ユーザーは、x のダミーの値を与える必要があります。)

a=o : 1 (one) を意味します。x の値は、全ての曲線に関して同じです。

a=g : 一般 (general) を意味します。

b=l : 対数軸スケールを X 軸に用います。

c=l : 対数軸スケールを Y 軸に用います。

-パラメータ x,y : 同じ大きさ [nl,nc] の 2 つの行列 (nc は曲線の数、nl は各曲線の点の数)

-パラメータ style : 大きさ (1,nc) の実数ベクトル。曲線 j に関して使用するスタイルは、size(j) により定義されます。(一つの曲線のみが描かれる時、 style は、キャプションを使用する際のスタイルと位置を指定することが可能です。)

-パラメータ strf : 以下の文字列に一致する長さ 3 "xyz" の文字列。:

x=1 : キャプションを表示

y=1 : プロットの境界を指定するために引数 rect を使用する。
rect=[xmin,ymin,xmax,ymax]

y=2 : プロットの境界を計算する。

y=0 : 現在の境界。

z=1 : 軸を描く際に目盛の数を引数 nax により指定することが可能です。

z=2 : プロットは、箱により囲われるだけです。

-パラメータ leg : キャプションの文字列であり、プロット曲線毎に指定します。この文字列は、記号 @ により区切られたフィールドから構成されます。 例えば、‘‘module@phase’’ (以下の例を参照して下さい。) この文字列は、対応する曲線のスタイルを連想させる小さな線と共にプロットに表示されます。

-パラメータ rect : プロットの境界を指定する 4 つの値からなるベクトル。rect=[xmin,ymin,xmax,ymax]

plot 関数毎に、引数なしでコマンドのみを実行するとデモが表示されます。(例えば、plot2d3())

5.3.2　特別な 2 次元プロット

- champ：R^2 のベクトルフィールド
- fchamp：ある関数により定義された R^2 のベクトルフィールド
- fplot2d：ある関数により定義された曲線の 2 次元プロット
- fgrayplot：2 次元プロット上のグレイレベル
- errbar：誤差バー付きのプロットを作成します。

5.3.3　キャプションとプレゼンテーション

- xgrid：2 次元グラフィックにグリッドを付加します。
- xtitle：2 次元グラフィックにタイトルと軸の名称を付加します。
- titlepage：グラフィックのタイトルページ
- plotframe：スケール、グリッド付きグラフィックのフレーム

グラデーション番号を選び、丸め番号を得ることにより、グリッドとグラデーションを付加するためにコマンド plotframe が使用されます。

5.3.4　幾何学的図のプロット

多角形のプロット

- xsegs：接続していない線分の組みを描きます。
- xrect：長方形を一つ描きます。
- xfrect：長方形を一つ塗りつぶします。
- xrects：長方形の組を塗りつぶすか、描きます。
- xpoly：多角形を描きます。
- xpolys：多角形の組みを描きます。
- xfpoly：多角形を塗りつぶします。
- xfpolys：多角形の組みを塗りつぶします。
- xarrows：接続していない矢印の組みを描きます。
- xfrect：一つの長方形を塗りつぶします。
- xclea：グラフィックウインドウ上の長方形を消去します。

曲線のプロット

- xarc：楕円を描きます。
- xfarc：楕円を塗りつぶします。
- xarcs：一組みの楕円を塗りつぶすまたは描きます。

5.3.5　プロットに書き込む

- xstring：文字列または文字列の行列を描きます。
- xstringl：文字列を囲む長方形を計算します。
- xstringb：指定した箱に文字列を描きます。
- xnumb：数字の組を描きます。

5.3.6　プロットの加工とグラフィックスコンテキスト

グラフィックスコンテキスト

グラフィックスのいくつかのパラメータは、グラフィックコンテキストにより制御されます。(例えば、線の太さ) そして、その他のパラメータは、グラフィックス引数により制御されます。最初の例で (図 (5.1))、全てデフォルトの引数を用います。

- xset：グラフィックコンテキストの値をセットします。xset の使用例の幾つかを示します。:

  (i)-xset("use color",flag) は、色 (color) または flag の値 (1 または 0) に対応するモノクロプロットに変更します。

  (ii)-xset("window",window-number) は、カレントウインドウをウインドウ window-number にセットし、そのウインドウが存在しない場合、それを作成します。

  (iii)-xset("wpos",x,y) は、グラフィックウインドウの左上の点の位置を指定します。

  フォントの選択、ウインドウの幅と高さ、ドライバ... は、xset で設定できます。

- xget：カレントのグラフィックコンテキストに関する情報を得ます。xset により修正されるパラメータの全ての値は、xget により得ることができます。
- xlfont：X ウインドウマネージャから新しいフォントファミリーをロードします。

いくつかの操作

座標変換：

- isoview：ウインドウの変更を行わない等長スケーリング

  この関数は、前にプロットしたウインドウにおいてウインドウの大きさを変えること無しに等長スケーリングを可能にします。(以下の例を参照して下さい。)

- square : ウインドウのサイズ変更を伴う等長スケーリング

  コマンドパラメータに対応してウインドウのサイズが変更されます。

- scaling : データのスケーリング

- rotate : 回転

  scaling と rotate は、複数点の値 (x,y) に対応する 2 行の行列についてアフィン変換と幾何学的回転をそれぞれ実行します。

- xgetech, xsetech : グラフィックウインドウの内部でスケールを変更します。

  カレントのグラフィックスケールは、高レベルプロットコマンドにより変更可能です。この値を得たり、直接修正したいと思うかもしれません。: これが、xgetech, xsetech の役割です。

5.4　いくつかの例

ここでは、2 次元プロットに関する別の機能と図の作成に対応する一連のコマンドを示します。この最初の例は、図 5.3 に対応します。

```
x=-%pi:0.3:%pi;
y1=sin(x);y2=cos(x);y3=x;
X=[x;x;x]; Y=[y1;y2;y3];
plot2d2("g00",X',Y');
xbasc();
plot2d1("g00",X',Y',[1 2 3]','101","caption1@caption2@caption3");
xtitle(["General";"Title"],"x-axis title","y-axis title");
xgrid([10,20]);
xclea(0.5,-0.5,1.5,1.5);
titlepage("titlepage");
xstring(0.65,0.3,["xstring after";"xclear"],0,1);
```

次に図 5.5 を作成するための一連のコマンドを示します。

```
// initialize default environment variables
plot(1:10)
xbasc()
// simple rectangle
xrect(0,1,3,1)
// filling a rectangle
xfrect(3.1,1,3,1)
// writing in the rectangle
xstring(0.5,0.5,"xrect(0,1,3,1)")
// writing black on black !
xstring(4.,0.5,"xfrect(3.1,1,3,1)")
```

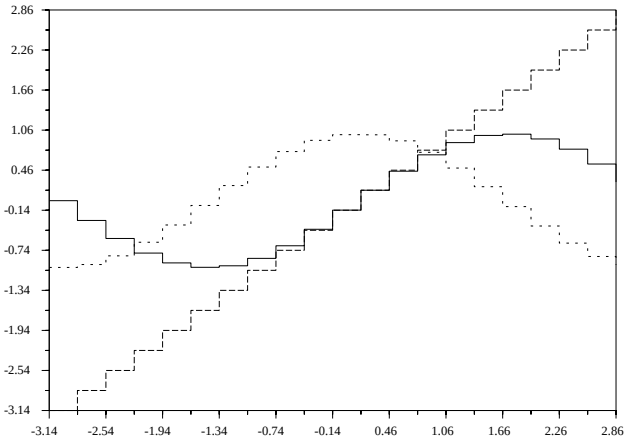


図 5.2: 簡単な 2 次元プロット

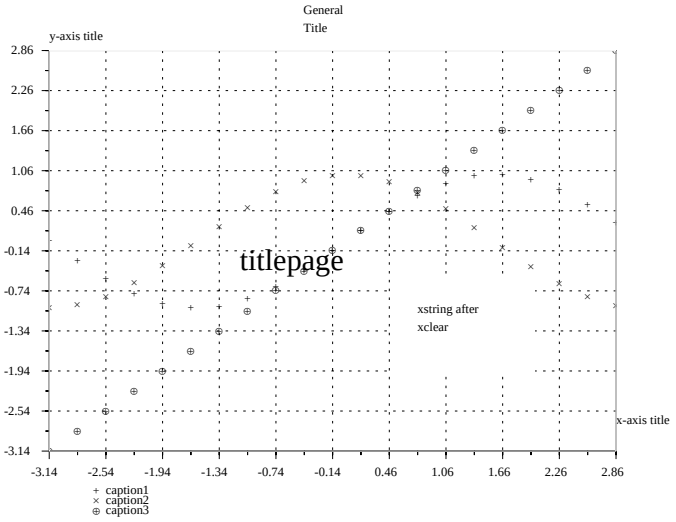


図 5.3: 2 次元プロットのいくつかの機能

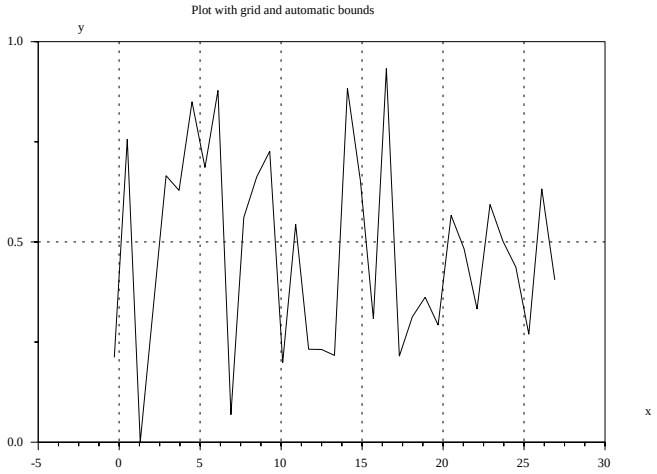


図 5.4: plotframe の使用

```
// reversing the video
xset("alufunction",6)
xstring(4.,0.5,"xfrect(3.1,1,3,1)")
xset("alufunction",3)
// drawing a polyline
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)")
// drawing arrows
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)")
// drawing a part of an ellipsis
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)
xstring(5.5,4.,"xfarc(5.,5.,4.,2.,0.,64*360.)")
xset("alufunction",3)
// writing a string
xstring(0.,4.5,"WRITING-BY-XSTRING()",-22.5)
```

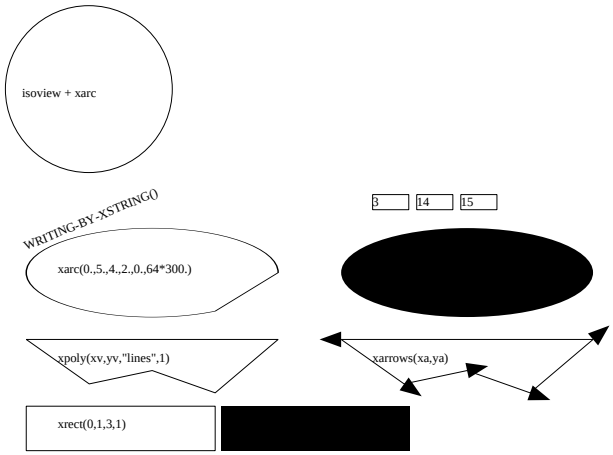


図 5.5: 幾何学的グラフィックスとコメント

5.5　3 次元プロット

5.5.1　基本的な 3 次元プロット

- `plot3d`：点行列の 3 次元プロット ：3 つの行列 x,y,z を plot3d(x,y,z) に用います。この場合、x,y 座標系で点の値が z になります。他の引数はオプションです。
- `plot3d1`：行列点の等高線付き 3 次元プロット
- `fplot3d`：関数で記述される表面の 3 次元プロット。この場合、z は、外部関数 z=f(x,y) で与えられます。
- `fplot3d1`：ある関数で記述される表面の等高線付き 3 次元プロット

5.5.2　特別な 3 次元プロット

- `param3d`：3 次元空間でパラメータ表現の曲線をプロットします。
- `contour`：ある行列により与えられる 3 次元関数の等高曲線
- `grayplot10`：2 次元プロット上に等高線を描きます。
- `fcontour10`：ある関数により与えられる 3 次元関数の等高曲線
- `hist3d`：3 次元棒グラフ
- `secto3d`：扇形から plot3d 互換データに面の表現の変換を行います。
- `eval3d`：標準グリッド上で関数を評価します。(feval も参照して下さい。)

5.5.3　2 次元グラフィックスと 3 次元グラフィックスの組合せ

3 次元プロット関数を使用する際、デフォルトのグラフィック境界は固定ですが、R^3 空間にあります。3 次元グラフィックに情報を加えるためにグラフィックプリミティブを使用したい場合、3 次元座標を 2 次元グラフィック座標に変換するために geom3d 関数を使用することが可能です。図 5.6 はこの機能を説明しています。

```
xinit('d7-10.ps');
   r=(%pi):-0.01:0;x=r.*cos(10*r);y=r.*sin(10*r);
   deff("[z]=surf(x,y)","z=sin(x)*cos(y)");
   t=%pi*(-10:10)/10;
   fplot3d(t,t,surf,35,45,"X@Y@Z",[-1,2,3]);
   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)');
```

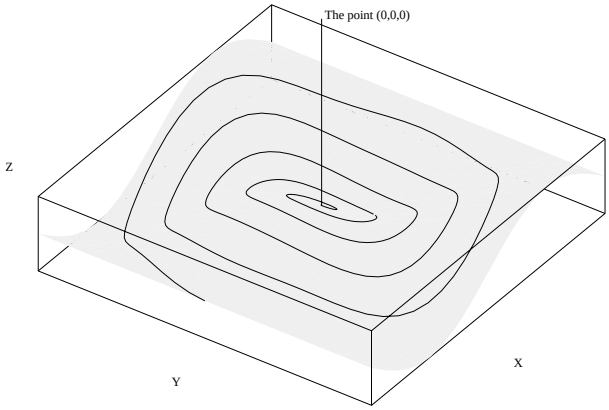


図 5.6: 2 次元プロットと 3 次元プロット

5.5.4　サブウインドウ

同じグラフィックウインドウに複数のプロットを作成することが可能です。(図 5.7)

```
xinit('d7-8.ps');
  t=(0:.05:1)';st=sin(2*%pi*t);
  xsetech([0,0,1,0.5]);
```

```
plot2d2("onn",t,st);
xsetech([0,0.5,1,0.5]);
plot2d3("onn",t,st);
xsetech([0,0,1,1]);
```

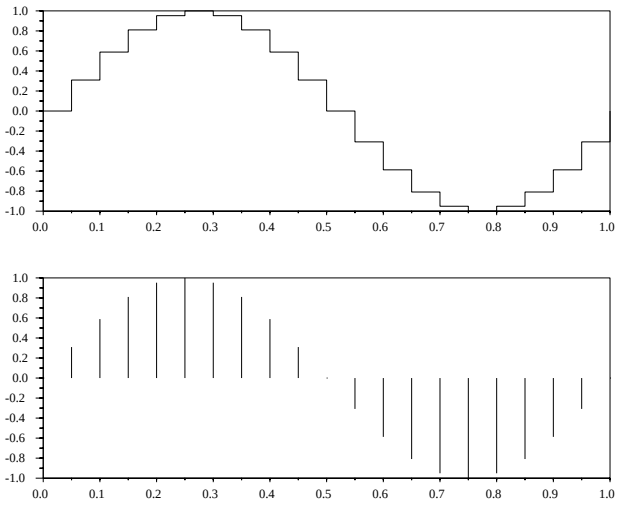


図 5.7: xsetech の使用

5.5.5　図の組

次の例では、2 次元グラフィックスと 3 次元グラフィックスに関して異なったプロット関数の簡単な概要を示します。図 5.8 が作成され、コマンド Blatexprs のヘルプと共にこのドキュメントに挿入されています。

```
//some examples
str_l=list();
//
str_l(1)=['plot3d1();';
      'title=[''plot3d1 : z=sin(x)*cos(y)''];';
      'xtitle(title,'' '','' '');'];
//
str_l(2)=['contour();';
      'title=[''contour ''];';
      'xtitle(title,'' '','' '');'];
//
str_l(3)=['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
```

5.6　Scilab グラフィックスの出力と LaTeX への挿入

Scilab グラフィックを処理し、結果を印刷するための (UNIX シェル) プログラムの使用法をここで説明します。これらのプログラムは、Scilab のサブディレクトリ `bin` にあります。

5.6.1　ウインドウから紙へ

プロットを印刷する最も簡単なコマンドは、ScilabGraphic ウインドウの `print` ボタンをクリックすることです。

5.6.2　ポストスクリプトファイルの作成

Scilab プロットを含むポストスクリプトファイルを作成するための最も簡単な方法をこの章の始めに示しました。これは、次のようなものです。

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

-->xinit('foo.ps')

-->plot3d1();

-->xend()

-->driver('Rec')

-->plot3d1()

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

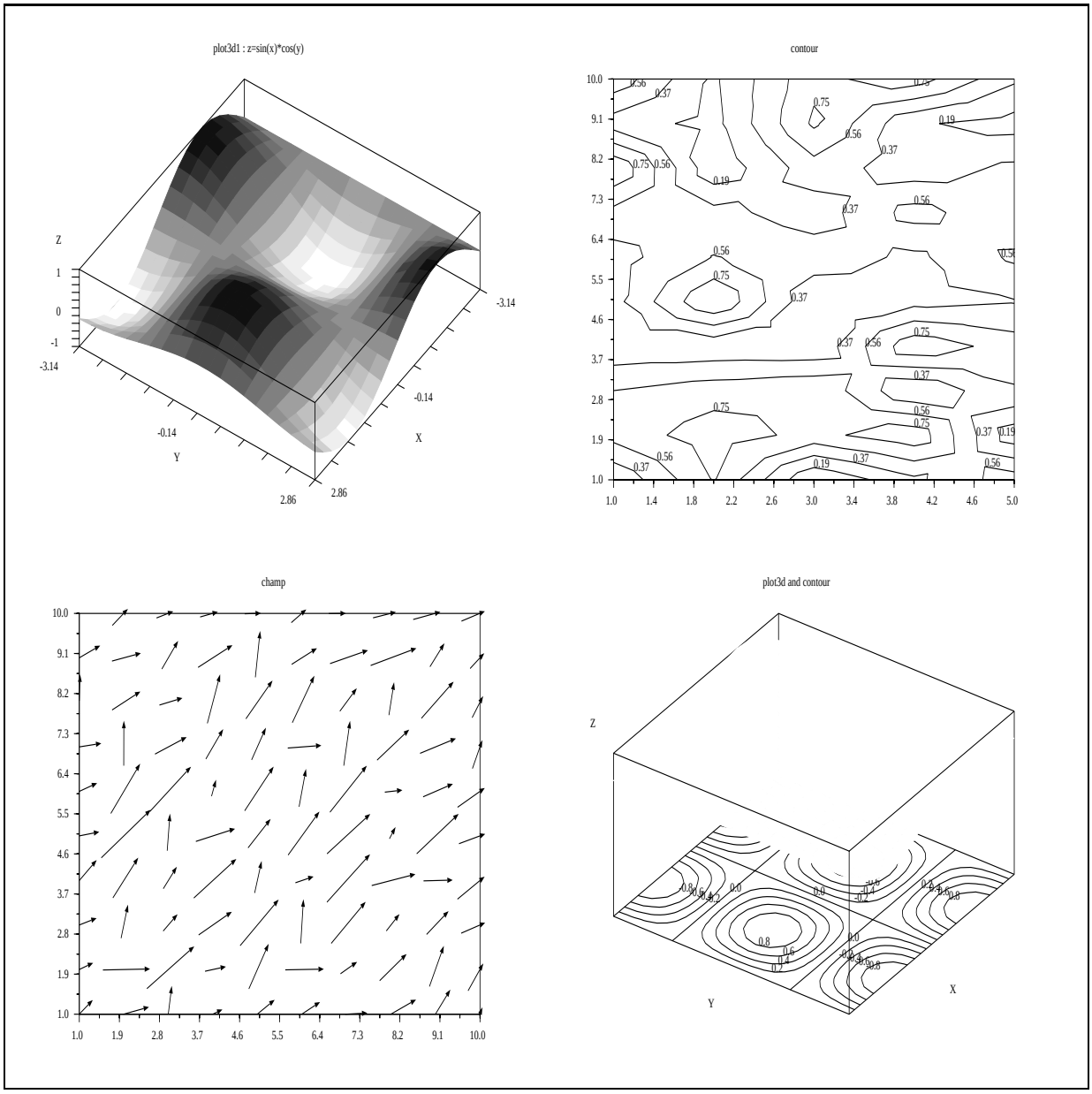

plot3d1 : z=sin(x)*cos(y)
Z
1
0
-1
-3.14
-0.14
Y
2.86
-3.14
-0.14
X
2.86
contour
champ
plot3d and contour
Z
Y
X

図 5.8: 図のグループ

Scilab により作成されたポストスクリプトファイル (foo.ps　または foo1.ps) を直接ポストスクリプトプリンタへ送ることはできません。プリアンブルが必要です。そこで、印刷は、Scilab に付属する UNIX スクリプトまたはプログラムを介して行います。プログラム Blpr は、一枚の紙に 1 組みの Scilab グラフィックスを印刷するために使用されます。このコマンドは、次のように使用します。

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

この後、ファイル result を次のような古典的な Unix コマンドで印刷することが可能です。

```
lpr -Pprinter-name result
```

または、結果を見るためには、自分の Unix ワークステーション上の ghostview ポストスクリプトインタプリタを使用して下さい。

パイプを使用することによりファイル result を作成することを回避することができます。この場合、> result を出力コマンド | lpr またはプレビューコマンド | ghostview - に交換します。

最良の結果 (最良の大きさの図) は、1 ページに 2 つの図を印刷する時に得られます。

5.6.3　ポストスクリプトファイルの LaTeX への読み込み

Blatexpr Unix シェルとプログラム Batexpr2 と Blatexprs は、LaTeX で Scilab グラフィックスを挿入することを補助するために提供されます。

前に作成されたファイル foo.ps がある時、次のような命令を Unix シェルで打ち込みます。

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

すると、2 つのファイル foo.epsf と foo.tex が作成されます。元のポストスクリプトファイルは、無変更のまま残ります。LaTeX ドキュメントに図を挿入するために、次のような LaTeX コードを LaTeX ドキュメントに含める必要があります。

```
\input foo.tex
\dessin{図のキャプション}{The-label}
```

また、ポストスクリプトプレビュア ghostview を用いても図を見ることができます。

プログラム Blatexprs は同様のことを行います。: 1 枚の LaTeX の絵の中に複数のポストスクリプトの図を挿入する際に使用します。

次の例では、最初にポストスクリプトドライバ Pos を使用します。続いて、連続的にそれぞれ別のプロットを行った 4 つのポストスクリプトファイル fig1.ps, ..., fig4.ps を初期化し、最後にドライバ Rec (記録付き X11 ドライバ) に戻ります。

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

-->driver('Pos')

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

```
-->

-->plot3d1(t,t,sin(t)'*cos(t),35,45,'X@Y@Z',[2,2,4]);

-->xend()

-->

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

-->xend()

-->

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

-->xend()

-->

-->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,' ',' ');

-->xend()

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

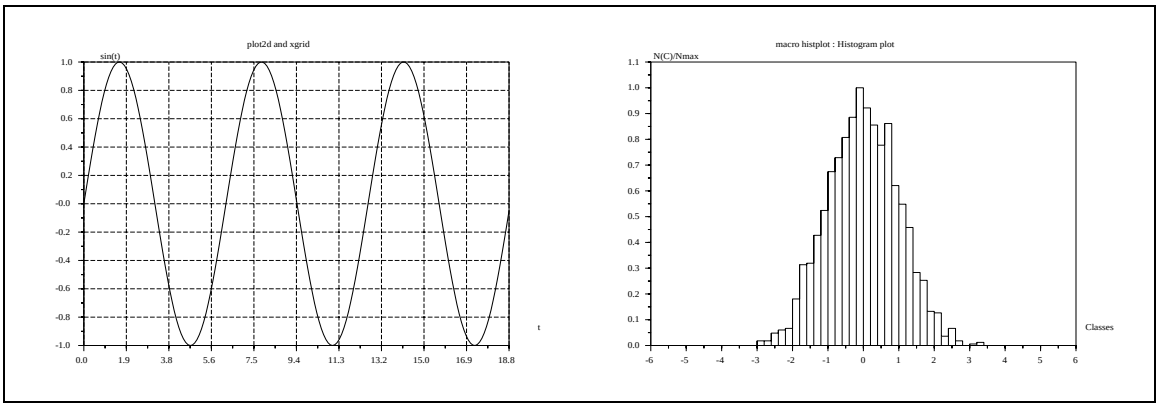


図 5.9: Blatexp2 の例

続いて次のコマンドを実行します。

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

これにより、2 つのファイル `multi.tex` と `multi.ps` が得られます。次のようにすることにより、LaTeX ソースファイルに結果を挿入することができます。

```
\input multi.tex
\dessin{図のキャプション}{The-label}
```

2 番目の行 `dessin...` は必ず必要であり、別のディレクトリで作業している場合 (以下の例を参照) には、当然、挿入するファイルの絶対パスを与える必要があることに注意して下さい。ファイル `multi.tex` は、2 つのパラメータ、キャプションとラベルを有するコマンド `dessin` の定義のみです。コマンド `dessin` は、キャプションまたはラベルを使用したくない場合には、1 つまたは 2 つの空の引数 `‘‘ ’’` を使用することが可能です。

ポストスクリプトファイルは、`\special` コマンドにより LaTeX に挿入します。この構文は、`dvips` プログラムと互換性があります。

プログラム `Blatexpr2` は、2 つの図を横に並べる場合に使用します。

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

メインの LaTeX ドキュメントをあるディレクトリに置き、全ての図をサブディレクトリに置いておくことは、しばしば便利です。絵がサブディレクトリ `figures` に置かれている場合に、その絵をメインのドキュメントに挿入する適当な方法は、次のようなものです。

```
\def\Figdir{figures/} % My figures are in the {\tt figures/ } subdirectory.
\input{\Figdir fig.tex}
\dessin{絵のキャプション}{The-label}
```

`\def\Figdir{figures/}` の宣言が 2 回使用されています。最初は (`latex` を使用する際に) ファイル `fig.tex` を見付けるため、2 回目は `fig.tex` の中の `special` LaTeX コマンド用に (dvips レベルで使用される) 正しいパス名を生成するためです。

-注意：デフォルトのドライバは、Rec です。すなわち、一つのレコードが一つのウインドウに対応するように、すべてのグラフィックコマンドは記録されます。xbasc() コマンドはアクティブウインドウのプロットとこのウインドウに対応する全ての記録を消去します。clear ボタンも同じ効果を有しています。xclear コマンドは、プロットを消去しますが、記録は保存されます。よって、コマンド xbasc() または clear を使用する必要はほとんどないでしょう。このようなコマンドを使用した時にプロットを再実行すると (環境が消去されることを忘れていた場合、) 奇妙な結果が得られるでしょう。スケールのみが保存されており、“window-plot” と完全に別の “paper-plot” が得られるでしょう。

5.6.4　Xfig 使用によるポストスクリプト

プロットに関するポストスクリプトファイルを得るための別の便利な手法は Xfig を使うことです。簡単なコマンド xs2fig(active-window-number,file-name) により Xfig 形式のファイルを得ることができます。

このコマンドは、ドライバ Rec を使用します。

ウインドウ ScilabGraphic0 がアクティブである時、次のように入力すると、

```
-->t=-%pi:0.3:%pi;

-->plot3d1(t,t,sin(t)'*cos(t),35,45,'X@Y@Z',[2,2,4]);

-->xs2fig(0,'demo.fig');
```

ウインドウ 0 のプロットを含むファイル demo.fig が得られます。

続いて、Xfig を使用してすることができます。好きなように修正した後、LaTeX ファイルに挿入することができるようにポストスクリプトファイルを作成します。次の図は、Xfig でいくつかのコメントを加えることにより作成されたものです。

5.6.5　Encapsulated Postscript ファイル

前に述べたように、Blatexpr の使用により 2 つのファイルが作成されます。つまり、LaTeX ファイルに挿入される .tex ファイルと .epsf ファイルです。

BEpsf を使用することにより、.ps ファイルと等価な encapsulated Postscript ファイルを得ることができます。

Blatexpr により作成された .epsf ファイルは、encapsulated Postscript ファイルでないことに注意して下さい。このファイルには、bounding box がありません。BEpsf は、bounding box を有する encapsulated Postscript ファイルである .eps ファイルを作成します。

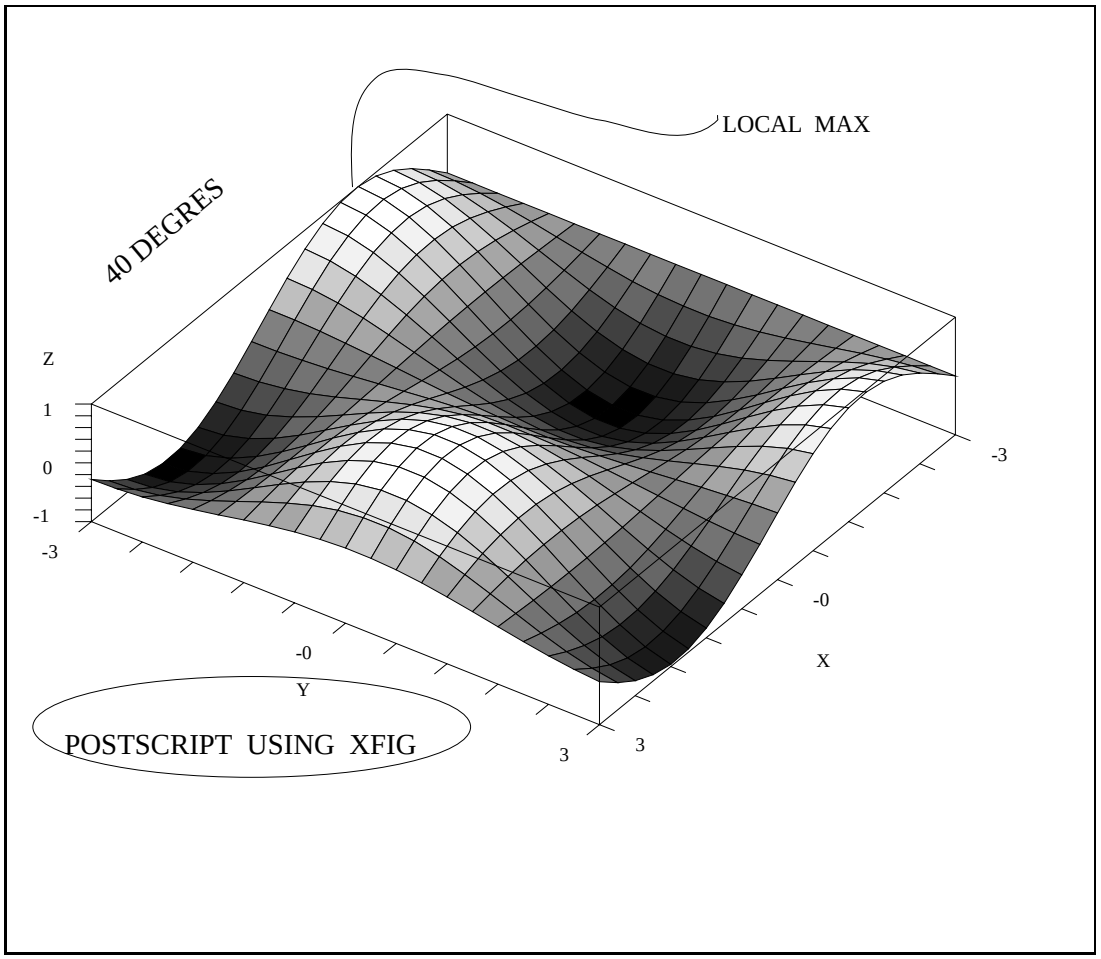


図 5.10: Xfig の使用による Encapsulated Postscript

第6章　Maple と Scilab のインターフェース

数式処理システム Maple の数式処理計算を Scilab の数値演算機能と組み合わせるために、Maple オブジェクトを Scilab 関数に変換することが可能です。

数値による評価の効率を確保するためにこれは Fortran による数値的な評価により行われます。変換のプロセス自体は、`maple2scilab` という Maple プロシージャにより行われます。

6.1　Maple2scilab

関数 `maple2scilab` は、スカラー関数か行列であるかによらず、Maple オブジェクトを Fortran サブルーチンに変換し、関連する Scilab 関数を作成します。`maple2scilab` のコードは、ディレクトリ `SCIDIR/maple` にあります。

`maple2scilab` の呼び出し手順は、次のようなものです。

```
maple2scilab(function-name,object,args)
```

- 最初の引数、`function-name` は、Scilab の関数名を示す名前です。
- 2 番目の引数、`object` は、Scilab に変換する式の Maple における名前です。
- 3 番目の引数は、Maple-object `object` の式パラメータを含む引数のリストです。

`maple2scilab` が Maple において実行された場合、2 つのファイルが作成されます。一つは、Fortran コードを含み、もう一つは、関連する Scilab 関数を含んでいます。それらが、存在する限り、ユーザーはその内容について知る必要はありません。

作成される Fortran ルーチンは、 次のような呼び出し手順を有しています。

```
<Scilab-name>(x1,x2,...,xn,matrix)
```

このサブルーチンは、引数 `x1,x2,...,xn` の関数として行列 matrix(i,j) を計算します。各引数は、Maple スカラーまたは引数のリストである配列とすることができます。Fortran サブルーチンは `<Scilab-name>.f` という名前のファイルに、Scilab 関数は `<Scilab-name>.sci` という名前のファイルに作成されます。Scilab における数値による評価を行うために、ユーザーは Fortran サブルーチンをコンパイルし Scilab にリンクし (例えば、Menu-bar のオプション '`link`')、関連する関数をロードする必要があります。(Menu-bar オプション '`getfc`') `link` 操作に関する情報は、Scilab のマニュアルに与えられています。Fortran ルーチンは、動的リンクまたは `default` ディレクトリの `interf.f` ファイルにより、Scilab に読み込むことができます。もちろん、この 2 段階の手順は、シェルスクリプトにより (または、Scilab の `unix` を使用して、) 自動化することができます。

Maple2scilab は、Maple の共有ライブラリである “Macrofort” ライブラリを使用します。

6.1.1　簡単なスカラーの例

Maple セッション

```
> read(‘maple2scilab.maple‘):
> f:=b+a*sin(x);

                              f := b + a sin(x)

> maple2scilab(’f_m’,f,[x,a,b]);
```

ここで、Maple 変数 f はスカラー式ですが、Maple ベクトルまたは行列とすることもできます。Scilab における f の名前は、 ’f_m’ になります。(Scilab の名前は、6 文字以下に制限されることに注意して下さい。) 関数 maple2scilab は、2 つのファイル、f_m.f と f_m.sci を Maple が動作開始したディレクトリに作成します。別のディレクトリを指定する場合は、次のように Maple にパスを指定します。rpath:=‘/work/‘ ;

　すると、全てのファイルは、サブディレクトリ work に作成されます。ファイル f_m.f は、スタンドアローンの Fortran ルーチンを含んでいます。このルーチンは、ファイル f_m.sci で定義された関数 f_m により Scilab に動的にリンクされます。

Scilab セッション

```
-->unix(’make f_m.o’);

-->link(’f_m.o’,’f_m’);

linking  _f_m_ defined in f_m.o

-->getf(’f_m.sci’,’c’)

-->f_m(%pi,1,2)
 ans       =

    2.
```

6.1.2　行列の例

　以下は、Maple の行列を Scilab に変換する例です。

Maple セッション

```
> with(linalg):read(‘maple2scilab.maple‘):

> x:=vector(2):par:=vector(2):

> mat:=matrix(2,2,[x[1]^2+par[1],x[1]*x[2],par[2],x[2]]);

                            [     2                     ]
                            [ x[1]  + par[1]  x[1] x[2] ]
```

```
                    mat :=  [                              ]
                            [     par[2]          x[2]     ]

> maple2scilab('mat',mat,[x,par]);
```

Scilab セッション

```
-->unix('make mat.o');

-->link('mat.o','mat')

linking  _mat_ defined in mat.o

-->getf('mat.sci','c')

-->par=[50;60];x=[1;2];

-->mat(x,par)
 ans       =

!   51.    2. !
!   60.    2. !
```

コードの生成　以下は、前記の 2 つの例において `maple2scilab` により作成されたコード (Fortran サブルーチンと Scilab 関数) です。

Fortran ルーチン

```
c
c     SUBROUTINE f_m
c
      subroutine f_m(x,a,b,fmat)
      doubleprecision x,a,b
      implicit doubleprecision (t)
      doubleprecision fmat(1,1)
         fmat(1,1) = b+a*sin(x)
      end

c
c     SUBROUTINE mat
c
      subroutine mat(x,par,fmat)
      doubleprecision x,par(2)
      implicit doubleprecision (t)
      doubleprecision fmat(2,2)
         t2 = x(1)**2
         fmat(2,2) = x(2)
         fmat(2,1) = par(2)
```

```
      fmat(1,2) = x(1)*x(2)
      fmat(1,1) = t2+par(1)
    end
```

Scilab 関数

```
function [var]=f_m(x,a,b)
var=fort('f_m',x,1,'d',a,2,'d',b,3,'d','out',[1,1],4,'d')
//end

function [var]=fmat(x,par)
var=fort('fmat',x,1,'d',par,2,'d','out',[2,2],3,'d')
//end
```

付 録A　A demo session

最初のデモに対応する Scilab セッションを以下に示します。

```
-->//SCILAB OBJECTS             1. SCALARS

-->a=1                //constant
 a  =

    1.

-->1==1               //boolean
 ans  =

  T

-->'string'           //character string
 ans  =

 string

-->z=poly(0,'z')      // polynomial with variable 'z' and with one root at zero
 z  =

    z

-->p=1+3*z+4.5*z^2    //polynomial
 p  =

                2
    1 + 3z + 4.5z

-->r=z/p              //rational
 r  =

          z
    -------------
                2
    1 + 3z + 4.5z
```

```
-->//SCILAB OBJECTS             2. MATRICES

-->a=[a+1 2 3
     0 0 atan(1)
     5 9 -1]            //constant matrix
 a  =

!   2.    2.    3.        !
!   0.    0.    0.7853982 !
!   5.    9.  - 1.        !

-->b=[%t,%f]            //boolean matrix
 b  =

! T F !

-->mc=['this','is';
    'a' ,'matrix']      //matrix of strings
 mc  =

!this  is      !
!              !
!a     matrix  !

-->mp=[p,1-z;
    1,z*p]              //polynomial matrix
 mp  =

!                2                   !
!   1 + 3z + 4.5z     1 - z          !
!                                     !
!                          2      3  !
!   1                 z + 3z + 4.5z  !

-->mp=[p 1-z]
 mp  =

!                2           !
!   1 + 3z + 4.5z     1 - z  !

-->mp=[mp;1 1+z*p]      //matrix polynomial
 mp  =
```

```
!                2                          !
!   1 + 3z + 4.5z     1 - z                 !
!                                           !
!                                2      3   !
!   1                 1 + z + 3z + 4.5z     !

-->f=mp/poly([1+%i 1-%i 1],'z')   //rational matrix
 f  =

!                  2                          !
!    1 + 3z + 4.5z        - 1                 !
!   ---------------     ---------             !
!                2   3              2         !
! - 2 + 4z - 3z + z     2 - 2z + z            !
!                                             !
!                                  2      3    !
!          1            1 + z + 3z + 4.5z     !
!   ---------------     -----------------     !
!                2   3                2   3   !
! - 2 + 4z - 3z + z    - 2 + 4z - 3z + z      !

-->//SCILAB OBJECTS             3. LISTS

-->l=list(a,-(1:5), mp,['this','is';'a','list'])   //list
 l  =


       l>1

!   2.    2.    3.        !
!   0.    0.    0.7853982 !
!   5.    9.  - 1.        !

       l>2

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

       l>3

!                2                          !
!   1 + 3z + 4.5z     1 - z                 !
!                                           !
```

```
!                                2      3 !
!   1                1 + z + 3z + 4.5z  !

       l>4

!this  is    !
!            !
!a     list  !

-->b=[1 0;0 1;0 0];c=[1 -1 0];d=0*c*b;x0=[0;0;0];

-->sl=syslin('c',a,b,c,d,x0)    //Linear system in state-space representation.
 sl  =


       sl(1)   (state-space system:)

 lss

       sl(2) = A matrix =

!   2.    2.    3.        !
!   0.    0.    0.7853982 !
!   5.    9.  - 1.        !

       sl(3) = B matrix =

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

       sl(4) = C matrix =

!   1.  - 1.    0. !

       sl(5) = D matrix =

!   0.    0. !

       sl(6) = X0 (initial state) =

!   0. !
!   0. !
```

```
!   0. !

       sl(7) = Time domain =

 c

-->slt=ss2tf(sl)               // Transfer matrix
 slt  =

!                          2                                2          !
!       - 10.995574 + s + s                    46 + 3s - s            !
!   ----------------------------      ----------------------------   !
!                             2   3                              2     !
!   6.2831853 - 24.068583s - s + s     6.2831853 - 24.068583s - s      !
!         3                                                           !
!      + s                                                            !

-->//                  OPERATIONS

-->v=1:5;v*v'                  //constant matrix
 ans  =

    55.

-->mp'*mp+eye                  //polynomial matrix
 ans  =

!               2     3        4                 2                    !
!   3 + 6z + 18z + 27z + 20.25z     2 + 3z + 4.5z                      !
!                                                                      !
!                2                         2     3     4     5        6 !
!   2 + 3z + 4.5z                   3 + 8z + 15z + 18z + 27z + 20.25z  !

-->mp1=mp(1,1)+4.5*%i          //complex
 mp1  =

real part


                2
    1 + 3z + 4.5z
imaginary part
```

```
    4.5

-->fi=c*(z*eye-a)^(-1)*b;       //transfer function evaluation

-->f(:,1)*fi                    //rationals
 ans  =


         column 1

!                                              2      3      4         !
!         - 10.995574 - 31.986723z - 45.480084z + 7.5z + 4.5z          !
!   ------------------------------------------------------------------  !
!                                    2           3           4         !
! - 12.566371 + 73.269908z - 113.12389z + 72.488936z - 17.068583z      !
!         5   6                                                        !
!      - 4z + z                                                        !
!                                                                      !
!                                         2                            !
!                        - 10.995574 + z + z                           !
!   ------------------------------------------------------------------  !
!                                    2           3           4         !
! - 12.566371 + 73.269908z - 113.12389z + 72.488936z - 17.068583z      !
!         5   6                                                        !
!      - 4z + z                                                        !

         column 2

!                                    2       3      4                 !
!                   46 + 141z + 215z + 10.5z - 4.5z                    !
!   ------------------------------------------------------------------  !
!                                    2           3           4         !
! - 12.566371 + 73.269908z - 113.12389z + 72.488936z - 17.068583z      !
!         5   6                                                        !
!      - 4z + z                                                        !
!                                                                      !
!                                      2                               !
!                            46 + 3z - z                               !
!   ------------------------------------------------------------------  !
!                                    2           3           4         !
! - 12.566371 + 73.269908z - 113.12389z + 72.488936z - 17.068583z      !
!         5   6                                                        !
```

```
!      - 4z + z                                                    !

-->m=[mp -mp; mp' mp+eye]       //usual Matlab syntax for polynomials
 m  =


         column 1 to 3

!                2                                          2 !
!   1 + 3z + 4.5z     1 - z               - 1 - 3z - 4.5z  !
!                                                           !
!                                2      3                   !
!   1                 1 + z + 3z + 4.5z   - 1               !
!                                                           !
!                2                                          2 !
!   1 + 3z + 4.5z     1                     2 + 3z + 4.5z  !
!                                                           !
!                                2      3                   !
!   1 - z             1 + z + 3z + 4.5z     1               !

         column 4

! - 1 + z              !
!                      !
!             2      3 !
! - 1 - z - 3z - 4.5z  !
!                      !
!   1 - z              !
!                      !
!             2      3 !
!   2 + z + 3z + 4.5z  !

-->[fi, fi(:,1)]                // ... or rationals
 ans  =


         column 1 to 2

!                          2                                 2          !
!       - 10.995574 + z + z                    46 + 3z - z            !
!   ----------------------------      ----------------------------   !
!                             2   3                              2     !
!   6.2831853 - 24.068583z - z + z     6.2831853 - 24.068583z - z      !
```

```
!         3                                                    !
!      + z                                                     !

         column 3

!                          2          !
!       - 10.995574 + z + z           !
!   ------------------------------    !
!                             2   3   !
!   6.2831853 - 24.068583z - z + z    !

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

-->num=f(2);den=f(3);           //operation on transfer matrix

-->//                  SOME NUMERICAL PRIMITIVES

-->inv(a)                       //Inverse
 ans  =

!   1.125  - 4.6154933  - 0.25 !
! - 0.625    2.705634     0.25 !
!   0.       1.2732395    0.   !

-->inv(mp)                      //Inverse
 ans  =

!                    2      3                                    !
!        - 1 - z - 3z - 4.5z                    1 - z            !
!   -------------------------------    -------------------------------  !
!             2     3     4        5              2     3     4        !
! - 5z - 10.5z - 18z - 27z - 20.25z    - 5z - 10.5z - 18z - 27z         !
!              5                                                    !
!      - 20.25z                                                     !
!                                                                   !
!                                                          2        !
!                  1                          - 1 - 3z - 4.5z       !
!   -------------------------------    -------------------------------  !
!             2     3     4        5              2     3     4        !
! - 5z - 10.5z - 18z - 27z - 20.25z    - 5z - 10.5z - 18z - 27z         !
!              5                                                    !
!      - 20.25z                                                     !
```

```
-->inv(sl*sl')                  //Product of two linear systems and inverse
 ans  =


       ans(1)   (state-space system:)

 lss

       ans(2) = A matrix =

!   5.9635231  - 2.6249762  - 6.3980513    2.7956764 !
!   2.7249932    6.0907542  - 5.8266323  - 8.9360032 !
!   0.           0.         - 5.1379153    0.6683049 !
!   0.           0.         - 0.6976228  - 4.9163619 !

       ans(3) = B matrix =

!   12.954406 !
! - 5.0993579 !
!   0.8307530 !
! - 2.9870086 !

       ans(4) = C matrix =

! - 2.3504729  - 3.5004798    3.4091491    6.2019961 !

       ans(5) = D matrix =

                                 2
    2.4292037 - 9.021D-17s + 0.5s

       ans(6) = X0 (initial state) =

!   0. !
!   0. !
!   0. !
!   0. !

       ans(7) = Time domain =

 c

-->w=ss2tf(ans)                 //Transfer function representation
```

```
 w  =

                                        2            3            4
    19.739209 - 151.22737s + 283.36517s + 30.351769s - 23.568583s
          5      6
       - s + 0.5s
    ------------------------------------------------------------------
                                                2    3   4
              1118.4513 + 127.00443s - 51.995574s - 2s + s

-->inv(ss2tf(sl)*ss2tf(sl'))    //Product of two transfer functions and inverse
 ans  =

                                        2            3            4
    39.478418 - 302.45474s + 566.73034s + 60.703538s - 47.137167s
           5   6
       - 2s + s
    ------------------------------------------------------------------
                                                  2    3    4
             2236.9027 + 254.00885s - 103.99115s - 4s + 2s

-->clean(w-ans)
 ans  =

    0
    -
    1

-->n=contr(a,b)                 //Controllability
 n  =

    3.

-->k=ppol(a,b,[-1-%i -1+%i -1])        //Pole placement
 k  =

!   0.1832061  - 0.3358779    1.740458  !
!   2.7463953    3.8167939    1.7650419 !

-->poly(a-b*k,'z')-poly([-1-%i -1+%i -1],'z')    //Check...
 ans  =

  - 3.109D-15 - 3.109D-15z
```

```
-->s=sin(0:0.1:5*%pi);

-->ss=fft(s(1:128),-1);        //FFT

-->xbasc();

-->plot2d3("enn",1,abs(ss)');      //simple plot

-->x=lyap(a,diag([1 2 3]),'cont')   //Lyapunov equation
 x  =

! - 1.4958251  - 4.3299851    0.6983300 !
! - 4.3299851  - 0.3504060    1.07333   !
!   0.6983300    1.07333      1.4379815 !

-->//              ON LINE DEFINITION OF MACRO

-->deff('[x]=fact(n)','if n=0 then x=1,else x=n*fact(n-1),end')

-->10+fact(5)
 ans  =

    130.

-->//                    OPTIMIZATION

-->deff('[f,g,ind]=rosenbro(x,ind)', 'a=x(2)-x(1)^2 , b=1-x(2) ,...
f=100.*a^2 + b^2 , g(1)=-400.*x(1)*a , g(2)=200.*a -2.*b ');

-->comp(rosenbro);[f,x,g]=optim(rosenbro,[2;2],'qn')
 norm of projected gradient lower than   0.0000000D+00

 g  =

!   0. !
!   0. !
 x  =

!   1. !
!   1. !
 f  =
```

```
    0.

-->//                    SIMULATION

-->a=rand(3,3)
 a  =

!   0.3616361    0.4826472    0.5015342 !
!   0.2922267    0.3321719    0.4368588 !
!   0.5664249    0.5935095    0.2693125 !

-->e=exp(a)
 e  =

!   1.8016766    0.9861359    0.9295708 !
!   0.6462788    1.7366226    0.7731203 !
!   1.0024892    1.1047133    1.7455217 !

-->deff('[ydot]=f(t,y)','ydot=a*y');comp(f)

-->e(:,1)-ode([1;0;0],0,1,f)
 ans  =

! - 6.303D-08 !
! - 5.028D-08 !
! - 6.558D-08 !

-->//                   SYSTEM DEFINITION

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


    s

-->h=[1/s,1/(s+1);1/s/(s+1),1/(s+2)/(s+2)]
 h  =

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

```
!        2              2 !
!   s + s     4 + 4s + s  !

-->w=tf2ss(h);

-->ss2tf(w)
 ans  =

!         1                 1             !
!   -------------         -----           !
!   1.373D-15 + s         1 + s           !
!                                         !
!           1             1 - 3.677D-15s  !
!   -----------------     --------------  !
!                    2                2   !
!   7.288D-14 + s + s       4 + 4s + s    !

-->h1=clean(ans)
 h1  =

!   1           1         !
!   -         -----       !
!   s         1 + s       !
!                         !
!     1           1       !
!   -----     ---------   !
!        2              2 !
!   s + s     4 + 4s + s  !

-->//             EXAMPLE: SECOND ORDER SYSTEM ANALYSIS

-->sl=syslin('c',1/(s*s+0.2*s+1))
 sl  =

         1
    -----------
                2
    1 + 0.2s + s

-->instants=0:0.05:20;

-->//             step response:
```

```
-->y=csim('step',instants,sl);

-->xbasc();plot2d(instants',y')

-->//             Delayed step response

-->deff('[in]=u(t)','if t<3 then in=0;else in=1;end');

-->comp(u);

-->y1=csim(u,instants,sl);plot2d(instants',y1');

-->//             Impulse response;

-->yi=csim('imp',instants,sl);xbasc();plot2d(instants',yi');

-->yi1=csim('step',instants,s*sl);plot2d(instants',yi1');

-->//              Discretization

-->dt=0.05;

-->sld=dscr(tf2ss(sl),0.05);

-->//               Step response

-->u=ones(instants);
Warning :redefining function: u


-->yyy=flts(u,sld);

-->xbasc();plot(instants,yyy)

-->//              Impulse response

-->u=0*ones(instants);u(1)=1/dt;

-->yy=flts(u,sld);

-->xbasc();plot(instants,yy)

-->//            system interconnexion
```

```
-->w1=[w,w];

-->clean(ss2tf(w1))
 ans  =

!   1           1            1           1         !
!   -         -----          -         -----       !
!   s         1 + s          s         1 + s       !
!                                                  !
!     1           1            1           1       !
!   -----     ---------      -----     ---------   !
!        2             2          2             2  !
!   s + s     4 + 4s + s     s + s     4 + 4s + s  !

-->w2=[w;w];

-->clean(ss2tf(w2))
 ans  =

!   1           1         !
!   -         -----       !
!   s         1 + s       !
!                         !
!     1           1       !
!   -----     ---------   !
!        2             2  !
!   s + s     4 + 4s + s  !
!                         !
!   1           1         !
!   -         -----       !
!   s         1 + s       !
!                         !
!     1           1       !
!   -----     ---------   !
!        2             2  !
!   s + s     4 + 4s + s  !

-->//               change of variable

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

-->horner(h,(1-z)/(1+z))  //bilinear transform
```

```
 ans  =

!   1 + z          1 + z       !
!   -----          -----       !
!   1 - z            2         !
!                              !
!             2              2 !
!   1 + 2z + z     1 + 2z + z  !
!   ----------     ----------  !
!                            2 !
!     2 - 2z       9 + 6z + z  !

-->//                 PRIMITIVES

-->H=[1.    1.    1.    0.;
   2.  - 1.    0.    1;
   1.    0.    1.    1.;
   0.    1.    2.  - 1];

-->ww=spec(H)
 ww  =

!   2.7320508 !
! - 2.7320508 !
!   0.7320508 !
! - 0.7320508 !

-->//             STABLE SUBSPACES

-->[X,d]=schur(H,'cont');

-->X'*H*X
 ans  =

! - 2.7320508    0.           0.           1.        !
!   0.         - 0.7320508  - 1.           0.        !
!   0.           0.           2.7320508    0.        !
!   0.           0.           0.           0.7320508 !

-->[X,d]=schur(H,'disc');

-->X'*H*X
 ans  =
```

```
!   0.7320508    0.           0.           1.        !
!   0.         - 0.7320508  - 1.           0.        !
!   0.           0.           2.7320508    0.        !
!   0.           0.           0.         - 2.7320508 !

-->//Selection of user-defined eigenvalues (# 3 and 4 here);

-->deff('[flg]=sel(x)','flg=0,ev=x(2)/x(3),if abs(ev-ww(3))<0.0001|abs(ev-ww(4))<0.00001 then

-->[X,d]=schur(H,sel)
 d  =

    2.
 X  =

! - 0.5705632  - 0.2430494  - 0.6640233  - 0.4176813 !
! - 0.4176813    0.6640233  - 0.2430494    0.5705632 !
!   0.5705632  - 0.2430494  - 0.6640233    0.4176813 !
!   0.4176813    0.6640233  - 0.2430494  - 0.5705632 !

-->X'*H*X
 ans  =

!   0.7320508    0.           0.           1.        !
!   0.         - 0.7320508  - 1.           0.        !
!   0.           0.           2.7320508    0.        !
!   0.           0.           0.         - 2.7320508 !

-->//               With matrix pencil

-->[X,d]=gschur(H,eye(H),sel)
 d  =

    2.
 X  =

!   0.5705632    0.2430494    0.6640233    0.4176813 !
!   0.4176813  - 0.6640233    0.2430494  - 0.5705632 !
! - 0.5705632    0.2430494    0.6640233  - 0.4176813 !
! - 0.4176813  - 0.6640233    0.2430494    0.5705632 !

-->X'*H*X
```

```
 ans  =

!   0.7320508    0.           0.           1.         !
!   0.         - 0.7320508  - 1.           0.         !
!   0.           0.           2.7320508    0.         !
!   0.           0.           0.         - 2.7320508  !

-->//            block diagonalization

-->[ab,x,bs]=bdiag(H);

-->inv(x)*H*x
 ans  =

!   2.7320508    0.           0.           0.         !
!   0.         - 2.7320508    0.           0.         !
!   0.           0.           0.7320508    0.         !
!   0.           0.           0.         - 0.7320508  !

-->//                    Matrix pencils

-->E=rand(3,2)*rand(2,3);

-->A=rand(3,2)*rand(2,3);

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

-->w=det(s*D-A)   //determinant
 w  =

                          2
  - 0.0801176s + 0.0423727s

-->[al,be]=gspec(A,E);

-->al./(be+%eps*ones(be))
 ans  =

!   1.576D+15  !
!   1.8907826  !
!   1.688D-16  !

-->roots(w)
```

```
 ans  =

!   0         !
!   1.8907826 !

-->[Ns,d]=coffg(s*D-A);    //inverse of polynomial matrix;

-->clean(Ns/d*(s*D-A))
 ans  =

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

-->[Q,M,i1]=pencan(E,A);   // Canonical form;
    rank A^k    rcond
        2.      0.347D+00
    rank A^k    rcond
        2.      0.552D+00

-->M*E*Q
 ans  =

!   1.           0.    0. !
! - 4.689D-16    1.    0. !
!   0.           0.    0. !

-->M*A*Q
 ans  =

!   1.8609512    0.4018602    0. !
!   0.1381447    0.0298314    0. !
!   0.           0.           1. !

-->//           PAUSD-RESUME
```

```
-->write(%io(2),'pause command...');
pause command...

-->write(%io(2),'TO CONTINUE...');
TO CONTINUE...

-->write(%io(2),'ENTER ''resume (or return) or click on resume!!''');
ENTER 'resume (or return) or click on resume!!'

-->pause;

-1->resume;

-->//           CALLING EXTERNAL ROUTINE

-->foo=['      subroutine foo(a,b,c)';
     '      c=a+b';
     '      end'  ];

-->unix_s('\rm foo.f')

-->write('foo.f',foo);

-->unix_s('make foo.o')     //Compiling...(needs fortran compiler)

-->//..................WARNING.....................

-->//NEXT COMMAND LINE WILL DO THE LINK OF THE ROUTINE foo WITH SCILAB

-->//THIS COMMAND MAY FAIL FOR SystemV COMPILED VERSIONS OF SCILAB

-->//BECAUSE LINK NEEDS THE LIBRARIES (SEE THE HELP OF "LINK" COMMAND)

-->link('foo.o','foo')    //Linking to Scilab

-->//5+7 by fortran

-->fort('foo',5,1,'r',7,2,'r','out',[1,1],3,'r')
 ans  =

    12.
```

```
-->
```

図 目 次