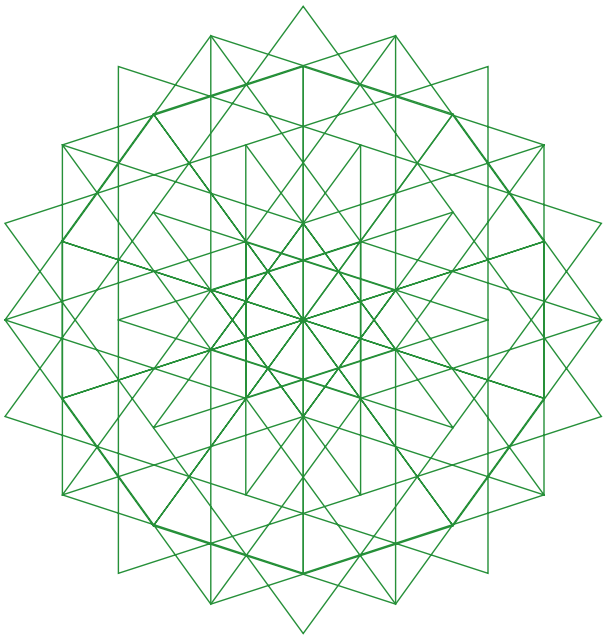
MATLAB を使いましょう



川上　博

1998

目次

第 I 部

MATLAB の基本

第 1 章

行列の入力と加工

MATLAB は行列の演算を基本とするソフトです．そこでまずは行列を作ることから始めましょう．その後で行列の要素をみたり，行列を加工する練習をしましょう．

1.1　ベクトルと行列要素の作り方

ここでは，とりあえず行列の入力例をみてみましょう．

1.1.1　「：」 と「[;]」 命令を使う

【例 1.1】

次の行列を入力してみよう．

1. ≫ A=0:10
2. ≫ t=0 : pi/2: 2*pi
3. ≫ B=[1 2 3; 4 5 6]
4. ≫ A=[B; 7 8 9]
5. ≫ H='Hello world!'

【解説】 「：」 命令は普通の行ベクトル（横長ベクトル）を作るのに便利です．

$$x = x_{min} \ : \ \Delta x : \ x_{max}$$

が基本です．「最小値：刻み幅：最大値」と入力すると，$[x_{min},\ x_{min} + \Delta x,\ x_{min} + 2\Delta x, \cdots, x_{max}]$ の横長ベクトルが出来上がり，このデータが変数 x に代入されます．刻み幅を省略すると，刻み幅は 1 と見なされます．

横長ベクトルは，グラフの横軸となるデータを作るのにとても便利です．たとえば

≫ t=0 : pi/30 : 2*pi;

≫ plot(t, sin(t))

とすれば正弦波のグラフが描けます．

行列の入力には「[」と「]」を使います．「；」は「行の区切り記号」です．したがって，例 1.1 の 3. では 2×3 （2 行 3 列）の行列

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

が作られて変数 B に代入されます．4. の例は，出来上がったこの行列 B に第 3 行 [7 8 9] をつけ加えた，次の 3×3 の行列 A ができます．

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

入力命令の後に「；」を付けると計算結果が画面上に出力されなくなります．長いベクトルを作った時やグラフのデータなど表示の必要な無いときに使うと便利です．　　≫ A=1:100;
文字列データは例 1.1 の 5. のように「'」と「'」で文字列を囲んで入力します．

1.1.2　数値アレーをつくって，くっ付ける演算子「 [ ]」

少し入力練習を続けましょう．一般に「[ ]」は複数の数値データをまとめて 1 つのベクトルや行列をつくる演算子です．数値アレー生成子 (array constractor) とでも呼んでおきましょう．「 [ ]」は数値アレーどおしをくっ付ける演算子コンカット (concatenation) でもある．

【練習 1.1】

次のベクトルや行列を入力してください．

1. $A = [5]$, $B = \begin{bmatrix} 2 & 1 & -1 \\ -1 & 5 & 1 \\ 1 & -1 & 2 \end{bmatrix}$, $C = \begin{bmatrix} 1+2j & \sqrt{2} \\ 3 & 5j \end{bmatrix}$, $D = \begin{bmatrix} 1-5j \\ 2+4j \\ 3-3j \\ 4+2j \\ 5-j \end{bmatrix}$

2. $A = \begin{bmatrix} 2 & 4 & 6 & 8 & 10 \\ 10 & 8 & 6 & 4 & 2 \end{bmatrix}$, $B = \begin{bmatrix} 1 & 3 & 5 & 7 & 9 \\ 9 & 8 & 7 & 6 & 5 \end{bmatrix}$

3. $A = \begin{bmatrix} 'Solving' & 'our' & 'problems' & 'with' & 'MATLAB.' \end{bmatrix}$, $B = ['I\ like'\ A]$

【注意】 1. $A = [5]$ は $A = 5$ と同じです．1×1 の行列はスカラーです．これを確かめるには $A = [5]$ と $B = 5$ と入力して，2 つを比較します．すなわち

≫ A==B

として下さい．「1 」が出力されると「$A = B$ が真」，「0 」が出力されると「偽」ということです．
3. j は虚数単位 $\sqrt{-1}$ です．i と j は虚数単位に予約された変数です．結果が i に変更されていても驚かないで下さい．$\sqrt{2}$ を入力するには $sqrt(2)$ とします．
4. これは「：」をうまく使って下さい．

$$A = [2 : 2 : 10\ ;\ 10 : -2 : 2]$$

あるいは

$$A = [(2 : 2 : 10)\ ;\ (10 : -2 : 2)]$$

と入力すると良いでしょう．「(」と「)」はカッコの中を優先して計算するように指定する記号です．読みやすくするために付けておいた方がいいでしょう．

≫ [A B]

≫ [A ; B]

とするとどんな行列が得られますか．
5. A と B の入力後，何が得られましたか．そう

I like Solving our problems with MATLAB.

となりましたか．単語がくっついてしまった人はいませんか．どうすればいいのでしょう．

1.2　スカラー，ベクトル，そして行列

1.2.1　行列の定義

スカラー，ベクトルそして行列などの説明ぬきで入力練習をしました．大体はこれまでに数学で習ったことから察しがついたと思います．MATLAB では，すべての入力データをアレー (array) [*1] と呼ばれるデータとして記憶します．よく使う数値アレーを表 1.1 に示しておきます．

この表で，次の 2 つのアレーに注目しておきましょう．これらは，MATLAB 特有のスカラーと行列です．

- NaN これは，”Not a Number” という定数です．0/0 などで現れます．また，数値アレーに挿んでグラフィックのデータとしておもしろい使い方ができます．覚えておきましょう．
- 空行列は「何もない行列」なのです．この不思議な行列は，行列から新しい行列を作ったり，行や列を削除したりするときに大活躍するでしょう．

遅くなりましたが，この辺であらためて行列 (matrix)[*2]の定義をしておきましょう．数 (scalar)[*3]を長方形に並べたものが行列です．縦に m 個，横に n 個並べた場合を，$m \times n$ （m 行 n 列）の行列と

[*1] アレーは，C 言語や Pascal それに Fortran などでは「配列」と訳されています．MATLAB では，これらの言語よりもっと広い意味で使われているようですから，アレーをそのまま訳さずに使うことにします．
[*2] 行列の複数形は matrices です．
[*3] 数には，実数と複素数があります．虚数単位は i または j です．

表 1.1　MATLAB が扱う数値アレーの表.

アレーの名前	内容	例
スカラー	実数，複素数 1×1 の行列	$a = 5.5,\ b = 2 * pi + 3j$
スカラー定数	円周率 虚数単位 数でない	pi i または j NaN
行ベクトル	$1 \times n$ の行列	$a = [1\ 2\ 3\ 4\ 5]$
列ベクトル	$m \times 1$ の行列	$a = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$
行列	$m \times n$ の行列	$A = \begin{bmatrix} 1 & 2 & 3 \\ 5 & 3 & 1 \end{bmatrix}$
空行列	0×0 の行列	$A = [\ ]$ zeros(0,4) 0×4 の空行列 zeros(3,0) 3×0 の空行列
定行列		eye(3) 単位行列 ones(3,4) 要素が 1 の行列 zeros(3,4) 要素が 0 の行列 rand(3,4) 要素が乱数の行列 magic(4) 4×4 の魔法陣 pascal(4) *Pascal* の三角形

言います．これは次のように表します．

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} = [a_{pq}] \tag{1.1}$$

a_{pq} を行列の要素 (element)，添字の $(p,\ q)$ は配列の位置を表す配列変数です[*4]．2 次元配列になっています．

$$A_{vec} = vec\,(A) = \begin{bmatrix} a_{11} \\ \vdots \\ a_{m1} \\ a_{12} \\ \vdots \\ a_{m2} \\ \vdots \\ a_{1n} \\ \vdots \\ a_{mn} \end{bmatrix} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \alpha_{r-1} \\ \alpha_r \end{bmatrix} \tag{1.2}$$

のように，行列 A を列を優先した 1 次元列ベクトルとして 1 次元アレー (array) と考えることもできます．これを行列 A の**ベクトル化** と呼んでおきましょう．1 次元配列にしたために，同じ要素：

$$a_{pq} \ = \alpha_r$$

の間の配列変数 $(p,\ q)$ と r の間には次の関係ができます．

$$r = (q-1)\,m + p \tag{1.3}$$

MATLAB では，a_{pq} や α_r を呼び出す場合，行列の名前に続いて「(　)」を付けて

$$a_{pq} \ \Rightarrow \ A\,(p,\ q)$$

$$\alpha_r \ \Rightarrow \ A\,(r)$$

のようにします．つまり，2 次元配列でも 1 次元配列でもどちらでも行列の要素を自由に呼び出せる仕組みになっています．この便利さが逆に混乱を起こすかも知れません．注意して下さい．特に配列変数の間の関係式 (1.3) を忘れないようにしておくと混乱が回避できます．では，すこし練習してみましょう．

[*4] 通常の教科書では a_{ij} のように，配列変数に i と j を使います．MATLAB ではこれらは虚数単位を表す予約定数ですから，ここでは p と q をもちいました．今後は適当に使いますが混乱はしないでしょう．

1.2.2　行列の中味の参照

いよいよこれから行列の「要素」,「行」,「列」を指定したり，それらを取り出して新しい行列をつくることを考えましょう.

まず，練習用の行列を 1 つ用意しましょう．魔法陣にしましょうか．つぎの命令を実行して行列 M を作って下さい.

≫ M=magic(5)

次の行列が得られます.

$$M = \begin{bmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{bmatrix}$$

この行列を

$$M = [m_{pq}], \quad p,\ q = 1,\ 2,\ \cdots,\ 5$$

とおくと

- 要素 m_{pq} をみるには，たとえば 2 行 3 列目の要素をみるには

  ≫ M(2, 3)

  とします．7 が答えになったはずです.

  ≫ M(12)

  ではどうでしょうか．同じになりましたか．式 (1.3) より $12 = (3-1)5+2$ です.

  ≫ M(12 : 15)

  とすると何が出てきますか．勿論この入力が許されるのならば

  ≫ M([12 13 14 15])

  もいいんでしょうね.

- p 行をみる．たとえば 2 行目をみるには，次のように入力します.

  ≫ M(2, :)

  最後の行をみるには

  ≫ M(end, :)

  などというのも使えます.

- たとえば 2 から 4 行目までをみるには，次のように入力します.

  ≫ M(2 : 4, :)

  では，こんなのはいかがでしょう.

≫ M(4 : 2, :)

答えは，Empty matrix: 0-by-5 と出ましたか．そう，0 行 5 列の空行列が出力されます．

- q 列をみる．たとえば 3 列目をみるには，次のように入力します．

  ≫ M(: , 3)

  最後の列をみるには

  ≫ M(: , end)

  が使えます．

- たとえば 3 列目と 4 列目をみるには

  ≫ M(: , 3 : 4)

  また，2 行目の 3 列目と 4 列目をみるには

  ≫ M(2 , 3 : 4)

  とします．もう行列のどんな要素でも取り出せるようになりましたか．

- 2 から 4 行目までの 3 列目と 4 列目を取り出し，行列 N に代入するには，

  ≫ N=M(2 : 4 , 3 : 4)

  とすればいいわけです．必要な部分行列を作るのに使いましょう．

1.3　行列を加工する

次に，行列を加工してみましょう．

- 行列を 1 列目から順番に前の列の最後の行にくっつけて 1 列行列（列ベクトル）にする．つまり，行列 (1.1) をベクトル化して列ベクトル (1.2) を作ります．

  ≫ M(:)

  結果がどのような行列になったかは

  ≫ size(ans)

  とすると分かります．答えが

  ans=

  　　25　　1

  となれば，最初の数字が行数，2 番目の数が列数です．この場合は 25×1 の行列，つまり列ベクトルになりました．

- たとえば，2 行目を取り除く．

  ≫ M=magic(5)

  ≫ M(2, :)=[ ]

- たとえば，2 行目と 3 行目を取り除く．

  ≫ M=magic(5)

  ≫ M(2 : 3, :)=[ ]

- たとえば，3 列目を取り除く．

  ≫ M=magic(5)

  ≫ M(:, 3)=[ ]

- たとえば，2 列目と 3 列目を取り除く．

  ≫ M=magic(5)

  ≫ M(:, 2 : 3)=[ ]

- たとえば，2 番目から 2 おきに 20 番目までを取り除き，残りの要素が列ベクトルとなる．

  ≫ M=magic(5)

  ≫ M(2 : 2 : 20)=[ ]

- ベクトル X の列を並び替えます．

  ≫ X=1 : 10

  ≫ X(10 : -1 : 1)

- 行の入れ替え．

  ≫ M=magic(5)

  ≫ M([1,5],:)=M([5,1],:)

- 行を入れ替えて取り出す．

  ≫ M=magic(5)

  ≫ M([2, 5,1],:)

- 行の置き換え．

  ≫ M=magic(5)

  ≫ N=[1 2 3 4 5]

  ≫ M(2, :)=N

- 行の置き換え．

  ≫ M=magic(5)

  ≫ N=[1 2 3 4 5]

  ≫ M(2, :)=N

表 1.2　MATLAB の行列操作のための関数

MATLAB の関数	関数名
diag	対角要素の列ベクトルを作る
rot90	行列を 90 度回転させる
tril	下三角要素を取り出す
triu	上三角要素を取り出す
fliplr	各列を左右ひっくり返す
flipud	各行を上下ひっくり返す

- 対応する行と列を小行列 N で置き換える.

  ≫ M=magic(5)
  ≫ N=[1 2; 3 4]
  ≫ M(2:3, 3:4)=N

- 対応する行と列を小行列 N で置き換える.

  ≫ M=magic(5)
  ≫ N=[1 2; 3 4]
  ≫ M([1, 3], [3, 4])=N

これですべてと言うわけではありませんが，ほぼ行列を使いこなせるのではないかと思います．特に，コンカット「[　]」とコロン「:」をうまく使えるようになって下さい．表 1.2 の関数も役に立つかも知れません．こんなのもあることを心に留めておいて下さい．

1.4　複数個の行列を作る

行列を複数個同じ名前で作れると便利です．配列を持った ℓ 個の行列

$$A_1,\ A_2,\ \cdots,\ A_\ell$$

を作るのにはどうしたらいいのでしょうか．MATLAB では，これは第 3 番目の配列にして簡単に作れます.

- 3×3 の行列を 4 個作ってみましょう.

  ≫ A=ones(3)

≫ A(: , : , 2)=2*ones(3)
≫ A(: , : , 3)=3*ones(3)
≫ A(: , : , 4)=4*ones(3)

結果がどのような行列になったかは

≫ size(A)

とすると分かります．答えが

ans=
3　　3　　4

となれば，最初の数字が行数，2 番目の数が列数，そして最後の数が A という名の行列が 4 個という意味です．この場合 MATLAB では「次元」(dimension) と言っています．添字付き行列は 3 次元行列というわけです．

- たとえば，2 番目の行列をみるには次のようにすればよいのです．
  ≫ A(: , : , 2)

- たとえば，1 番目の行列の 1 列目と 2 番目の行列の 1 列目をコンカット．
  ≫ [A(: , 1 , 1), A(: , 1 , 2)]

3 次元配列になっても，配列に対するこれまでの操作は同じです．必要に応じて要素や小行列を見ることができます．アレーの大きさ（サイズ）や次元を見る関数を表 1.3 にしておきます．

- whos は次のようになるでしょう
  ≫ whos('A')

Name	Size	Bytes	Class
A	$3 \times 3 \times 4$	228	double array

Grand total is 36 elements using 228 bytes

1.5　変数を消去する

これまでに色々定義したベクトルや行列をメモリーから消去する命令は

≫ clear

です．これできれいさっぱりメモリーの掃除ができます．勿論 wild card 「 * 」の使用が可能です．

≫ clear A∗

とすると，A から始まる変数は全部消去できます．

表 1.3　アレー情報を知るための関数

知りたい情報	関数	例
アレーのサイズ	size	size(A) ans= 2　3　4　(2 × 3 行列が 4 個)
アレーの次元	ndims	ndims(A) ans= 3
アレーの状態	whos	whos Name Size Bytes Class を表示

1.6　この章のまとめ

この章では，数値アレーの入力の方法，すなわち数値アレーの作り方を学び，作ったアレーを参照したり加工したりする方法を体験しました．

1.6.1　アレーの作り方と加工法

アレーの作り方のまとめ

1. MATLAB では「横長ベクトル」が基本となるベクトルである．一定間隔にならんだ数値ベクトルを作るには「：」を使うと便利である．
2. 「[　]」はベクトルや行列を作るのに使う演算子である．数値と数値の間は「　」(空白) か「,」で区切り，「;」を使って行と行を区別する．
3. 「'」はアレーを転置させる演算子である．
4. 文字や文字列を入力するには，入力したい文字列を「'」で囲む．たとえば，'Hello' のようにする．

アレーの参照や加工法まとめ

1. 行列は，2 次元配列として A(2, 3) のようにすると 2 行 3 列目の要素を参照できる．また，1 次元配列として A(12) のようにも参照できる．両者の引数間の関係式については，式 (1.3) をみてください．
2. 数値アレーを参照するときには，引数の指定に「：」が使える．
3. アレーの適当な要素を新しい要素で入れ換えるには，代入「=」を使えばよい．
4. アレーの情報を知るには，関数 size を使うとよい．

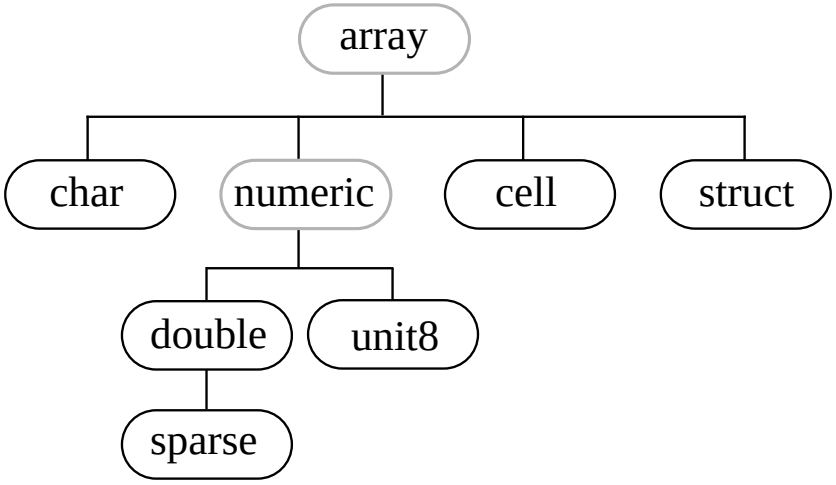


図 1.1　データ・タイプの樹状階層構造.

1.6.2　アレーって何だろう

MATLAB のマニュアルから Data Types の説明の部分を引用してみましょう.

There are six fundamental data types(classes) in MATLAB, each one a multidimensional array. The six classes are double, char, sparse, init8, cell, and struct. The two-dimensional versions of these arrays are called matrices and are where MATLAB gets its name.

You will probably spend most of your time working with only two of these data types: the double precision matrix (double) and the character array (char) or string. This is because all computations are done in double-precision numbers or strings.

The other data types are for specialized situations like image processing (unit8), sparse matrices (sparse), and large scale programming (cell and struct).

You can't create variables with the types "numeric" or "array". These *virtual* types serve only to group together types that share some common attributes.

The unit8 data type is for memory efficient storage only. You can apply basic operations such as subscripting and reshaping to these types of arrays but you can't perform any math with them. You must convert such arrays to double via the double function before doing any math operations.

You can overload methods for the build-in data types in exactly the same way you overload a method for an object. For example to define sort for a unit8 array, create a method(sort.m) and place it into @unit8 within a directory on your path.

The table that follows describes the data types in more detail.

セルとか構造体 (struct) があるようです. どちらも C 言語の構造体そのもののようです. 構造体がわざわざあるのは C 言語とのインターフェイスをよくするためでしょうか. ところでセルってどんな

表 1.4　MATLAB が扱うアレーの表.

Class	Example	Description
double	[1 2; 3 4]	Double precision numeric array (this is the most common MATLAB variable type).
char	'Helloe'	Character array (each character is 16 bits long). Also referred to as string.
sparse	speye(5)	Sparse double precision matrix (2-D only). The sparse matrix stores matrices with only few non-zero elements in a fraction of the space required for an equivalent full matrix. Sparse matrices invoke special methods especially tailored to solve sparse problems.
cell	{17 'hello' eye(3)}	Cell array. Elements of cell array contain other arrays. Cell arrays collect related data and information of a dissimilar size together.

struct	a.date=12;
	a.mat=magic(5);
	a.color='red';
		Structure array. Structure arrays have field names. The fields contain other arrays. Like cell arrays, structures collect related data and information together.

unit8	unit8(magic(3))	Unsigned 8 bit integer array. The unit8 array stores integers in the range from 0 to 255 in 1/8 the memory required for a double precision array. No mathematical operations are defined for unit8 arrays.
UserObject	inline('sin(x)')	User-defined type.

物なのでしょうか.「 [ ]」で数値アレーを定義したとき，要素は数値しか許されなかったことに気付いていましたか.「 [ ]」は入れ子 (nest) も許していません．セルは要素にベクトルや行列など何でも許し，しかも入れ子を可能にするアレーなのです.

セルは,「 [ ]」のカッコを「 { }」にすれば定義できます．たとえば，ベクトルとサイズの違う行列をセルにしてみましょう.

≫ C={[1, 2], [1 2 3;4 5 6], [7 8; 9 0; 2 1]}

とセル C を定義すると，3 つのセルが定義されて，そのサイズが表示されます.

≫ C{2}

等としてセルの中味を調べてみましょう．これですべてのカッコが出そろいました.

1.7　演習問題

1. 次の式を入力し結果がどうなるか，またなぜそうなるのか考えてみよう．

   ≫ [[[]]]　　≫ [],[],[]　　≫ [[],[]]　　≫ [[]],[]

   ≫ {{{}}}　　≫ {},{},{}　　≫ {{},{}}　　≫ {{}},{}

2. eye(4) を使って，4×4 の単位行列を作り，これを加工して次の行列を作ってください．

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

   【注意】行とか列を回して入れ換える関数，たとえば shiftrotate(A) などというのが，あればいいのですが残念です．そのうちにもう少し勉強して，自分でそんな関数を作りましょう．また，楽しみが 1 つ増えました．

3. 次に示す 4×4 のべき零行列の標準型を作ってください．

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

4. MATLAB Help menu から Examples and Demos を選択し，Demo Window から Matrix manipulation を選択して，行列の操作のデモを試してみよう．【注意】よくあることですが，このように Demo を試してみる場合，画面は英語で書いてあるし，マウスをクリックすると計算機が勝手に計算してくれているしで，ついついボーとして，ただただ眺めているばかりで，何をやっているのやらさっぱり分からなくなってしまい勝ちです．いけませんねえ．そんなことでは．英語の勉強になるし，行列の勉強はできるし，おまけに Windows 95 の使い方まで修得できるのですから，1 石 3 鳥 という気持ちにならなければいけません．そうそう，別の Demo もついでにやってしまう位の好奇心もほしいですねえ．

5. MATLAB Help Window を開き，matlab\elmat をダブルクリックで開いて，行列の基本操作の色々な関数を実行してみよう．特に，最後の欄をみると種々の特殊な行列を定義する関数がある．次の関数を実行し，適当に行や列を操作して遊んでみよう．

   - compan，hadamard，hilb，vander
   - magic，pascal

   【注意】ここに出てくる変な行列ってどんなところに使うのかしら？ いい質問です．さっそく行列の本で調べてみましょう．こんな努力こそがあなたの将来に光を投げかけるのです．そうそう Hadamard （アダマールと読む）行列は 2^n のサイズの行列しか定義されていません．hadamard(2), hadamard(4), hadamard(8) などと入力してみましょう．えっ，これらの行列には規則のあることが分かったって？ だからどうなんでしょうねえ．

第 2 章

行列の演算

いよいよ行列演算の練習です．MATLAB の心臓とも言える大事な計算です．

2.1　行列の演算あれこれ

ベクトルや行列には仲間どおしの計算（演算）ができます．おおざっぱに言って，次のような演算が可能です．

1. スカラーすなわち実数と複素数の場合：足し算とその逆算の引き算，そして掛け算とその逆算の割り算．このような 2 種類の演算が自由にできる集合は「体」と呼ばれています．
2. ベクトルの場合：ベクトルどおしの足し算とその逆算の引き算，そしてスカラーとの積，すなわちスカラー倍．このような集合は「ベクトル空間」と呼ばれています．ベクトルどおしの積はありません．ただし，内積やベクトル積といった特殊な積を考えることがあります．
3. 行列の場合：行列どおしの足し算とその逆算の引き算，勿論スカラーとの積；更に行列どおしの積．このような集合は「代数」と呼ばれています．行列の積には色々な積があります．

このように，色々な演算がスカラーはスカラー，ベクトルはベクトル，そして行列は行列の間で定義されていることに加えて，それら相互の間にも掛け算などの演算が可能な場合がでてきます．

したがって，「ややこしいかも知れない」ことは覚悟しておきましょう．特に「積，すなわち掛け算」に多様性がでてきます．それだけおもしろいということでしょうか．演算が次の 3 つの場合におおまかに分けられることを知っておくと便利かも知れません．

- スカラー倍はすべての要素（配列）にスカラーが掛け算される．MATLAB では通常の演算記号を使う．
- 配列が同じものどおしの演算を行う．MATLAB では通常の演算記号の前に「.」を付けた演算記号を使う．
- 配列間で一定のルールに従って演算を行う．たとえば行列の積など．MATLAB では通常の演算記号を使う．

幾つかの演算の定義を表 2.1 にまとめておきます．スカラーをギリシャ語のアルファベットで，ベクトルを小文字のローマ字で，また行列を大文字で表してあります．すなわち，ベクトルは $a = [a_r]$, $b = [b_r]$，行列は $A = [a_{pq}]$, $B = [b_{pq}]$ などと表してあります．

「.」（ピリオド）付きの演算とそうでない演算は，「.」を見落とし勝ちですから注意して下さい．もちろん，その違いを分かることが先決です．

2.2　アレー演算

【例 2.1】

次の計算をしてみよう．

1. ≫ t=0:10
   ≫ t*5
   ≫ t.^2
   ≫ t /2
   ≫ t’
2. ≫ A=[1 2 3; 4 5 6; 7 8 9]
   ≫ A*2
   ≫ A.^2
   ≫ A/2
   ≫ A’
3. ≫ A=[1 2 3; 4 5 6; 7 8 9]
   ≫ B=2*ones(3)
   ≫ A.*B
   ≫ A*B
   ≫ A./B
   ≫ 2*A-3*B
4. ≫ A=[1 1;1 -1]
   ≫ kron(A, A)

【解説】 結果と表 2.1 を見比べて計算が合っていることを確かめて下さい．

1.「.^」は要素ごとのべき乗です．したがって，横長ベクトル

≫ x=-5 : 0.1 : 5

の各要素（成分）を

$$f(x) = x^3 - x^2 - 5x - 2$$

のような関数の値にするには

≫ $y = x.\hat{}3 - x.\hat{}2 - 5 * x - 2$

とすればいいことになります．結果は

表 2.1　色々な演算の表.

演算の名前	演算の定義	MATLAB の演算
スカラー倍	$\alpha a = [\alpha a_r]$	$\alpha * a \ or \ a * \alpha$
	$\frac{1}{\alpha} a = \left[\frac{1}{\alpha} a_r\right]$	$a/\alpha \ or \ \alpha \backslash a$
	$\alpha A = [\alpha a_{pq}]$	$\alpha * A \ or \ A * \alpha$
	$\frac{1}{\alpha} A = \left[\frac{1}{\alpha} a_{pq}\right]$	$A/\alpha \ or \ \alpha \backslash A$
	$[(a_{pq})^n]$	$A.\hat{}n$
ベクトルの加法	$a + b = [a_r + b_r]$	$a + b$
ベクトルの減法	$a - b = [a_r - b_r]$	$a - b$
行列の加法	$A + B = [a_{pq} + b_{pq}]$	$A + B$
行列の減法	$A - B = [a_{pq} - b_{pq}]$	$A - B$
行列の要素どおしの乗法	$A \odot B = [a_{pq} b_{pq}]$	$A. * B$
行列の要素どおしの除法	$A \oslash B = [a_{pq}/b_{pq}]$	$A./B$
行列の乗法	$AB = \left[\sum_k a_{pk} b_{kq}\right]$	$A * B$
行列の Kronecker 積	$A \otimes B = [a_{pq} B]$	kron(A, B)
逆行列の積	$AX = B \Rightarrow X = A^{-1}B$	$A\backslash B$
(連立方程式の解)	$XA = B \Rightarrow X = BA^{-1}$	B/A
行列のべき	A^n	$A\hat{}n$
転置行列	A^t	$A.'$
共役転置行列	A^t	A'

≫ plot(x, y)

≫ grid

とすれば，見ることができます．

4. Kronecker 積と呼ばれる行列の積です．

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & 1\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\ 1\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & -1\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

となります．

【練習 2.1】

次の計算をしてください．

1. $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix} /2$
2. $\begin{bmatrix} 1 & 2 & 3 \\ 6 & 5 & 4 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 1 & 6 \\ 2 & 5 \\ 3 & 4 \end{bmatrix}$
3. $\begin{bmatrix} 1 & 5 & 6 \\ 9 & 2 & 4 \\ 7 & 8 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$ を解く．
4. $\begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$
5. $\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$

表 2.2　MATLAB の初等関数

	MATLAB の関数	関数名
複素数	abs	絶対値
	angle	位相角
	real	実数部
	imag	虚数部
	conj	複素共役
三角関数	sin	正弦
	cos	余弦
	tan	正接
	atan	逆正接
	atan2	逆正接（4 象限）
指数・対数関数	exp	指数関数
	log	自然対数
	log10	常用対数
その他	rem	割り算の余り
	sign	符号関数
	sqrt	平方根

2.3　関数の計算

2.3.1　初等関数

MATLAB には多くの関数がすでに用意されています．初等関数でよく使いそうなものを表 2.2 にまとめておきます．

これらの関数は，引数にスカラー，ベクトルや行列が許されます．これは非常に便利です．関数はア

レーの要素毎に作用します．したがって，たとえば行列

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

の余弦 cos をとると

$$\cos(A) = \begin{bmatrix} \cos(a) & \cos(b) \\ \cos(c) & \cos(d) \end{bmatrix}$$

となって，配列の要素一つひとつの cos が取られます．

- 関数はアレーの配列要素一つひとつに作用する．

と覚えて下さい[*1] ．

≫ t=0 : pi/30 : 2*pi

≫ plot(t, sin(t))

などとすると簡単に正弦関数が計算されてグラフに描けます．

2.3.2　行列に作用する関数

行列を入れると，特定の計算をして返してくれる関数達があります．ここでは，それらの内のよく使われる関数を表 2.3 にまとめておきます．次の関数を入力して，結果を確かめてみましょう．

≫ M=magic(5)

≫ sum(M)

≫ trace(M)

≫ sum(M')

≫ sum(rot90(M))

これだけ揃っていれば，線形代数で出てきたベクトル空間の諸性質を調べたり，線形定係数常微分方程式を解いたりいろんなことができそうです．たとえば

$$rank\left(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}\right) = 2$$

[*1] アレー (array) は MATLAB の最も基本的なデータであったことを思い出してください．アレーはまたリスト (list) とも呼ばれているデータ構造の一つです．アレーに作用できる関数のことを arrayable とでも呼びたいところです．すると上の例では「cos はアレーアブルである」と言うことができます．

表 2.3　MATLAB の行列関数

	MATLAB の関数	関数名
行列解析	det	行列式
	sum	各列の和を作り行ベクトル表示
	trace	跡：対角要素の和
	norm	行列やベクトルのノルム
	rank	ランク：一次独立な行や列の数
	null	Kernel：写像の零空間
固有値	eig	固有値と固有ベクトル
	poly	特性多項式
指数・対数関数	expm	行列の指数関数
	logm	行列の自然対数
	sqrtm	行列の平方根

が容易に確かめられますから，方程式

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

の解はとても制限されたものになるでしょう．

2.4　演習問題

1. 次の連立方程式を解きなさい．解けない場合はその理由を考えなさい．
   1. $2x + 13y - 3z = -7,\ x + y = 1,\ x + 7z = 22$
   2. $x - 2y - 12z = 12,\ 2x + 2y + 2z = 4,\ 2x + 3y + 4z = 3$
   3. $x + y + z = 5,\ x - y - z = 4,\ 2x + 6y + 6z = 12$
   4. $x + 3y + z = 4,\ -x - y + z = -1,\ 2x + 4y = 0$
   5. $x + 2y + z - 4w + 1 = 0,\ x + 2y - z + 2w - 1 = 0,$
      $2x + 4y + z - 5w + 1 = 0,\ x + 2y + 3z - 10w + 2 = 0$

2. 今も昔も，それは変わらないことなのですが，とある南の国に鶴と亀と蟻が住んでいたそうな．ある日，亀が主催した川辺のパーティに，鶴亀蟻合わせて 78 匹も集まったそうな．鶴と亀の足の数は合わせて 120 本，蟻と鶴の足の合計は 264 本であったという．鶴亀蟻それぞれ何匹いるのでしょう．あ，そうそうどんなパティーになったかは，残念ながらそれは昔のことなので誰も知らないそうです．
3. 次の行列の積を計算しなさい．
   1. $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 7 & 8 \\ 9 & 0 \\ 1 & 2 \end{bmatrix}$
   2. $\begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 & 4 \\ -1 & 3 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 3 \end{bmatrix}$
   3. $\begin{bmatrix} 1 & -1 & 1 \\ -1 & 0 & 2 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & -1 \\ -1 & 1 & 2 \\ 2 & 0 & -2 \end{bmatrix}$
   4. $\begin{bmatrix} 7 & 1 \\ -1 & 0 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} 5 \\ -4 \end{bmatrix}$
   5. $\begin{bmatrix} 7 & 1 \\ -1 & 0 \\ 2 & 3 \end{bmatrix} \otimes \begin{bmatrix} 5 \\ -4 \end{bmatrix}$
   6. $\begin{bmatrix} 1 & -1 & 1 \\ -1 & 0 & 2 \\ -1 & 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 & -1 \\ -1 & 1 & 2 \\ 2 & 0 & -2 \end{bmatrix}$
   7. $A = \begin{bmatrix} 1 & 2 & 1 \\ -1 & 4 & 1 \\ 2 & -4 & 0 \end{bmatrix}$, $P = \begin{bmatrix} 1 & -2 & -1 \\ 1 & 1 & 1 \\ -1 & 1 & 0 \end{bmatrix} \Rightarrow find\ PAP^{-1}$

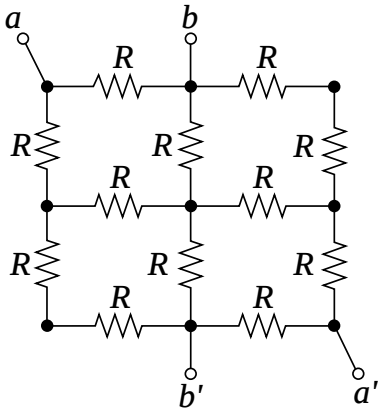


図 2.1　抵抗回路

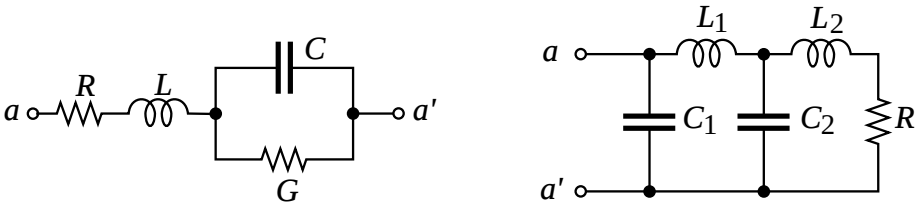


図 2.2　合成インピーダンス

4. A=[1: 5] とします．これを用いて次の行列を求めてください．

$$B = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 6 & 8 & 10 \\ 3 & 6 & 9 & 12 & 15 \\ 4 & 8 & 12 & 16 & 20 \\ 5 & 10 & 15 & 20 & 25 \end{bmatrix}$$

5. 図 2.1 に示した抵抗回路の端子 aa', bb', ab, ab' から見た合成抵抗を求めなさい．
6. 図 2.2(a), (b) に示した回路の端子 ab から見た合成インピーダンスを求めなさい．ただし，すべての素子定数は 1 であるとします．
7. 行列の rank と null についてその幾何学的意味を調べ，前節最後に述べた方程式の解を求めなさい．
8. 図 2.3 に示したアミダクジは横棒を一つ通過する毎に abcd の置換を受ける．これを行列で表し，全体の置換をこれらの積と考え，この変換行列を求めよ．

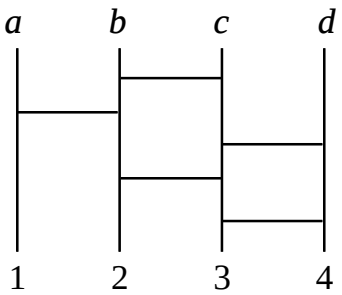


図 2.3　4 人用のアミダクジ

第 3 章

グラッフィクスに親しむ

Visualization は計算を楽しくしてくれます．計算結果をグラフに描いてみましょう．

3.1　2 次元グラッフィクス

ここでは，2 次元グラッフィクスを描くことを練習します．

3.1.1　plot 命令を使う

【例 3.1】

正弦波を描いてみよう．

1. ≫ t=0 : pi/30 : 2*pi ;
   ≫ plot(t, sin(t))
2. ≫ t=0 : pi/30 : 2*pi ;
   ≫ plot(cos(t), sin(t))
3. ≫ t=0 : pi/30 : 2*pi ;
   ≫ plot(t, cos(t), t, sin(t))
4. ≫ t=linspace(0 , 2*pi, 30);
   ≫ plot(t, cos(t),'r' , t, sin(t), 'b*')

【解説】 plot 命令は普通のグラフを描くのに使います．

plot(横軸データ, 縦軸データ)

が基本です．横軸データ, 縦軸データの次に曲線の色を指定できます．

plot(横軸データ, 縦軸データ, 'r')

のようにすれば良いのです．$'r'$ は red すなわち赤のことです．$'r*'$ などとすると曲線が $*$ で描かれます．表 3.1 に色と曲線のシンボル指定をあげておきました．

linspace(x_{min}, x_{max}, データの個数)

とすると，区間 $[x_{min}, x_{max}]$ をデータの個数で等間隔に割った行ベクトルができます．

表 3.1　色と曲線のシンボル表

シンボル	色	シンボル	ラインタイプ
y	黄	.	点
m	赤紫	o	円
c	シアン	x	x 印
r	赤	+	+ 印
g	緑	*	星印
b	青	-	実線
w	白	:	点線
k	黒	-.	鎖線

すでに描いてあるグラフの上に「重ねがき」するにはどうすればいいのでしょうか.

≫ hold on

としておいて，描きたいグラフをかきます.

≫ plot(t, sin(3*t))

そのあと

≫ hold off

とするといいでしょう.

【練習 3.1】

次の関数のグラフを描いて下さい．横軸の範囲は指定したとおりにして下さい.

1. $y = \exp(-0.3t)\sin(3t),\ 0 < t < 10$
2. $y = 2\sin(t) + \cos(4t),\ 0 < t < 2\pi$
3. $y = 2\sin(t) + \cos(4t)\ and\ y = -2\cos(t) + \sin(3t),\ 0 < t < 2\pi$
4. $x = \sin(t)\ and\ y = \cos(t) + \sin(3t),\ 0 < t < 2\pi$

なお，グラフの「枠」を正方形にするには

≫ axis square

とします．通常は axis normal になっています．また，

≫ title 'Exercise 1'

≫ xlabel 'time'

≫ ylabel 'voltage'

表 3.2　2 次元グラフィックス命令

命令	意味
plot(x,y)	曲線のプロット
ploar(t,r)	極座標プロット
semilogx(x,y)	片対数プロット
loglog(x,y)	両対数プロット
stem(x,y)	棒状プロット
grid on/off	グリッドの挿入・抹消
clf	グラフを消す

などと気取ってみるのもおもしろいでしょう．

描いたグラフの部分グラフをえがくには

axis($[x_{min}\ \ x_{max}\ \ y_{min}\ \ y_{max}]$)

とするといいでしょう．最後に，同時に 3 本のグラフを描くには

plot(t, y_1, t, y_2, t, y_3)

のようにデータを次々と並列して指定します．勿論それぞれの線に色やタイプを指定できます．やってみましょう．そうそう，忘れるところでした

≫ grid on

とやると「網目」がかかります．grid off で網目は除かれます．

最後に，描いたグラフを消去するには

≫ clf

とします．グラフィックスの窓はまっさらになり，次のグラフが描かれるのを待っています．

3.1.2　polar 命令を使う

この命令は極座標表示をします．

【例 3.2】

次のグラフを描いてみよう．

1. ≫ t=0 : pi/50 : 2*pi ;
   ≫ polar(t, exp(-0.1*t))
2. ≫ t=0 : pi/50 : 2*pi ;
   ≫ polar(cos(t), sin(t))
3. ≫ t=0 : pi/50 : 2*pi ;
   ≫ polar(cos(t), 2*sin(t)+cos(4*t))

【解説】 polar 命令は極座標表示のグラフを描くのに使います．

polar(角度，動径)

とすればよい．

3.1.3　semilogx 命令を使う

この命令は片対数グラフを表示します．

【例 3.3】

次の関数は，2 次系の伝達関数です．

$$G(s) = \frac{1}{s^2 + 2\zeta s + 1}$$

ゲインは次式となります．

$$y = 20\log|G| = -10\log\left\{\left(1 - x^2\right)^2 + (2\zeta x)^2\right\}$$

このグラフを描いてみよう．

≫ x=logspace(-1 , 1, 100);
≫ zeta=.1
≫ y=-10*log10((1.0-x.^2).^2+(2.0*zeta*x).^2);
≫ semilogx(x, y, 'r')
≫ grid on

【解説】 横軸に対数目盛の間隔を作るには

logspace(横軸最初の値の 10 のべき乗，最後の値の 10 のべき乗，データの個数)

とすればいいのです．したがって，上の例では，区間 $[10^{-1},\ 10^{1}]$ が 100 分割されます．あとは関数を作って semilogx に渡してやればゲインが描けます．zeta の値を色々変えてゲイン曲線を描いてみましょう．

線の太さや色を変えたいときには，グラフを描くときに，まずグラフ命令全体を適当な変数に蓄えて，その属性を変えるといいでしょう．たとえば，

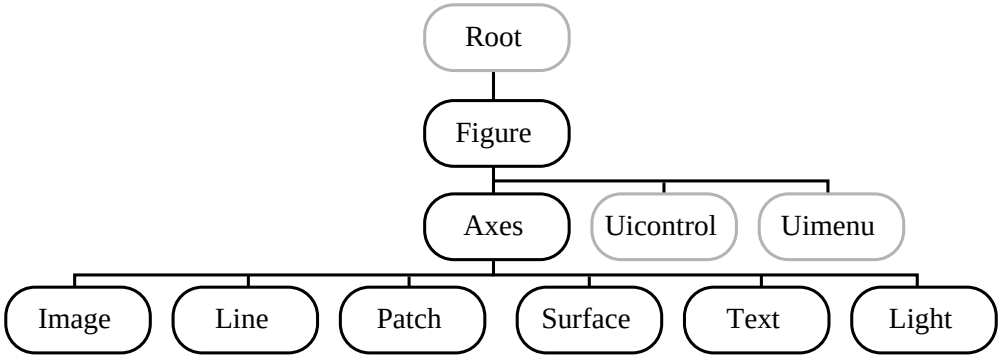


図 3.1　グラフィック・オブジェクトの樹状階層構造.

≫ hndl=semilogx(x, y, 'r')
≫ set(hndl, 'Color', 'blue')
≫ set(hndl, 'LineWidth', 4)
といった具合にするといいでしょう.

【練習 3.2】

例 3.3 の伝達関数の位相線図を描きなさい．位相は次の関数で与えられます.

$$\varphi = -\tan^{-1}\frac{2\zeta x}{1-x^2}$$

【解説】注意することは tan^{-1} の扱いです．atan2 の関数を使うといいでしょう.
atan2(y 軸の値，x 軸の値)
とします．この例では
≫ y=-180*atan2(2.0*zeta*x, 1.0-x.^2)/pi;
となります.

3.1.4　グラフィックス・オブジェクトでグラフを操作する

グラフィックスに関しては，簡単に描ければ描けるほど，自分好みの図を描きたくなるものです．そこで色々とグラフィックスの命令を使うことになるのですが，命令の種類が多いので混乱してしまいます.

MATLAB は Version 5 からオブジェクト指向プログラミング (objet oriented programming: OOP) 可能な言語となりました．グラフの操作 (handling) もその観点から扱いやすくなっています.

グラフィック・オブジェクトは，図 3.1 に示したような親子関係を持っています．これをグラフィック・オブジェクトの樹状階層構造とでもいうのでしょうか．黒枠のオブジェクトが我々が直接グラフを操作するときに関係してきます.

では，グラフィック・ウインドを開いていたらそれらを全部閉じて，次の命令でためしてみましょう.

1. グラフィック・ウインドを 3 個開く.

≫ figure
≫ figure
≫ figure

3 個まっさらのウインドが現れ，それぞれ出来た順序にタイトル・バーに番号が付けられます．最後のウインドがアクティブ・ウインドです．今の時点でグラフを描くと Figure No. 3 のウインドに描かれます．

2. アクティブ・ウインドを切り換えるには，
   ≫ figure(1)
   のように，ウインド番号を指定します．これで，複数のウインドを開いて，好みのウインドに作図することが可能となります．
3. オブジェクトは，一般に「子」は「親」の性質を合わせて持っています．これを継承と言っています．今開いているウインドをすべて閉じ，次の命令を実行して下さい．
   ≫ axis([-1,1,-2,2])
   Figure No. 1 のウインドが開かれて（つまり，figure 命令が実行されて）軸が描かれました．
4. 図 3.1 の一番根っこの親は，MATLAB のウインドで，これは MATLAB を動かしたときに自動的に作られます．我々は操作できません．
5. Uicontrol(user interface control) と Uimenu(user interface menu) はウインドにボタンを付けたり，メニューを付けたりするときに使うのでしょう．Demo プログラムでよく見かけます．やってみたい人は help の助けを借りてどうぞ試してみて下さい．

3.1.5　一つの窓にグラフを複数描く

前々節で描いた Bode 線図などは，一つのウインドに上下並べて，ゲイン線図と位相線図を描きたくなります．これをやってみましょう．

subplot(m,n,p)

という命令を使います．これは，現在のウインドに $m \times n$ 個の作図領域を作って，その p 番目の領域に以下の作図命令を実行させる命令です．では，早速描きましょう．

```
≫ x=logspace(-1 , 1, 100);
≫ y=-10*log10((1.0-x.^2).^2+(0.2*x).^2);
≫ z=-180*atan2(0.2*x, 1.0-x.^2)/pi;
≫ subplot(2,1,1);
≫ semilogx(x, y, 'r');grid on
```

上の作図領域にゲイン線図が描けましたか？ 次に下の領域に位相線図を描きましょう．

```
≫ subplot(2,1,2);
≫ semilogx(x, z, 'b');grid on
```

おやおや，y 軸が -200 度まであるなんていやですねえ．では，こうしましょう．

```
≫ axis([0.1,10,-180,0])
```

まったく，こんどは目盛り (tick) が気に入りません．これでどうでしょう．

≫ set(gca, 'ytick', [-180:30:0])

やっと，気に入ってもらえましたか．

え？　gca って何かって？ これはプロパティー graphics current axes のことで軸についての情報がつまっている構造体のようなものです．そのほか

- gcf – Root Current Figure
- gca – Figure's Current Axes
- gco – Figure's Current Object

などがあります．

≫ get(gca)

≫ get(gcf)

などで，現在のプロパティーをみることができます．勿論，set 命令はこれらを設定する命令です．詳しくは help でどうぞ．

3.2　3 次元グラッフィクス

3 次元グラッフィクスは魅惑に満ちています．こんなおもしろいテーマに捕まったら大変です．さらっと通り過ぎることにしましょう．それに 3D と言っても 2D 画面にしか描けないのですから，2D の応用じゃないでしょうか．

3.2.1　plot3 命令を使う

【例 3.4】

トーラス (torus) を描いてみよう．トーラスの方程式は

$$x = (r_1 + r_2\cos\varphi)\cos\theta,\ y = (r_1 + r_2\cos\varphi)\sin\theta,\ z = r_2\sin\varphi$$

で与えられます．

【解説】

≫ t=0:pi/20:60*pi;

≫ r=2+0.5*cos(t/30);

≫ x=r.*cos(t)

≫ y=r.*sin(t)

≫ z=0.5*sin(t/30);

≫ plot3(x,y,z,'r'):axis([-2,2,-2,2,-1,1]):grid on

plot3 命令は 3D グラフィックスの基本命令で 3 次元にパラメータ付けされた曲線を描くのに使います．

plot(x 軸データ, y 軸データ, z 軸データ, 'r')

のように使います．勿論，$'r'$ は red すなわち赤のことです．

3.2.2　曲面を描く

【例 3.5】

次の関数のグラフを描いて下さい．横軸と縦軸の範囲は $-\pi \leq x,\ y \leq \pi$ とします．

$$z = \sin x \cos y$$

関数を定義してワイヤーフレームで曲面を描きます．

【解説】

≫ t=-pi:pi/20:pi;

≫ x=t; y=t;

≫ z=sin(x')*cos(y);

≫ mesh(x,y,z)

次に，網目を塗り込んだ曲面を描きます．

≫ surf(x,y,z)

これらは，

≫ mesh(z)

や

≫ surf(z)

としても描けます．何処が違うって？ よく目盛りをみてください．引数が一つの場合は

mesh(i, j, z_{ij})

と見なして，作図されています．

3.2.3　パラメータ付けされた曲面を描く

【例 3.6】

球面を描いて網目に適当な色を塗ってみましょう

```
≫ colormap cool;
≫ k=4;n=2^k-1;
≫ theta=pi*[-n:2:n]/n;
≫ phi=(pi/2)*[-n:2:n]'/n;
≫ x=cos(phi)*cos(theta);
≫ y=cos(phi)*sin(theta);
≫ z=sin(phi)*ones(size(theta));
≫ c=hadamard(2^k);
≫ surf(x,y,z,c);
≫ axis square
```

図の横にカラーバーを付けると

```
≫ colorbar
```

【解説】mesh や surf の命令の引数が行列になれば，面上の頂点の位置 $(x_{ij},\ y_{ij},\ z_{ij})$ がカラー (c_{ij}) で着色されることになります．

ここで，カラーマップの指定を表 3.3 に示しておきます．

では，遊んでみましょう．

```
≫ shading interp
```

えっ！ なんで？

```
≫ colormap hsv
```

みごとな変身ですね．更に，

```
≫ light
≫ lighting gouraud [*1]
```

勿論，こんなことなら

```
≫ help shading
```

や

```
≫ help lighting
```

で色々と関数を調べて，試したくなるでしょう．

それにしても，平面の長方形領域をメッシュに分割して，これを適当な曲面に張り付ける．このプログラムには直接関係しない数学的な部分が，以外とやっかいなのではないでしょうか．

たとえば，例の球面に張り付けた場合を参考にしてトーラス面に張り付けて，同じように 3D グラ

[*1] gouraud は「グロー」と読みます．

表 3.3　カラーマップの指定表

関数	意味
hsv	hue-saturation-value(HSV)
jet	a variant of HSV
hot	black-yellow-white
cool	shades of cyan and magenda
grey	linear gray-scale
bone	gray-scale with tinge of blue
copper	linear copper-tone
flag	alternating red, white, blue, and black
pink	pastel shades of pink
white	all white color map
prism	prism color map
summer	shades of green and yellow
autum	shades of red and yellow
winter	shades of blue and green
spring	shades of magenda and yellow

フィックスを楽しんでみてください.

花びらをつくるとか. 菊の花を描くとか. 色々描きたい対象を作って試みてみてはいかがでしょう.

3.3　演習問題

1. polar プロットにより次の関数を描け.

$$r = |a \sin n\theta + b \sin 3n\theta + c \sin 5n\theta|$$

ここに, たとえば

$$a = 7,\ b = 1,\ c = 2,\ n = 1.5$$

と選んで試してください．その後これらのパラメータを変化させて図の移りかわる様子を観察しなさい．この関数は

戸川隼人著：花の CG, サイエンス社, 昭和 63 年

の p. 16 に示されています．この本は，大変魅力のある本です．他の例についても実験されることを希望します．

2. 例題 3.3 に習って，制御の教科書から適当な伝達関数を選び，Bode 線図や Nyquist 線図を描いてください．
3. 例題 3.4 にあるトーラスの方程式を使って，例題 3.6 を参考にして曲面を描いてください．たとえば，次の一連の命令で描けます．

```
%Torus by surf
colormap cool;
k=5;n=2^k-1;
theta=(2*pi/3)*[-n:2:n]/n;
phi=(pi)*[-n:2:n]'/n;
r1=1;r2=0.3;
a=ones(size(phi));

x=(r1*a+r2*cos(phi))*cos(theta);
y=(r1*a+r2*cos(phi))*sin(theta);
z=r2*sin(phi)*ones(size(theta));
c=hadamard(2^k);
surf(x,y,z,c);
axis([-1,1,-1,1,-1,1]);
axis square
colorbar
figure(1)
```

4. 上の問題を一般化して，平面の長方形領域を，3 次元に埋め込まれた 2 次元曲面に写像する手法を考えなさい．
5. 上の問題から，トーラスを部分的に描くにはどうすればよいかが簡単に分かります．経度や緯度で輪切りにしたトーラスを描いてみましょう．
6. 結び目のあるトーラスを描いてみよう．
7. 問題 1 で紹介した戸川先生の本にある「アサガオの花」を描いてみよう．

第 II 部

MATLAB のプログラミング

第 4 章

プログラムしてみよう

ちょっと慣れてくると，自分好みのプログラムを作ってみたくなるものです．簡単な命令を組み合わせるとプログラムができあがります．知っておいて損にはなりません．作ってみましょう．

4.1 MATLAB のプログラム

MATLAB には，2 種類のプログラムの扱いがあります．1 つは MATLAB 本体から「対話形式」で打ち込んでいた命令を単にファイル形式にまとめて作るプログラムです．これは「スクリプト (script)」と呼ばれています．

他の 1 つは「関数 (function)」です．こちらは，関数名や引数を持っています．これまでにも MATLAB 本体で既に作ってくれている多くの関数を使ってきました．関数は，たとえば sin(theta) のような形で使います．

スクリプトにしろ，関数にしろ MATLAB では，1 つのファイルとして保存します．このときファイルの名前の後ろの「拡張子」はいつも「．m」とします．このことから，MATLAB プログラムのファイルは，総称して「M - file 」と呼ばれています．したがって，作ったプログラムを保存するときは，次のルールにしたがって保存すると無難です．

- ファイル名の「拡張子」は「．m」とする．たとえば

  myprog.m

  とする．これは絶対に従わなければなりません．
- ファイルの名前は
  - スクリプトの場合はどんな名前を付けても良い．
  - 関数の場合は，関数名と同じ名前にする．

さて，作ったプログラムを MATLAB 本体から使うときには，ファイル名で呼び出します．

≫ myprog

といった具合です．

それから，実際にプログラムのファイルをエディターで作らなければなりません．これも簡単です．MATLAB ウインド左上の白いファイルの絵のアイコンをダブル・クリックして下さい．新しいファイ

ルが開かれます．

プログラムが終わったら，エディターのメニューの「File」から「Save As」選んで上で述べたルールに従ってファイル名を入力し保存すれば出来上がりです．

それでは，簡単なプログラムを作ってみましょう．

【例 4.1】

整数 1 から 100 までの和を計算するプログラムを作ってみよう．

【解説】

1. とりあえず，スクリプトで作ってみましょう．

≫ sum([1:100])

とすると答えは求められます．これをプログラムにすればよいのです．なお，% のついた行はコメント行です．また，普通の行でも % 以降は無視されます．

```
% This is my first program : the addition from 1 to 100
sum([1:100])
```

できたファイルを，たとえば「wa100.m」という名前で保存します．MATLAB から使うときは

≫ wa100

とすればよいのです．どうですか．正解が出ましたか．

ちょっと，これではプログラムなの？ と言いたくなりますが，これは MATLAB 式プログラムです．C に馴染んでいるあなたはたとえば次のような「プログラムらしい」プログラムを書くことでしょう．

2.　次もスクリプトプログラムです．

```
% This is my second program : the addition from 1 to 100
wa=0;
for i=1:100
  wa=wa+i;
end
wa
```

このプログラムは繰り返し命令「for　end」を使っています．「wafor.m」という名前で保存し，実行してみましょう．繰り返し命令については，次の節で説明します．まあ，大体のところは分かりますね．最後の「wa」を付けておいたのは，答えを実行後に表示するためです．次は関数としてプログラムしてみましょう．

3.　せっかくですから，1 から n までの和を計算する関数「wa3」という名前の関数を作りましょう．

```
function a=wa3(n)
% This is my third program : the addition from 1 to n
a=0;
for i=1:n
  a=a+i;
end
%
% Of course you can write shortly as follows:
% a=sum([1:n])
```

```
%
```

関数のプログラムは最初の行に

```
function a=wa3(n)
```

のように「関数」の宣言と「関数名（引数)」を書く．関数名の前の等号より前は「出力の帰り値」を書きます．この場合だと和 a を返しますから a = wa3(n) のように書きました．その代わりにプログラムの最後に「出力の帰り値」を書く必要はありません．

では最後に，ちょっと変わったプログラムをしてみましょう．

4.　このようなプログラムは再帰型プログラム (recursive programming) と呼ばれています．勿論「recwa.m」として保存しましょう．

```
function a=recwa(n)
% This is my first recursive program : the addition from 1 to n
if n==1
  a=1;
  return;
else
  a=n+recwa(n-1);
end
```

このプログラムでは，

- n=1 の場合には a=1 を返す．
- n=n の場合には，a=n+recwa(n-1) として，自分自身の n を減らした関数を呼ぶ．

となっています．関数型プログラミングでは，このような再帰型関数を書くことが普通です．それでは，次節で制御命令にどんなものがあるのか具体的に見ることにしましょう．

4.2　代入と繰り返し

計算機のプログラムの基本は「読み・書き・そろばん」と言われています．「読み」はデータを読んでくること「入力」です．「書き」は，一連の処理結果を「出力」することです．「そろばん」は，計算やデータの「処理」をすることです．

ところで，この計算のところで最も重要な命令は「代入」です．これは

```
a=b+c;
```

のように「＝」で，左辺の計算結果を右辺の変数に代入することです．あまりに単純な処理のために，軽くみられてしまい勝ちですが，これなくしてプログラムは作れません．

次に，重要な命令は「繰り返し」計算です．これには，次の 2 つの命令が基本となります．

- あらかじめ定められた回数だけ，繰り返す．「for　end」文がこれに当たります．
- 何回繰り返すかは未知で，ある条件が満足された時に，繰り返しを終了する．いわゆる「不確定」な繰り返しです．「while　end」文を使います．

最初に説明しておくべきだったかも知れませんが，処理の本体を実行するには，第 1 章から第 3 章までに述べたような各種の「演算」をしなければなりません．「演算 (operator)」には

- 算術演算：四則演算，行列演算などなど．
- 関係演算：大小の判定「$>, <, <=, >=$」，等しいかどうかの判定「$==$，$\sim =$」など．
- 論理演算：「and: &」「or：|」「not：$\sim$」の演算．

があります．関係演算と論理演算は「条件文」の部分を書くときに使います．いずれも

- 条件が「真」のとき：「1」を出力する．
- 条件が「偽」のとき：「0」を出力する．

ことになっています．

さて，確定した繰り返しの命令は，次の「for」文です．

```
for  index=1: 2: 100
  処理の本体；
end
```

index の部分は繰り返しの回数を指定しています．この場合ですと「 index という変数を 1 から 2 づつ増やしながら 100 まで（実際は 99 まで）処理本体を繰り返せ」ということになります．

次に，不確定な繰り返しの命令です．これには「while」文を使います．

```
while  expression
  処理の本体；
end
```

expression の部分に判定条件文を書きます．たとえば，index<100 といった具合です．良い例とは言えませんが，例 5.1 の wa3 のプログラムを while 文で書き直してみましょう．

```
function a=wa4(n)
% This is my 5th program : the addition from 1 to n
a=0;
i=1;
while i < n+1
  a=a+i;
  i=i+1;
end
```

条件文の「判定条件」はいつも気をつかうところです．なぜ n+1 なのでしょうか．また，ここを n とするにはどうしたらいいのでしょうか．判定文には，< を使うかあるいは <= を使うかと言った細かいところにも気を配る必要があります．

4.3　判断

プログラムの流れを「条件（場合)」に従って枝分かれさせる命令が判断文と言えます．プログラムの流れを制御する (flow control) と言ったりもします．これもプログラムをする上でなくてななら ない

大切な文です．場合分けは，基本的に「if　else　end」文で処理します．

- ```
  if　論理式
    処理の本体；
  end
  ```
- ```
  if　論理式
    処理の本体 1；
  else
    処理の本体 2；
  end
  ```
- ```
  if　論理式 1
    処理の本体 1；
  elseif　論理式 2
    処理の本体 2；
  else
    処理の本体 3；
  end
  ```

などの形式があります．適当に条件となる論理式に従って「場合分け」を増やしてやればいいことになります．

【例 4.2】

2 つの 2 × 2 の行列 A, B を入力し，'wa' と 'seki' を判定して，和または積を計算するプログラムを作ってみよう．

【解説】

```
function C=WaorSeki(A,B,ch)
% if ch=='w' then C=A+B and if ch=='s' then C=A*B.
if ch=='w'
  C=A+B;
elseif ch=='s'
  C=A*B;
else
  disp('Cannot calculate!!');
end
```

となります．'w' を入れると wa（和）が，'s' と入れると seki（積）が出力されます．どちらにも該当しない場合は，'Cannot calculate!!' と出力するようにしました．ちょっと苦しい判定です．もっとスマートな条件を考えて見て下さい．

枝分かれの場合の数が多いときは「switch　case　end」文を使うと良いでしょう．

≫ help switch

で調べておきましょう．
```

4.4　タートル・グラフィック

さて，これでプログラムの命令の説明は終わりです．簡単ですねえ．本当にプログラムできるの？ と思ってしまいます．私の経験からは，これで何でもできると保証できます．あとは，具体的に問題に当たって砕けろといったところでしょうか．

それでは，簡単な問題をやってみましょう．

【例 4.3】

タートル・グラフィックスの基本命令を作り，色々な繰り返し図形を描け．タートル・グラフィックスとは，グラフィックスの画面の原点に仮想の「亀」が上（y 軸）を向いている初期値から，回転と前進の命令にしたがって亀の軌跡を描くプログラムのことです．

【解説】 さっそく，初期状態を設定するスクリプト，回転と前進の関数を書いてみましょう．

1. タートルの初期化プログラム（スクリプト）

```
function TI()
% turtle initialization: save as "TI.m"
global U X Path deg
U=[0;1];
X=[0;0];
Path=X;
deg=pi/180;
```

ここで，U は方位を記憶する単位ベクトルです．最初は y 軸を上方向に向いています． X は通過する平面の点を記憶する変数，Path は現在のタートルの位置を記憶します．deg は角度を radian に変換する係数です．これらの変数は，次に定義する回転と前進の関数からも使えるように「大域変数 (global)」として記憶します．

2. 回転させる関数

```
function R(a)
% rotation by a degree: save as "R.m"
global U X Path deg
theta=a*deg;
U=[cos(theta)  sin(theta); -sin(theta) cos(theta)]* U;
```

3. 前進させる関数

```
function F(s)
% forward s length: save as "F.m"
global U X Path deg
Path=Path+s*U;
X=[X Path];
```

4. ジャンプさせる関数

```
function J(s)
```

```
% jump s length: save as "J.m"
global U X Path deg
Path=Path+s*U;
X=[X NaN Path];
```

5. 表示のスクリプト

```
function ST()
% show graphics:save as "ST.m"
global U X Path deg
plot(X(1, :), X(2, :));
axis square;  axis off;
set(gcf, 'Color',[1 1 1]);
figure(1);
```

これで完成しました．なんて簡単なプログラムでしょう．では，実験しましょう．次のスクリプトを実行して下さい．

6. クモの巣スクリプト No. 1

```
%
TI;
for i=1:20
  for j=1:10
    F(1);
    R(108);
  end
  R(18);
end
ST;
```

7. クモの巣スクリプト No. 2

```
Tinit;
for i=1:10
  for j=1:10
    F(1);
    R(108);
  end
  R(36);
end
ST;
```

8. 木の枝を描く．まず branch.m を作り，rectree で動かす．

```
function branch(length, level)
% save as "branch.m"
if level==1
  return;
else
  F(length);
  R(45);
```

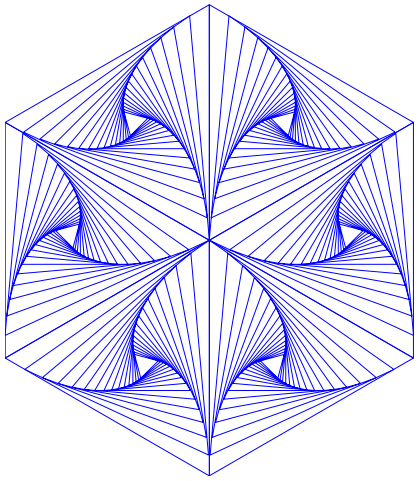


図 4.1　タートルグラフィックスで描いた三角形の集合.

```
  branch(length/2, level -1);
  R(-90);
  branch(length/2, level -1);
  R(45);
  F(-length);
end
```

次のプログラムで木を描きます.

```
function rectree(length, level)
%  recursive tree:save as "rectree.m"
TI;
branch(length, level);
ST;
```

試しに,

≫ rectree(10, 5)

と入力してみましょう. どうでしたか. 木が描けるはずです.

タートル・グラフィックスはなかなかおもしろい課題です. 色々とプログラムの手法を勉強するにもってこいの話題だと思います. 自分の線図を描いてみましょう.

これまで関数をバラバラに定義してきましたが, ここで全体を 1 つの M-file に纏めて書いておきましょう. こうしておくと後日何のプログラムだったか忘れてしまったときに読みやすくなります. ここでは, subfunction を使うサンプルプログラムにもなっています. なお, 行の節約印刷のため, 部分的に複文にしてあります.

```
function op()
% ------------------------------------------------------------
% - optical art: coded by H. Kawakami Oct. 23, 1998
% - main program: This main program is replced by the problem
% - you want to solve.
```

```
% ------------------------------------------------------------
global U X Path deg parity
TI(1);
for i=1:6
  parity=-parity;  A(1);  B(1,15);  H([0;0]);
  if parity==1
    R(60*i);
  else
    R(60*i+60);
  end
end
ST(1)
% ------------------------------------------------------------
% - subfunctions A(a) and B(a, b) are used
% - in the main program op()
% ------------------------------------------------------------
function A(a)
% draw triangle
global U X Path deg parity
for i=1:3
  F(a);  R(parity*120);
end

function B(a,b)
% draw small triangles
global U X Path deg parity
c=a;
for i=1:b
  R(parity*125);  c=0.956*c;  F(c);  R(parity*120);  c=0.9*c;
  F(c);  R(parity*120);  F(c);
end

% ------------------------------------------------------------
% - The following subfunctions are commonly used
% - for turtle graphics
% ------------------------------------------------------------
function TI(q)
% turtle initialization
global U X Path deg parity
U=[0;1];  X=[0;0];  Path=X;  deg=pi/180;  parity=1;

function ST(q)
% show turtle
global U X Path deg parity
plot(X(1,:),X(2,:));  axis square;  axis off;
set(gcf,'Color',[1 1 1]);  figure(1)

function R(r)
% rotation by r degree
global U X Path deg
```

```
theta=r*deg;  U=[cos(theta) sin(theta);-sin(theta) cos(theta)]*U;

function F(s)
% forward s length
global U X Path deg
Path=Path+s*U;  X=[X Path];

function H(h)
% return to home position
global U X Path deg
U=[0;1];  Path=h;  X=[X [NaN; NaN] Path];
% ------------------------------------------------------------
```

4.5　微分方程式を解く：力学系の状態表示

4.5.1　発振のメカニズムをみる

さて，少しは役に立つ問題を考えてみましょう．時間と共に変化する現象をみるときは，考えているシステムの数学的モデルが微分方程式となります．たとえば，電気回路の過渡現象，力学や制御でみる状態の運動，それに生物が持つ様々なリズムなどを説明するために，微分方程式のモデルが活躍しています．ここでは，2 次元のリズムを生成するモデルとしてよく知られたファン・デア・ポール (van der Pol) 方程式を解く問題を考えることにします．

次の方程式を考えましょう．

$$\frac{d^2x}{dt^2} - \varepsilon\left(1 - x^2\right)\frac{dx}{dt} + x = 0 \tag{4.1}$$

この方程式は，もともと真空管発振器の発振現象を説明する最も簡単なモデルとして van der Pol によって提案された方程式です．このモデルの簡単な説明は「回路 3 講義補充ノート」pp. 62-66 にしておきました．興味のあるひとは参照して下さい．式 (4.1) は，初期値が原点以外どこにあっても，時間の経過とともに安定な振動解（これをリミット・サイクルという）に落ち着いて行きます．この様子をみることにしましょう．

一般に，微分方程式を数値計算するには，まず方程式を一階の連立微分方程式に書き直してから計算します．そこで，式 (4.1) を次の連立方程式に書き直します．

$$\begin{aligned} \frac{dx}{dt} &= y \\ \frac{dy}{dt} &= \varepsilon\left(1 - x^2\right)y - x \end{aligned} \tag{4.2}$$

実は，このように書き直すことによって，この方程式は 2 つの状態 $(x,\ y)$ についての速度ベクトルを記述しているということが分かります．そこで，状態 x や y が時間と共にどのように変化するのかが知りたい訳です．原点の近くに初期状態 $(x_0,\ y_0)$ をおいて，この初期値から出発する解がリミット・サイクルに巻き付いて行く様子をみることにしましょう．

4.5.2　とりあえず試しに数値積分してみる

MATLAB には常微分方程式を解いてくれる関数がたくさん用意されています．これを使って最も短いプログラムを作ってみましょう．「ode45」という関数を使います．これは，4 次 5 段のルンゲ・クッタ法 (Runge-Kutta method) を使った微分方程式を解くための関数です．積分する時間間隔と初期値を引数に与えると，時間間隔を適当なサイズに分割し，その各点での積分結果を返してくれます．早速プログラムしてみましょう．

```
% van der Pol equation: the first essay.
% save as vdP1.m
t0=[0, 20*pi];
x0=[0.1; 0.0];
[t, x]=ode45('myvdP', t0,x0);

plot(x(:, 1), x(:, 2));
axis square
% end of program
```

このプログラムには何処にもファン・デア・ポール方程式が書いてありません．方程式を書いた関数「myvdP」は別に用意しなくてはなりません．つぎの関数を書いて，関数名と同じ名前で保存しましょう．

```
function dx=myvdP(t,x)
% van der Pol equation
% save as myvdP.m
dx=[x(2); 0.5*(1-x(1)*x(1))*x(2)-x(1)];
% end of function
```

プログラムができたので，早速実行してみましょう．

≫ vdP1

どうでしょうか．原点近くから巻きだした渦巻き状の曲線が得られたでしょう．

では，次にこのプログラムを少しだけお化粧してあげることにしましょう．次のプログラムがそれです．

```
% van der Pol equation: the second essay.
% save as vdP2.m
t0=[0, 20*pi];
x0=[0.1; 0.0];
[t, x]=ode45('myvdP', t0,x0);
% phase portrait
figure(1);
plot(x(:, 1), x(:, 2));
axis([-2.5, 2.5, -2.5, 2.5]);
axis square
% wave forms of x(t) and x-dot(t)
figure(2);
```

```
subplot(2,1,1);
plot(t,x(:, 1), 'r');
axis([0, 20*pi, -2.5, 2.5]);
subplot(2,1,2);
plot(t,x(:, 2), 'r');
axis([0, 20*pi, -2.5, 2.5]);
% end of program
```

これで，ウインドウ No. 1 には相平面図 (phase portrait) が，また No.2 には $x(t)$ と $dx(t)/dt$ の波形が描かれました[*1]．

このプログラムでは，式 (4.2) の ε は関数「myvdP」の中で 0.5 と与えられています．これを色々変えて波形や相平面図がどう変化するか観察してみましょう．スクリプト・ファイルだったメイン・プログラムを関数にして，引数として ε を与えるようにすると便利でしょう．このときは，ε を大域変数として，関数「myvdP」に引き継ぐといいと思います．そうそう，ε を eps とするとマズイですね．eps はマシン・イプシロンとして予約された変数です．別の単語を選びましょう．苦し紛れに myeps とかやった人いませんか．変数の名前の選択は色々「流儀」があって大変です．まあ，好きなように名付けて下さい．

4.5.3　動きをみるプログラム

動的システムのシミュレーションには，時間的に実際状態がどう変化しているか「目に見える」ことが大切です．前小節のプログラムでは，関数「ode45」で一挙に計算してその結果を表示するだけですから，この動きを見ることができません．この点を改良してみましょう．

```
function vandP2
% ----------------------------------------
% main program
% phase portrait of van der Pol equation
% ----------------------------------------
global hndl

t=0; tmax=20*pi; h=0.1; x=[0.1; 0.0]; X=[x x];

GI(1);

while t<tmax
   [t, x]=RK(t,x,h);
   X=[x X(:,1)];
   set(hndl,'xdata',X(1,1:2),'ydata',X(2,1:2));
   drawnow;
end
%end of main
```

[*1] 各ウインドウを切り換えて見るには，ディスプレー画面の一番下にある横長の「タスク・バー」にある各ウインドウの名前をダブル・クリックするといいでしょう．

```
% van der Pol equation
function dx=vP(t,x)
dx=[x(2); 0.5*(1.0-x(1)*x(1))*x(2)-x(1)];

% 4th order Runge Kutta method
function [t, x]=RK(t,x,h)
f1=vP(t,x);
f2=vP(t+h/2,x+h*f1/2);
f3=vP(t+h/2,x+h*f2/2);
f4=vP(t+h,x+h*f3);
t=t+h;
x=x+h*(f1+2*(f2+f3)+f4)/6;

% Graphics initialization
function GI(g)
global hndl

hf=figure('Units','Normalized','Position',[.2 .2 .6 .6]);
hndl=line('color','r','linestyle','-','linewidth',1,...
  'erase','none','xdata',[],'ydata',[]);
axis([-3 3 -3 3]);  box on;
title('Phase Portrait');  xlabel('x(t)');  ylabel('x-dot');
grid on;  axis square;
figure(1);
```

このプログラムでは，4 次のルンゲ・クッタ法を使って，時間刻み h 毎に方程式を解いてそれを画面に表示することにしています．「line」命令を使って，引き続く時間区間の 2 つのデータを X に蓄えて，これを線で結んでいます．線の太さは，関数 GI() の中の line の中で 'linewidth' につづく数字を大きくすると，太くなります．なお，line インスタンスを作ると，自動的に figure と axes のデフォルト・インスタンスが作られます．

なお，このプログラムでルンゲ・クッタ法を定義している関数をみると，これは数値解析で習ったとおりの公式が書かれているだけです．方程式の次元に関係なく定義できていることに注意して下さい．次元の大きな問題を解くときも，状態 x の次元を大きくし，グラフィックスの部分を変えるだけで，このプログラムはそのまま使うことができます．

main program をみると，これはたったの 10 行で書けています．まず，使う変数を初期化し GI() でグラフィックスを初期化の後，while 文で主計算をしています．もう少し整理をして，main のスタイルを

```
% main program style
Init_Prob       % Initialization of the problem
Init_Graf       % Initialization of graphics
Computation     % Main routine
End_of_Prog     % Termination of program
```

のように 4 行プログラムにするといいでしょう．ただ，このようにすると global にたくさんの変数を定義するか，あるいは関数の引数の数が多くなってしまいます．どの辺で折り合いをつけるか問題です．

4.5.4　Rössler 方程式のカオス

せっかくですから，3 次元の力学系を解いて上述のプログラムに本質的な変更は何もないことを見ておきましょう．例として Rössler 方程式を取り上げることにします．

$$\begin{aligned} \frac{dx}{dt} &= -x - z \\ \frac{dy}{dt} &= x + ay \\ \frac{dz}{dt} &= bx + xz - cz \end{aligned} \tag{4.3}$$

この方程式は，最後の式に xz の非線形項があるだけですが， Rössler band と呼ばれる奇妙なアトラクタ[*2]を持っています．このアトラクタは「カオス」アトラクタの典型的な例です．

いま，式 (4.3) に含まれるパラメータや初期値，解の表示範囲は次の値としてプログラムします．

$$a = 0.35,\ b = 0.4,\ c = 4.5$$

$$x_0 = z_0 = 0.0,\ y_0 = 3.0$$

$$-6 \le x \le 10,\ -8 \le y \le 8,\ 0 \le z \le 8$$

作ったプログラムは以下の通りです．

```
function Rossler
% main program
global hndl1 hndl2 hndl3

t=0; tmax=200*pi; h=0.1; x=[0.0;3.0; 0.0];
L=20;  X=x*ones(1,L);

GI(x);

while t<tmax
   [t, x]=RK(t,x,h);
   X=[x X(:,1: L-1)];
   set(hndl1,'xdata',X(1,1),'ydata',X(2,1),...
       'zdata',X(3, 1));
   set(hndl2,'xdata',X(1,1:5),'ydata',X(2,1:5),...
       'zdata',X(3, 1:5));
   set(hndl3,'xdata',X(1,L-1:L),'ydata',X(2,L-1:L),...
       'zdata',X(3, L-1:L));
   drawnow;
```

[*2] アトラクタとは，広い意味での安定な定常状態のことです．「ちっとも落ち着かないのに定常状態もないものだ」と思うかも知れません．だから，広い意味といったのです．カオス・アトラクタに吸い込まれた状態は，折り重なってみえる帯状の集合内を閉じることなく彷徨しつづけます．実はこのアトラクタ内には無限個の周期軌道や非周期軌道があって，それらはすべて不安定なのですが全体としては有界領域に閉じこめられ，一体となってアトラクタを作っているのです．「奇妙な」とはこのことを指しています．

```
end
%end of main

% Rossler equation
function dx=fn(t,x)
dx=[-(x(2)+x(3)); x(1)+0.35*x(2); 0.4*x(1)+x(1)*x(3)-4.5*x(3)];

% Runge Kutta method
function [t, x]=RK(t,x,h)
  f1=fn(t,x);
  f2=fn(t+h/2,x+h*f1/2);
  f3=fn(t+h/2,x+h*f2/2);
  f4=fn(t+h,x+h*f3);
  t=t+h;
  x=x+h*(f1+2*(f2+f3)+f4)/6;

% Graphics initialization
function GI(x)
global hndl1 hndl2 hndl3
hf=figure('Units','Normalized','Position',[.2 .2 .6 .6]);
hndl1=line('color','r','linestyle','.','markersize',20,...
     'erase','xor','xdata',x(1),'ydata',x(2),'zdata',x(3));
hndl2=line('color','r','linestyle','-','linewidth',1,...
     'erase','none','xdata',[],'ydata',[],'zdata',[]);
hndl3=line('color','b','linestyle','-',...
     'erase','none','xdata',[],'ydata',[],'zdata',[]);
  axis([-6 10 -8 8 0 8]);
  box on;
  title('Phase Portrait: Rossler equation');
  xlabel('x(t)');  ylabel('y(t)');  zlabel('z(t)');
  grid on;  axis square;
  figure(1);
```

3 次元になったと言っても，'zdata' をつけ加えただけです．すこし動きをみるために軌道の頭の部分に着色しました．L を大きくすると頭の赤い軌道の部分が長くなります．これでハンドルが hndl1, hndl2, hndl3 と増えています．また GI(x) で初期値 x を渡していることも注意してください．実は，この着色表示は MATLAB のデモに Lorenz アトラクタの例があって，そのプログラムよりアイデアをもらってきたものです．このデモ・プログラムと上のプログラムを比較してみてください．subfunction を使った分だけ，プログラムがすっきりして読みやすくなっているはずです．勿論，デモ・プログラムには色々とアクセサリーが付いています．これは，次の章で考えることにしましょう．

4.6　よりよいプログラムを書くために

MAT LAB はアレーの行列演算を得意とするプログラミング言語と言えます．このことをできるだけ活用して高速計算のできるプログラムを書くことが大切です．このためには次の 2 つの事項を心がけてプログラムすることが推奨されています．

- ループのベクトル化を図る．すなわち，可能な限り「for end」文を用いないようにする．
- 前もってプログラムの最初に，使用予定のアレーは定義しておく．

この 2 つ目の心がけは，メモリーの効率的な利用にも役立ちます．あらかじめ使いそうなサイズの予約を取っておくと，メモリーのフラグメンテーションを回避できるでしょう．よいプログラムを作るには，まず他人の「良い」プログラムをまねることから始めるといいでしょう．

これらの例として，MATLAB のマニュアルには次のような例が載っています．まず，最初の例です．

```
i=0;
for t=0:0.01:10
  i=i+1;
  y(i)=sin(t);
end
```

とする代わりに，次のように書くことを勧めています．

```
t=0:0.01:10;
y=sin(t);
```

2 番目の例です．

```
y=zeros(1,100);
for i=1:100
  y(i)=det(X^i);
end
```

つまり，計算に先だって y のアレーを確保して置くわけです．

このようなちょっとした心がけがよりよいプログラムの生産に役立つことでしょう．「他人のプログラムのいい点を拝借すること」それに「ちょっとした心がけ」いつもコーディングの際には思い出してください．それに，時には色々なプログラム言語でどんなプログラムがなされているか，探検旅行に出かけるのもいいでしょう．Mathematica などというおもしろい言語などは心ときめく探検の良い例といえます．

まずは手元にある C 言語の教科書をあけて，目に付いたプログラムを MATLAB プログラムに書き直すことから始めてください．では，よいご旅行を！

4.7　演習問題

1. 窓を一つ開いて中央に Hello MATLAB World と書くプログラムを作ってください．
2. タートル・グラフィックスで自分流の図形を 3 個程度描いてください．
3. 4.4 にある optical art では 6 角形の図が描けます．では，同様な 8 角形，12 角形および 16 角形の図を描いてください．
4. 4.5.2 にある最も簡単な van der Pol 方程式のプログラムを変形して，(t, x, y) 表示の 3 次元図形を描いてください．
5. 電気回路の過渡現象は，線形システムの時間応答を理解する上で最もよい物理系であると言えます．特に LRC 回路を正弦波で強制振動させる例は，典型的な過渡現象の例を与えてくれます．次の方程式は，正規化された LRC の回路方程式です．

$$\begin{aligned} \frac{dx}{dt} &= y \\ \frac{dy}{dt} &= -x - ky + B\cos(\Omega t + \theta) \end{aligned} \tag{4.4}$$

   ここに，k は減衰定数，B は外力の振幅，そして θ は外力の位相を表しています．この方程式を解くプログラムを本文の van der Pol や Rössler の方程式のプログラムを参考にして作ってください．
6. 漸化式は，離散時間力学系を記述する運動方程式です．次の漸化式の軌道を (x, y) 平面に点列で表示するプログラムを作ってください．

$$\begin{aligned} x_{n+1} &= y_n + 0.008\left(1 - 0.05y_n^2\right) y_n + F(x_n) \\ y_{n+1} &= -x_n + F(x_{n+1}) \end{aligned} \tag{4.5}$$

   ここに

$$F(x) = \mu x + \frac{2(1-\mu)x^2}{1+x^2} \tag{4.6}$$

   とし，$\mu = -0.8$ とします．スケールは自分で試行錯誤して適当に定めてください．なお，この写像は Gumowski-Mira の写像と呼ばれています．
   そうそう，もっと簡単な次の 2 次写像はどうでしょうか．

$$\begin{aligned} x_{n+1} &= ax_n + y_n \\ y_{n+1} &= x_n^2 + b \end{aligned} \tag{4.7}$$

   まずは，パラメータとして $a = -0.1, b = -1.7$ と定めて様子を見てください．

第 5 章

グラフィックスと GUI を使う

ディスプレー画面に描かれた種々のグラフィックスを操ることや，プッシュ・ボタンなどを取り付けて使い手に優しいソフトにすることは，最近のプログラミングには避けて通れないことです．幸い MATLAB のグラフィックスや GUI(Graphical User Interface) はとても良くできています．これを少し学んでおきましょう．

かいつまんで言うと，この章の内容は，MATLAB に基づくグラフィックス・オブジェクトに関する話：OOP(object oriented programming) とイベント駆動型 (event driven) プログラミングの入門になっています．

5.1 インスタンスはオブジェクトにあらず

MATLAB のグラフィックス・オブジェクトは，おおまかに分けて次の 2 種類です．

- グラフィックス・オブジェクト．主に axes とその子供たち．これらの幾つかについてはこれまでにたびたび気にせずに使ってきました．
- グラフィカル・ユーザ・インターフェイス (GUI: Graphical User Interface) のためのオブジェクト．Uicontrol とその子供たちや Uimenu など．デモ・プログラムで見かけるような，ソフトを動かす人（ユーザ）が使う，様々なプログラム制御用のグラフィカルな「小物」です．プログラマーは，ユーザの働きかけをイベント (event) として記憶し，これに対応した処理プログラムを用意しなければなりません．イベントで動くように作ったプログラムのことをイベント駆動型 (event driven) プログラムといいます[*1]．

これら MATLAB のグラフィックス・オブジェクトは，それぞれの型（タイプ）の間に図 5.1 に示したような親子関係を持っています．これらの名前（タイプ）は，それ自身オブジェクトを作る関数名を兼ねています．表 5.1 参照．

作られたオブジェクトは，正確にいうとオブジェクトではなく，オブジェクトのインスタンス (instance) です．「実例」とでも訳すのでしょうか．「インスタンスとオブジェクト」の関係は，ちょっ

[*1] 当然ですが，ユーザにとって優しいソフトほど，プログラムが複雑となり作るとなると苦痛を伴う．簡素 (simple) で使い勝手がよいプログラムづくりを目指したいものです．

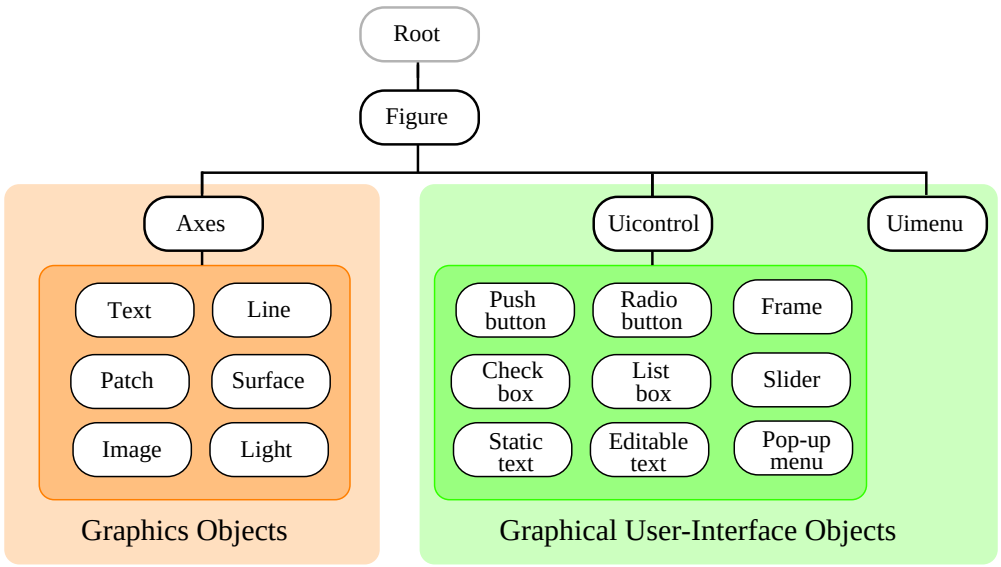


図 5.1　グラフィックス・オブジェクトとその親子関係.

表 5.1　グラフィックス・オブジェクトを生成する関数

生成関数	作られたオブジェクト
figure	グラフィックスを表示するウインド
uicontrol	GUI 用の小物
uimenu	Figure 用のメニュー
axes	直角座標系；以下の子供を持つ
image	2-D 画像
light	方向性のあるライト
line	線分
patch	多角形のセル
surface	面分
text	文字列

とニュアンスは違いますが,「白馬と馬」のような関係です[*2]．一般に，抽象名詞は使いづらいですね．

オブジェクト生成関数で作られた，オブジェクトのインスタンス（以後，こんな持って回ったような面倒なことを言わずに，単に「できたオブジェクト」のように言うことにします）には，他のインスタンスと間違わないように，1 つ 1 つ異なる「識別番号」が付けられます．これは，オブジェクトのハンドル (handle) [*3]と呼ばれています．

ハンドルは,

- root が 0,
- figure には 1 から始まる正の整数,
- その他のオブジェクトには実数

が付けられるように設計されています．

```
≫ hf=figure
hf=
    1
≫ ha=axes
ha=
   13.0006
```

のように，各インスタンスのハンドルが hf, ha に代入されます．figure の場合は，ハンドルが番号としてウインドウのタイトルバーに表示されています．なお，親オブジェクトを定義せずに，いきなり

```
≫ hl=line
hl=
    20.0004
```

のように line をつくると，親となる figure と axes がデフォルトで作られ，line オブジェクト生成関数のハンドルが返されます．

現在使っている（あるいは最後に指定した）オブジェクトはカレント (current) オブジェクトと呼ばれています．current figure, current axes, current object などです．

```
≫ gcf   % returns the value of the current figure's handle
≫ gca   % returns the value of the current axes' handle
≫ gco   % returns the value of the current object's handle
```

最後のカレント・オブジェクトは，マウスでクリックされたオブジェクトのハンドルを返します．あれこれオブジェクトをクリックした後，gco を実行しハンドルの値が変わることを確かめてみてください．

オブジェクトを作ったら，ハンドルを hndl1, hndl2, ... のように変数に蓄えておくと，後でこのハンドルを操作してオブジェクトを操ることができます．

[*2] 「白馬は馬に非ず」(公孫竜子白馬論: 戦国時代の公孫竜の議論の一つ) 馬と白馬は同一の概念ではないから、白馬は馬ではないの意．詭弁の例として用いる．白馬非馬論．(広辞苑)

[*3] ハンドルはインスタンスへのポインターのポインターと思ってもいいのでしょう．その場合，オブジェクトはオブジェクト型（構造体を一般化した型）と考えるといい訳です．

【例 5.1】

適当なサイズの figure window を 2 つ開いて，その中に 2 つの axes を描くプログラムを作ってください．

【解説】 プログラムの一例を示しておきます．ハンドルを操作して希望する axes に作図する方法を学ぶことができます．figure window や axes は積極的にインスタンスを定義して，これらのハンドルを陽に使うようにすればプログラムが読みやすくなります[*4]．もちろん，各インスタンスの制御も明確に行うことができます．

```
% Two figure windows each of which contains two axes
hf1=figure('Units','Normalized','Position',[.05 .3 .4 .4]);
hf2=figure('Units','Normalized','Position',[.50 .3 .4 .4]);
ha1=axes('Parent',hf1,'Position',[.05 .3 .4 .4]);
ha2=axes('Parent',hf1,'Position',[.55 .3 .4 .4]);
ha3=axes('Parent',hf2,'Position',[.05 .3 .4 .4]);
ha4=axes('Parent',hf2,'Position',[.55 .3 .4 .4]);
%==== example of plots ====
t=0:pi/30:2*pi;
axes(ha1);
plot(t,sin(t),'Parent',ha1);
axis([0 2*pi -1 1]);
axes(ha2);
plot(t,sin(t).^2,'Parent',ha2);
axis([0 2*pi -1 1]);
axes(ha3);
plot(t,cos(t),'Parent',ha3);
axis([0 2*pi -1 1]);
axes(ha4);
plot(t,cos(2*t),'Parent',ha4);
axis([0 2*pi -1 1]);
% end of program
```

さて，作られたグラフィックス・オブジェクトは，たくさんの性質を持っています．色や大きさ，どこに描かれているかといった位置情報などはこの性質の例です．これらは，オブジェクトのプロパティー (property) と呼ばれています．実際，多くのプログラムでは，定義したオブジェクトに対して

- ハンドルを指定して，
- オブジェクトのプロパティーを制御する[*5]

作業をコードすることになるでしょう．

そこで，オブジェクトのプロパティーを「得る」ことと「設定する」ことが必要になります．これを行う関数が， get と set です．

get(オブジェクトのハンドル, プロパティー名)

[*4] デフォルトの gcf や gca を多用すると，便利である反面，何が何処で定義されているのか分かりづらくなります．
[*5] インスタンスにメッセージ (message) を送ると言ったりする．

set(オブジェクトのハンドル, プロパティー名，プロパティーの値)

get の場合は，プロパティーの値が返されます．

【例 5.2】

適当なサイズの窓を開いて，その中央に「Hello MATLAB world!」と書くプログラムを作ってください．窓の位置やサイズをマウスで変化させて，これらのプロパティーが変化していることを確かめてください．窓を複数個開いて，あれこれ遊んでみましょう．

【解説】 プログラムの一例を示しておきます．最初は，for 文を除いて実行してください．

```
%hello MATLAB
center=0.5; dx=0.6; dy=0.2;
x0=center-dx/2; y0=center-dy/2;
for i=1:9,
  r=rand(1,3);s=ones(1,3)-r;
  hf=figure('Color',[r(1),r(2),r(3)]);
  set(hf,'Position',[250+20*i,300-20*i,300,200]);
  ha=axes('Position',[x0,y0,dx,dy]);
  set(ha, 'Visible','off');    %axis off;
  ht=text(center,center, 'Hello MATLAB world!');
  set(ht,'Color',[s(1),s(2),s(3)]);
  set(ht, 'HorizontalAlignment','Center');
end
% end of program
```

窓のサイズを変化させても，文字列が常に中央にくることも確かめてください．プロパティーの名前は get を使って調べるといいでしょう．n 番目の窓の色や文字列の色を変えるにはどうすればいいのでしょう．ハンドルを有効に使ってプロパティーを変えてみましょう．

さてここで，root ウインドと figure ウインド，figure ウインドの中に描かれるオブジェクト達の座標について見ておきましょう．図 5.2 参照．

- グラフィックス・オブジェクトの座標原点は，「左端の最下端」である[*6]．
- 座標の単位系 Units は，色々と指定できる．pixels, normalized, centimeters, inches, points などがある．root と figure のデフォルト (default) 値は pixels, axes とその子供たちは normalized である．normalized は全体を 1 と考えた単位のことである．勿論，axis 命令でスケールを変えると，その単位となる．
- グラフィックス・オブジェクトの位置と大きさは，プロパティー Position を [left, bottom, width, height] で与えると定まる．

上のプログラムで figure ウインドは pixels で，axes と text は normalized で指定していることに注意してください．これらはデフォルトを使っています．デフォルト値を変更するには

≫ set(gcf, 'Units', 'Normalized');

のようにします．この設定のあと，get 命令で確かめると変更されたことが確認できます．

[*6] なんと幸せなことでしょう．

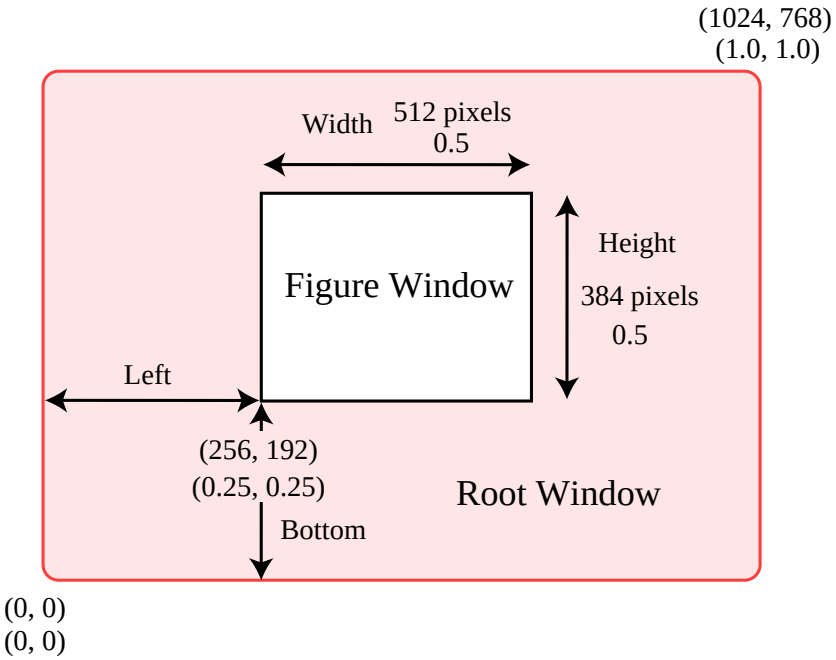


図 5.2　root ウインドと figure ウインドの座標：2 つの単位系.

【例 5.3】

適当なサイズの窓を開いて，正弦波を描き，その図に grid を on-off するプッシュボタンを付けたプログラムを作ってください.

【解説】 プログラムの一例を示しておきます．Uicontrol でプッシュボタンを 2 個定義しています.

```
% grid on off control by push button
t=0:pi/30:10; r=rand(1,3); s=ones(1,3)-r;

hf=figure('Color',[r(1),r(2),r(3)], 'Units','Normalized',...
  'Position',[.2 .2 .6 .6]);

ha=axes('Position',[0.2 0.3 0.6 0.6],'Box','on');
set(ha,'XColor',[s(1),s(2),s(3)],'YColor',[s(1),s(2),s(3)]);

hb=plot(t,sin(t));
set(hb,'Color',[s(1),s(2),s(3)],'LineStyle','o');

set(hf,'DefaultUicontrolUnits','Normalized');
Uicontrol(hf,'Style','Pushbutton','Position',[.25 .1 .2 .1],...
   'Callback','grid on', 'String','grid on');
Uicontrol(hf,'Style','Pushbutton','Position',[.55 .1 .2 .1],...
   'Callback','grid off', 'String','grid off');
% end of program
```

表 5.2　どのオブジェクトも持っているプロパティーの一例

プロパティー	意　　味
ButtonDownFcn	マウスボタン割り込み
ChangeFcn	プロパティーが変わったら作動
Children	子供のハンドル
Clipping	クリップするかしないか
CreateFcn	オブジェクトができたら作動
DeleteFcn	オブジェクトを消したら作動
BusyAction	割り込み制御
HandleVisibility	ハンドルを見せるか否か
Interruptible	割り込ませるか否か
Parent	親のハンドル
Selected	オブジェクトが選択されているか否か
Tag	あなたが付けるラベル
Type	オブジェクトの型
UserData	あなたが書き込んでおくデータ
Visible	可視状態か否か

各ボタンの画面に現れる名前は，String の値が表示されます．また，押したときに実行される関数は Callback の値に入れられています[*7]．この関数のことを「コールバック・ルーティーン (callback routine)」といいます．イベント処理のためのルーティーンです．

一般に，親のプロパティーを変えると，その子供たちの同じプロパティーは一斉に変わります．親の性質を「継承 (inheritance)」するといいます．子供たちに別の値を与えたいときには，その子供のプロパティー値を上書き（定義）してやります．

なお，すべてのオブジェクトが持っているプロパティーが幾つかあります．その一例を表 5.2 に示します．

[*7] 関数名（M-ファイル名）が書かれていると，その関数が呼び出されます．

5.2　グラフィックス・オブジェクト

この節では，オブジェクトのプロパティーの設定方法についてお話します．何しろ各オブジェクトには，猛烈に沢山のプロパティーがあるのですから，いちいち説明することは不可能です．マニュアルを見てくださいと言われても，何処にあるのか探すのも大変です．幸い MATLAB 自身とオンライン・マニュアルを使って大体のことはできるようになっています．

そこで，次のような 3 段階のプロパティー設定サイクルを提案します．

- まずオブジェクトをつくる．ハンドルはしっかりと変数に入れておく．
- get(ハンドル) で作ったオブジェクトのプロパティーを知る．また，set(ハンドル) で各プロパティーの設定可能な値の表を得る．更に分からない場合には，help でオンライン・マニュアルを参照する．
- これらの情報を基にして，必要なプロパティーを set 命令で設定する．

このサイクルを何度か繰り返すと，望みのオブジェクトができると思います．

5.2.1　Figure のプロパティー

Figure のプロパティーは，次の 9 種類の性質に分類されています[*8]．

1. Style and appearance: Menubar, Name, NumberTitle, Resize, Visible, WindowStyle
2. General information: Children, Parent, Position, Tag, Type, Units, UserData, Color
3. Colormap: Colormap, DitherMap, DitherMapMode, FixedColors, MinColorMap, ShareColors
4. Rendering graphics objects: BackingStore, Renderer
5. Current selections: CurrentAxes, CurrentCharacter, CurrentMenu, CurrentObject, CurrentPoint, SelectionType
6. Callback routine and execution: ButtonDownFcn, ChangeFcn, CloseRequestFcn, DeleteFcn, KeyPressFcn, ResizeFcn, BusyAction, Interruptible, WindowButtonUpFcn, WindowButtonDownFcn, WindowButtonMotionFcn
7. Pointer defintion: Pointer, PointerData, PointerShapeHotSpot
8. Figure handles: IntegerHandle, HandleVisibility, NextPlot
9. Printing: InvertHardcopy, PaperOrientation, PaperPosition, PaperPositionMode, PaperSize, PaperType, PaperUnits

さて，そこで一つ figure ウインドを開いて，プロパティーを見てみましょう．

```
》 hf=figure
```

[*8] "Using MATLAB Graphics", Chapter 9. を参照.

```
hf =

     1

》 get(hf)
BackingStore = on
CloseRequestFcn = closereq
Color = [0.8 0.8 0.8]
Colormap = [ (64 by 3) double array]
CurrentAxes = []
CurrentCharacter =
CurrentObject = []
CurrentPoint = [0 0]
Dithermap = [ (64 by 3) double array]
DithermapMode = manual
FixedColors = [ (3 by 3) double array]
IntegerHandle = on
InvertHardcopy = on
KeyPressFcn =
MenuBar = figure
MinColormap = [64]
Name =
NextPlot = add
NumberTitle = on
PaperUnits = inches
PaperOrientation = portrait
PaperPosition = [0.25 2.5 8 6]
PaperPositionMode = manual
PaperSize = [8.5 11]
PaperType = usletter
Pointer = arrow
PointerShapeCData = [ (16 by 16) double array]
PointerShapeHotSpot = [1 1]
Position = [120 120 560 420]
Renderer = painters
RendererMode = auto
Resize = on
ResizeFcn =
SelectionType = normal
ShareColors = on
Units = pixels
WindowButtonDownFcn =
WindowButtonMotionFcn =
WindowButtonUpFcn =
WindowStyle = normal

ButtonDownFcn =
Children = []
Clipping = on
CreateFcn =
DeleteFcn =
BusyAction = queue
HandleVisibility = on
Interruptible = on
```

```
Parent = [0]
Selected = off
SelectionHighlight = on
Tag =
Type = figure
UserData = []
Visible = on
```

それで，これらのプロパティーの設定可能な値は set でみることができます．{ } で囲まれている値が現在設定されている値となっています．

```
》 set(hf)
BackingStore: [ {on} | off ]
CloseRequestFcn
Color
Colormap
CurrentAxes
CurrentObject
CurrentPoint
Dithermap
DithermapMode: [ auto | {manual} ]
IntegerHandle: [ {on} | off ]
InvertHardcopy: [ {on} | off ]
KeyPressFcn
MenuBar: [ none | {figure} ]
MinColormap
Name
NextPlot: [ {add} | replace | replacechildren ]
NumberTitle: [ {on} | off ]
PaperUnits: [ {inches} | centimeters | normalized | points ]
PaperOrientation: [ {portrait} | landscape ]
PaperPosition
PaperPositionMode: [ auto | {manual} ]
PaperType: [ {usletter} | uslegal | a3 | a4letter | a5 | b4 | tabloid ]
Pointer: [ crosshair | fullcrosshair | {arrow} | ibeam | watch |
    topl | topr | botl | botr | left | top | right | bottom | circle |
    cross | fleur | custom ]
PointerShapeCData
PointerShapeHotSpot
Position
Renderer: [ {painters} | zbuffer ]
RendererMode: [ {auto} | manual ]
Resize: [ {on} | off ]
ResizeFcn
ShareColors: [ {on} | off ]
Units: [ inches | centimeters | normalized | points | {pixels} ]
WindowButtonDownFcn
WindowButtonMotionFcn
WindowButtonUpFcn
WindowStyle: [ {normal} | modal ]

ButtonDownFcn
Children
```

```
Clipping: [ {on} | off ]
CreateFcn
DeleteFcn
BusyAction: [ {queue} | cancel ]
HandleVisibility: [ {on} | callback | off ]
Interruptible: [ {on} | off ]
Parent
Selected: [ on | off ]
SelectionHighlight: [ {on} | off ]
Tag
UserData
Visible: [ {on} | off ]
```

【例 5.4】

MATLAB にはカーソルの位置を示すポインター (pointer) が 16 種類用意されているようです．これらを表示するプログラムを作りなさい．

【解説】 次のプログラムは，デモ・プログラムがら拝借したものです．nargin は入力の引数の数を返す関数です．最初，

```
》 ptrdemo
```

でプログラムを動かすと，引数の数が 0 なので，プログラムは初期化の部分を通ります．その最後の部分でマウスボタンが動くと割り込みがかかる

```
set(gcf,'WindowButtonMotionFcn','ptrdemo arg');
```

でダミーの引数 arg がついた，自分自身を起動して else 以下の部分を実行します．

カスタム・ポインターの定義は 16×16 の行列にポインター・マークを定義に，登録すれば使えます．行列の要素の値は，1 が黒，2 が白，NaN が透けて見える透明です．

```
function ptrdemo(arg)
% Pointers demo
ptext = {'crosshair','fullcross','arrow','watch';...
         'topl','topr','botl','botr';...
         'left','top','right','bottom';...
         'circle','cross','fleur','custom'};
if nargin == 0
   clf reset;
   g = [.5 .5 .5];
   ha = axes('Units','Normal','Position',[0 0 1 1],...
        'GridLineStyle','-','Xcolor',g,'Ycolor',g);
   set(ha,'Color',[.6 1 .6],'Xtick',[1 2 3],'Ytick',[1 2 3],...
  'XLim',[0 4],'YLim',[0 4]);
   grid on;

   % define a custom pointer
   P=ones(16)+1; P(1,:)=1;P(16,:)=1;P(:,1)=1;P(:,16)=1;
   P(1:4,8:9)=1;P(13:16,8:9)=1;P(8:9,1:4)=1;P(8:9,13:16)=1;
   P(5:12,5:12)=NaN;
   set(gcf,'Pointer','custom','PointerShapeCData',P,...
```

```
        'PointerShapeHotSpot',[9 9]);

    for y = 1:4
        for x = 1:4
           ht=text(x-0.95,y-1,ptext{y,x});
           set(ht,'FontSize',16,'VerticalAlignment','bottom');
        end
    end
    set(gcf,'WindowButtonMotionFcn','ptrdemo arg');
    set(gcf,'Units','Normal');
else
pt = get(gcf,'CurrentPoint');
ind = floor(pt * 4) + 1;
x = ind(1);
y = ind(2);
if (x > 0) & (x < 5) & (y > 0) & (y < 5)
       set(gcf,'Pointer', ptext{y,x});
    end
end
% end of program
```

5.2.2　Axes のプロパティー

Axes のプロパティーも，次の 10 種類の性質に分類されています[*9].

1. Style and appearance: Box, Clipping, GridLineStyle, Layer, LineStyleOrder, LineWidth, MinorGridLineStyle, SelectionHighlight, TickDir, TickDirMode, TickLength, Visible
2. General information: Children, ClippingMode, CurrentPoint, Parent, Position, Selected, Tag, Type, Units, UserData
3. Annotation: FontAngle, FontName, Fontsize, FontUnits, FontWeight, Title, XLabel, YLabel, ZLabel, XTickLabel, YtickLabel, ZTickLabel, XTickLabelMode, YTickLabelMode, ZTickLabelMode,
4. Axis control: XDir, YDir, ZDir, XGrid, YGrid, ZGrid, XLim, YLim, ZLim, XLimMode, YLimMode, ZLimMode, XScale, YScale, ZScale, XTick, YTick, ZTick, XTickMode, YTickMode, ZTickMode, XAxisLocation, YAxisLocation
5. Viewpoint: CameraPosition, CameraPositionMode, CameraTarget, CameraTargetMode, CameraUpVector, CameraUpVectorMode, CameraViewAngle, CameraViewAngleMode, View
6. Scaling and aspect ratio: DataAspectoRatio, DataAspectoRatioMode, PlotBoxAspectRatio, PlotBoxAspectRatioMode, ProjectionType,
7. Callback execution: BusyAction, ButtonDownFcn, CreateFcn, DeleteFcn, Interruptible
8. Rendering method: DrawMode

[*9] "Using MATLAB Graphics", Chapter 10. を参照.

9. Targeting axes: HandleVisibility, NextPlot
10. Color: AmbientLightColor, CLim, CLimMode, Color, ColorOrder, XColor, YColor, ZColor

さて，そこで一つ Axes を作って，プロパティーを見てみましょう．勿論，紙面の節約で表示を示すようなことはいたしません．みなさんは，実行して確かめてください．

```
》 ha=axes
》 get(ha)
》 set(ha)
```

【練習 5.1】

Axes とその子供たちのプロパティーをうまく設定して，例 3.3 と練習 3.2 の 2 つのボード線図を 1 つの図に描いてください．y 軸の目盛りは，左側がゲイン線図用，右側が位相線図用となるように振り付けてください．

【解説】 とりあえずゲイン線図を描いて，その情報を読みとって，位相線図を上書きしています．

```
function bode
% Bode diagram
x=logspace(-1,1,100);
zeta=.1;
y=-10*log10((1.0-x.^2).^2+(2.0*zeta*x).^2);
z=-180*atan2(2.0*zeta*x,1.0-x.^2)/pi;

hs1=semilogx(x,y,'b');
ha1=gca;
set(ha1,'Position',[.1 .2 .8 .7]);
title('Bode Diagram');
xlabel('Angular Frequency');
ylabel('Gain[dB]');
grid on;

ha2=axes('Position',get(ha1,'Position'),...
   'YAxisLocation','right','Color','none',...
   'XColor','k','YColor','k','XScale','log','YLim',[-180,0]);
hs2=line(x,z,'Color','r','Parent',ha2);
ylimits=get(ha2,'YLim');
yinc=(ylimits(2)-ylimits(1))/6;
set(ha2,'YTick',[ylimits(1):yinc:ylimits(2)]);
ylabel('Phase[degree]');
figure(1);
% end of program
```

5.3　GUI オブジェクト

グラフィカル・インターフェース (GUI) を定義する関数には，次のような関数があります．

- uicontrol のプロパティー Style で指定する，Frame, Text, Edit, Slider, Popupmenu, Listbox, Radiobutton, Checkbox, Pushbutton

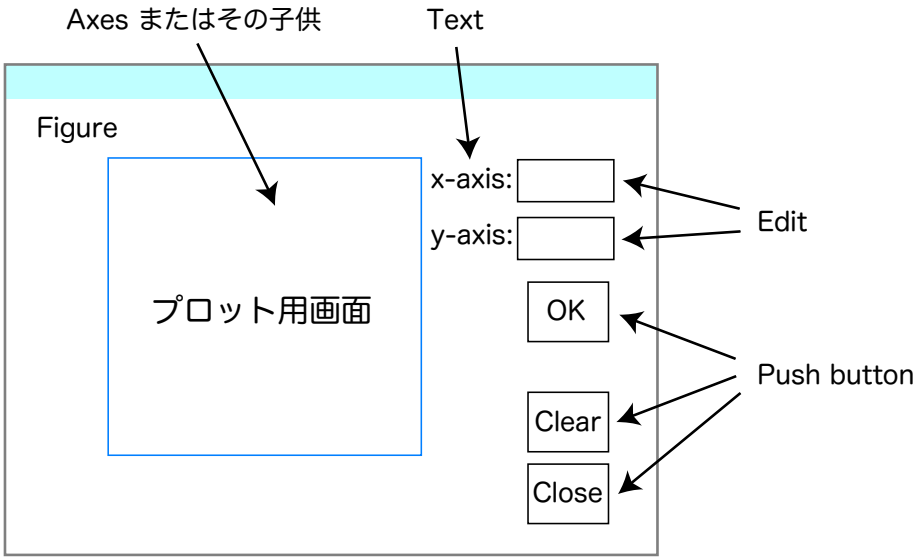


図 5.3　例 5.4 用のウインドウの設計例.

- uimenu
- errordlg, helpdlg, questdlg, warndlg, listdlg, msgbox, choices
- uigetfile, uiputfile, printdlg, uisetfont, uisetcolor
- miscellaneous: gtest, ginput, rbbox, waitforbuttonpress

一般に，プログラムの筋書きは次のようになります.

- これらの小物を画面に定義し，
- それらのコールバック・ルーティーンでイベントの処理をする

できるだけ，簡単な例を一つやってみましょう.

【例 5.5】

ウインドウにプロット用の座標平面を描き，マウスでクリックした点をマークし，その座標を小窓に表示するプログラムをつくりなさい．また，逆に表示用小窓に値を入力すると，その点を作った座標平面に点として表示するようにもしておきなさい.

【解説】 まず，つくるウインドウのおおざっぱなデザインをしておきます．これは，できた設計図をどうプログラムで実現するかとは一応無関係です.

そこで，上の問題の場合，図 5.3 に示したようなウインドウを作ることにしました．レイアウトが決まるとそれぞれの小物を何で実現するかが決まります．図には，これらを一緒に書き込んであります．それぞれの小物の役割は次の通りです.

- プロット用画面をクリックすると，その点にマークを入れ，左上の Edit の窓に座標を表示する.
- 逆に，左上の Edit の窓に座標を入力し，OK のボタンを押すと，プロット用画面に座標が表示される.

- Clear ボタンは，プロット用画面のそれまでに入力された点を消去する．Close はプログラムの終了．

さて，いよいよプログラムづくりです．一気に全部作るのも結構ですが，とりあえず一番大切なプロット用画面と下にある 2 つのボタンを作って，作動するプログラムをつくりましょう．その一例を示します．

```
function ShowPts
% First trial for Example 5.4
% This program has a callback routine:
% GetAndDrawE which is stored in
% separate file. Nov. 1, '98, HK
global xm ym hndl1
xm=[-3,3]; ym=[-3,3]; %Range of Coordinate
InitGUI; % Graphics initialization
InitG; % Graphics initialization
set(gca,'ButtonDownFcn','GetAndDrawE');
%end of main

% Graphics initialization
function InitGUI
global xm ym hndl1
hf=figure('Units','Normalized','Position',[.2 .2  .7 .7]);
set(hf,'DefaultUicontrolUnits','Normalized');
set(hf, 'Pointer','crosshair');
hupb1=uicontrol(hf,'Style','Pushbutton','String','Close',...
   'Value',0,'Position',[0.775,0.15,0.1,0.1],...
   'Callback','closereq');
hupb2=uicontrol(hf,'Style','Pushbutton','String','Clear',...
   'Value',0,'Position',[0.775,0.3,0.1,0.1],...
   'Callback','delete(line)');

% Graphics initialization
function InitG
global xm ym hndl1
hndl1=line('color','r','marker','.','markersize',20,...
   'erase','none','xdata',[],'ydata',[]);
set(gca,'Units','Normalized','Position',[.05 .2 .7 .7]);
axis([xm(1) xm(2) ym(1) ym(2)]);
box on;
title('Rectangular Coordinate');
xlabel('x-axis');
ylabel('y-axis');
grid on;
axis square;
figure(1);
% end of program
```

このプログラムでは，マウスのボタンが押されたときに呼び出す Callback ルーティーンとして，次のプログラムを関数として別に書いておきます．

```
function GetAndDrawE
% This is a callback routine for Example 5.4
```

```
global xm ym hndl1
a=get(gca,'CurrentPoint');
x=a(1,1);
y=a(1,2);
set(hndl1,'xdata',x,'ydata',y);
drawnow;
```

ここで，簡単に GUI プログラムに関して割り込みのスタイルをみておきましょう．プログラミング・スタイルとしては，次の 3 種類が考えられます．

1. callback routine を外部関数として用意するスタイル．この場合，プログラムの実行状態は，最初の実行で main プログラムは終了し，割り込みの都度 callback routine が実行されることになります．main は終了してしまっているので，subfunction は動作しません．したがって，callback routine は外部関数として定義しておかなければなりません．このため，file の数が多くなって煩雑ですがプログラム自体は簡潔でロバストな動作を期待できます．
2. main を関数にして，引数によって callback routine を実行するようにプログラムします．この場合，main プログラムは default を 1 度実行して終了しますが，callback routine が同じ関数の中に定義されているので，プログラムは一つ，すなわち file は一つ，で済みます．subfunction が動作しますから，長いプログラムになるときは，適当に subfunction を定義して簡潔なプログラムにすることができます．実際，この手法は demo プログラムで広く利用されています．
3. while 文で無限ループを作り，プログラムは実行状態にしておくスタイル．この場合もプログラムは未終了なので，subfunction の記述が許されます．したがって上の 2 番目のものと同様 file は 1 つで済みます．

目的に応じて，適宜スタイルを考えるといいでしょう．上に示したプログラムの例は，(1) のスタイルのものです．ついでに，(2) と (3) のスタイルのものを示しておきましょう．

```
function ShowPts(arg)
% Second trial for Example 5.4
% This program has a self-callback routine.
% The second style is applied. Nov. 9, '98 HiK.
global xm ym hndl1

if nargin==0
   init_all;
else
   show_point;
end
%end of main

function init_all
global xm ym hndl1
xm=[-3,3]; ym=[-3,3]; %Range of Coordinate
InitGUI; % Graphics initialization
InitG; % Graphics initialization
set(gca,'ButtonDownFcn','ShowPts arg');

function show_point
```

```
global xm ym hndl1
a=get(gca,'CurrentPoint');
x=a(1,1);
y=a(1,2);
set(hndl1,'xdata',x,'ydata',y);
drawnow;

% GUI initialization
% Graphics initialization
function InitGUI
global xm ym hndl1
hf=figure('Units','Normalized','Position',[.2 .2  .7 .7]);
set(hf,'DefaultUicontrolUnits','Normalized');
set(hf, 'Pointer','crosshair');
hupb1=uicontrol(hf,'Style','Pushbutton','String','Close',...
   'Value',0,'Position',[0.775,0.15,0.1,0.1],...
   'Callback','closereq');
hupb2=uicontrol(hf,'Style','Pushbutton','String','Clear',...
   'Value',0,'Position',[0.775,0.3,0.1,0.1],...
   'Callback','delete(line)');

function InitG
global xm ym hndl1
hndl1=line('color','r','marker','.','markersize',20,...
   'erase','none','xdata',[],'ydata',[]);
set(gca,'Units','Normalized','Position',[.05 .2 .7 .7]);
axis([xm(1) xm(2) ym(1) ym(2)]);
box on;
title('Rectangular Coordinate');
xlabel('x-axis');
ylabel('y-axis');
grid on;
axis square;
figure(1);
```

ここからは，3 番目のスタイルのプログラム例です．何かアニメーションなどしたい場合は，無限ループの中に組み込むといいでしょう．

```
function ShowPts
% Third trial for Example 5.4
% This program has an endless while statement.
% The third style is applied. Nov. 9, '98 HiK.
global xm ym hndl1 bu x y

xm=[-3,3]; ym=[-3,3]; bu=-1;
x=0;y=0;

InitGUI;
InitG;
set(gca,'ButtonDownFcn','bdf')
set(gca,'Userdata',1);

while get(gca,'Userdata')==1,
   if bu==1,
```

```
      gcpnt;
   end;
   drawnow;
end
clear global xm ym hndl1 x y;
closereq;
%end of main

% GUI initialization
function InitGUI
hf=figure('Units','Normalized','Position',[.2 .2  .7 .7]);
set(hf,'DefaultUicontrolUnits','Normalized');
set(hf, 'Pointer','crosshair');
hupb1=uicontrol(hf,'Style','Pushbutton','String','Close',...
   'Value',0,'Position',[0.775,0.15,0.1,0.1],...
   'Callback','set(gca,''Userdata'',-1)');
hupb2=uicontrol(hf,'Style','Pushbutton','String','Clear',...
   'Value',0,'Position',[0.775,0.3,0.1,0.1],...
   'Callback','delete(line)');

% Graphics initialization
function InitG
global xm ym hndl1
set(gca,'Units','Normalized','Position',[.05 .2 .7 .7]);
axis([xm(1) xm(2) ym(1) ym(2)]);
hndl1=line('color','r','marker','.','markersize',20,...
   'erase','none','xdata',[],'ydata',[]);
box on;
title('Rectangular Coordinate');
xlabel('x-axis');
ylabel('y-axis');
grid on;
axis square;
figure(1);

function gcpnt
global hndl1 bu x y
a=get(gca,'CurrentPoint');
x=a(1,1);
y=a(1,2);
set(hndl1,'xdata',x,'ydata',y);
bu=-1;

function bdf
global bu
bu=1;
```

さて，脇道にそれてしまったようですが，元の問題にかえりましょう．最後に，座標を表示する Edit を付けて完成させます．一応動作する最低限のプログラムができました．

```
function PlotPoint(arg)
% Main program for Example 5.4
% This program has two self-callback routines:
% Point_Draw and Put_Point,  Nov. 4, 1998.
```

```
% File Name: PlotPoint.m
global xm ym huex huey hndl1 hndl2

if nargin==0
   xm=[-3,3]; ym=[-3,3];
   InitGUI; % GUI initialization
   InitG; % Graphics initialization
   set(gca,'ButtonDownFcn','PlotPoint PDraw');
elseif arg=='PDraw'
   Point_Draw;
elseif aeg=='PPoint'
   Put_Point
end
%end of main

function Point_Draw
% This is a callback routine for ButtonDownFcn
global xm ym huex huey hndl1 hndl2
a=get(gca,'CurrentPoint');
x=a(1,1); y=a(1,2);
cx=num2str(x); cy=num2str(y);
set(hndl1,'xdata',x,'ydata',y);
set(huex,'String',cx); set(huey,'String',cy);
drawnow;

function Put_Point
% This is a callback routine for PushButton:OK
global xm ym huex huey hndl1 hndl2
fx=get(huex,'String');
x=str2num(fx);
if (x<xm(1)),
   x=xm(1);
elseif (x>xm(2))
   x=xm(2);
end;
fy=get(huey,'String');
y=str2num(fy);
if (y<ym(1)),
   y=ym(1);
elseif (y>ym(2))
   y=ym(2);
end;
set(hndl2,'xdata',x,'ydata',y);
drawnow;

% GUI Initialization
function InitGUI
global xm ym huex huey hndl1 hndl2
hf=figure('Units','Normalized','Position',[.2 .2  .7 .7]);
set(hf,'DefaultUicontrolUnits','Normalized');
set(hf, 'Pointer','crosshair');
hupb1=uicontrol(hf,'Style','Pushbutton','String','Close',...
   'Value',0,'Position',[0.775,0.15,0.1,0.1],...
   'Callback','closereq');
```

```
hupb2=uicontrol(hf,'Style','Pushbutton','String','Clear',...
   'Value',0,'Position',[0.775,0.3,0.1,0.1],...
   'Callback','delete(line)');
hupb3=uicontrol(hf,'Style','Pushbutton','String','OK',...
   'Value',0,'Position',[0.775,0.55,0.1,0.1],...
   'Callback','PutDraw');
huex=uicontrol(hf,'Style','Edit','String','',...
   'Position',[.75 .8 .15 .05],'HorizontalAlignment','Left');
huey=uicontrol(hf,'Style','Edit','String','',...
   'Position',[.75 .7 .15 .05],'HorizontalAlignment','Left');
hutx=uicontrol(hf,'Style','Text','String','x-axis:',...
   'Position',[.68 .8 .06 .05],'HorizontalAlignment','Center');
huty=uicontrol(hf,'Style','Text','String','y-axis:',...
   'Position',[.68 .7 .06 .05],'HorizontalAlignment','Center');

% Graphics Initialization
function InitG
global xm ym huex huey hndl1 hndl2
hndl1=line('color','r','marker','.','markersize',20,...
   'erase','none','xdata',[],'ydata',[]);
hndl2=line('color','g','marker','.','markersize',20,...
   'erase','none','xdata',[],'ydata',[]);
set(gca,'Units','Normalized','Position',[.05 .2 .7 .7]);
axis([xm(1) xm(2) ym(1) ym(2)]);
box on;
title('Rectangular Coordinate');
xlabel('x-axis'); ylabel('y-axis');
grid on; axis square;
figure(1);
```

5.4　GUI を付加したプログラミング

さて，これまでのネタを使ってプログラムすることを考えましょう．プログラムの筋書きは，ほぼ次のように書くといいでしょう．

```
function Event_driven_like_program
% Example of programming style
% main part
global  %definition of global variables
CreateWindows;
ComputeAndDraw;
CloseWindows;
%----------------------------------
function CreateWindows;
%  Creation and initialization of graphics objects
function CloseWindows;
%  Termination of graphics objects
%----------------------------------
function ComputeAndDraw
% main working part of this program
global  %definition of global variables
while 1,
```

```
  EventLoop;
  % here put the main computation
end;
function EventLoop;
function HitButton(event);
function HitKey(event);
%-----------------------------------
%define other functions needed for this program
% end of program
```

では，簡単な例として，4.5.3 で作ったプログラムを少し使いやすくしてみましょう．

【例 5.6】

van der Pol 方程式の相平面図をみる 4.5.3 のプログラムにプッシュボタンを 2 つ付けて，1 つは描いた軌道を消去するボタン，他の 1 つは計算終了のボタンにしなさい．また，マウスで相平面内の点をクリックすると，その点を初期値として改めて軌道を描くようにしておきなさい．

【解説】 プログラム一例を示します．

```
function PhPt
% main program
global hndl1 hndl2 hndl3 x X bu L

t=0; tmax=20*pi; h=0.1; bu=-1; L=20;
x=[0.1; 0.0]; X=x*ones(1, L);

InitGUI;
InitG(x);

set(gca,'ButtonDownFcn','bdf')
set(gca,'Userdata',1);

while get(gca,'Userdata')==1,
  if bu==1,
      gcpnt
   end;
   [t, x]=RK(t,x,h);
   X=[x X(:,1: L-1)];
   set(hndl1,'xdata',X(1,1),'ydata',X(2,1));
   set(hndl2,'xdata',X(1,1:L-1),'ydata',X(2,1:L-1));
   set(hndl3,'xdata',X(1,L-1:L),'ydata',X(2,L-1:L));
   drawnow;
end
%close(gcf);
closereq;
%end of main

% van der Pol equation
function dx=vP(t,x)
dx=[x(2); 0.5*(1.0-x(1)*x(1))*x(2)-x(1)];

% Runge Kutta method
function [t, x]=RK(t,x,h)
```

```
   f1=vP(t,x);
   f2=vP(t+h/2,x+h*f1/2);
   f3=vP(t+h/2,x+h*f2/2);
   f4=vP(t+h,x+h*f3);
   t=t+h;
   x=x+h*(f1+2*(f2+f3)+f4)/6;

% Graphics initialization
function InitGUI
  set(gcf,'DefaultUicontrolUnits','Normalized');
  set(gcf, 'Pointer','crosshair');
  hd1=uicontrol(gcf,'Style','Pushbutton','String','Close',...
     'Value',0,'Position',[0.85,0.15,0.1,0.1],...
     'Callback','set(gca,''Userdata'',-1)');
 % set(hd1, 'BackgroundColor',[.8 .8 .8]);
  hd2=uicontrol(gcf,'Style','Pushbutton','String','Cls',...
     'Value',0,'Position',[0.85,0.3,0.1,0.1],...
     'Callback','delete(line)');
 % set(hd2, 'BackgroundColor',[.1 .1 .1]);

% Graphics initialization
function InitG(x)
global hndl1 hndl2 hndl3
  hndl1=line('color','r','marker','.','markersize',25,...
     'erase','xor','xdata',x(1),'ydata',x(2));
  hndl2=line('color','r','linestyle','-','linewidth',2,...
     'erase','none','xdata',[],'ydata',[]);
  hndl3=line('color','b','linestyle','-','linewidth',2,...
     'erase','none','xdata',[],'ydata',[]);
  axis([-3 3 -3 3]);
  box on;
  title('Phase Portrait');
  xlabel('x(t)');
  ylabel('x-dot');
  grid on;
  axis square;
  figure(1);

function gcpnt
global hndl1 hndl2 hndl3 x X bu L
a=get(gca,'CurrentPoint');
x=[a(1,1); a(1,2)];
X=x*ones(1,L);
bu=-1;

function bdf
global hndl x X bu
bu=1;
% end of program
```

5.5　これで MATLAB プログラミングは修了しました

一応予定した話題はこれでおわりです．実は，MATLAB には

```
》guide
```

という，強力な GUI ツールがあって，これを使えば「小物」はたちどころにできてしまします．もちろん，みなさんの MATLAB で動きます．Property Editor や Callback Editor は，属性を変更するのにとても便利にできています．Visual Basic を使っているみたいなのではないでしょうか．

GUI に関しては，human interface という意味でデザインが大変です．Simplicity, Consistency and Familiarity などを心がけるようとのことです．demo プログラムなどにみられる良いデザインをまねることから始めるといいでしょう．

5.6　演習問題

1. 例 5.2 を参考にし，プッシュ・ボタンを追加して，sin, cos, tan のグラフを描くプログラムを作ってください．
2. タートル・グラフィックスで図形を描く途中をアニメとして表示するプログラムを作ってください．
3. 窓を一つ開いて，時計を作ってください．
4. 窓を一つ開いて，万年カレンダーを作ってください．
5. 前章の練習問題 5 のプログラムに，次のボタンをつけ加えた GUI プログラムを作ってください．
   - 画面を消去するボタンと計算を中止するボタンを付ける．
   - 画面をマウスでクリックすると，その点を初期値として計算を再開する．
   - $\Omega/2\pi$ 毎に軌道にマークを入れる．すなわち，専用ボタンを 1 つ用意して，軌道とマークを描き分けるようにする．
6. 次のプログラムは 3D 画面を回転させるプログラムです．その仕組みを解析し，おもしろいツールを考えてください．

```
function az(button)
global AZ_PNT
if nargin==0
   button='on';
end
if strcmp(button,'on')
   set(gcf,'WindowButtonDownFcn','az down');
   set(gcf,'WindowButtonMotionFcn','az motion');
   set(gcf,'WindowButtonUpFcn','az up');
   AZ_PNT=[];
elseif strcmp(button,'off')
   set(gcf,'WindowButtonDownFcn','');
   set(gcf,'WindowButtonMotionFcn','');
   set(gcf,'WindowButtonUpFcn','');
   clear global AZ_PNT
elseif strcmp(button,'down')
```

```
   AZ_PNT=get(gcf,'CurrentPoint');
elseif strcmp(button,'motion')
   if isempty(AZ_PNT)==0
      new_point=get(gcf,'CurrentPoint');
      x=[AZ_PNT(1),new_point(1)];
      y=[AZ_PNT(2),new_point(2)];
      [azimuth,zenith]=view;
      position=get(gcf,'Position');
      AZ_WINDOW=position(3:4);
      azimuth=azimuth-360.*(x(2)-x(1))/AZ_WINDOW(1);
      zenith=zenith-360.*(y(2)-y(1))/AZ_WINDOW(2);
      view([rem(azimuth,360),rem(zenith,360)])
      AZ_PNT=get(gcf,'CurrentPoint');
   end
elseif strcmp(button,'up')
   AZ_PNT=[];
end
```

第 III 部

MATLAB の応用あれこれ

第 6 章

周期解の分岐計算：非自律系の場合

この章の内容は，1987 年（昭和 62 年）3 月，研究室の技術報告としてつくったノートを書き直したものである。当時の報告に付けたプログラムは Fortran で書かれていた。今回は，このプログラムの部分だけを MatLab で書き直すことにしよう。

6.1 はじめに

Duffing 方程式：

$$\frac{d^2x}{dt^2} + k\frac{dx}{dt} + c_1 x + c_3 x^3 = B_0 + B\cos(t) \tag{6.1}$$

すなわち，

$$\begin{aligned} \frac{dx}{dt} &= y \\ \frac{dy}{dt} &= -ky - c_1 x - c_3 x^3 + B_0 + B\cos(t) \end{aligned} \tag{6.2}$$

は，5 つのパラメータ $\lambda = (k,\ c_1,\ c_3,\ B_0,\ B)$ を持っている。これらのパラメータの値により，式 (6.1) または式 (6.2) の周期解に種々の分岐が生じる。そこで分岐の生じるパラメータ値を計算するプログラムについて考えよう。

分岐の生じるパラメータ探索には，周期解を与える式と分岐の条件式を Newton 法で同時に解く手法を用いた。この解法は収束も早く，有効な方法であるが，最大の問題点は「最初に近似解 (first guess) をどのように見いだすか」という点である。我々はこの問題に対して原則として次の戦略を用いる。

- singular な問題の解に関する近似解を求めるには，non-singular な問題をまず解き，パラメータを変化させ，この non-singular な問題が解けなくなった点をこの近似解とみなす

このことから，分岐問題を考える場合にも，次の順番に計算を進める。

1. phase portrait により，周期解に対応する Poincaré 写像の安定，あるいは不安定固定点の近似値を求める。
2. この近似値を用いて，Newton 法により固定点を計算する。パラメータを変化させ，固定点が求められなくなる点を見いだす。この点は，特性乗数が分岐の条件を満たす近似値の候補である。

3. 上記の分岐の条件を満たす近似値を用いて，分岐計算を行う。

6.2　計算法の概略

6.2.1　周期解の計算

パラメータを含む次の微分方程式を考えよう：

$$\frac{dx}{dt} = f(t,\ x,\ \lambda) \tag{6.3}$$

ここに $x \in R^n$ は状態，$\lambda \in R$ は系のパラメータとする[*1]. また，写像 $f : R \times R^n \times R \to R^n$ は，なめらかな写像とし，時刻 t に関して周期的であり，その周期を 2π とする。初期条件 $x(t_0) = x_0$ （簡単のため $t_0 = 0$ とする）を満たす式 (6.3) の解を

$$x(t) = \phi(t, x_0, \lambda) \tag{6.4}$$

とする。Poincaré map T を

$$T : R^n \to R^n; \quad x \mapsto \phi\left(2\pi, x_0, \lambda\right) \tag{6.5}$$

と定義すれば，式 (6.3) の周期 $2\pi m$ の周期解は，T の $m-$ 周期点に対応する。これを求めるには

$$F(x) = \phi(2\pi m, x, \lambda) - x = 0 \tag{6.6}$$

を解けばよい。そこで，式 (6.6) を Newton 法によって解くことにしよう。このための繰り返しアルゴリズムは次のとおりである。

1. 近似解 $x = x^{(0)}$ を与える。
2. 近似解 $x = x^{(n+1)}$ を $x^{(n)}$ より次式を解いて求める。

$$\begin{array}{rcl} x^{(n+1)} & = & x^{(n)} + h \\ DF(x^{(n)})h & + & F(x^{(n)}) = 0 \end{array} \tag{6.7}$$

なお，DF の計算には式 (6.3) の変分方程式の解を用いるとよい。

6.2.2　分岐パラメータの計算

式 (6.3) の周期解の分岐を考えよう。T の固定点の分岐について述べる。周期点についても同様である。点 x を T の固定点とする：

$$T(x) - x = 0 \tag{6.8}$$

[*1] パラメータはスカラーと仮定した。実際，数値計算では残りのパラメータを固定し一つだけパラメータを変化させる場合が多いからである。このことによって一般性は失われない。

表 6.1　Duffing 方程式の変分方程式

変数	計算機変数	微分方程式	初期条件
$x = \phi_1(t, x_0, y_0, \lambda)$	x_1	$\dot{x}_1 = x_2$	x_0
$y = \phi_2(t, x_0, y_0, \lambda)$	x_2	$\dot{x}_2 = -kx_2 - c_1x_1 - c_3x^3 + B_0 + B\cos(t)$	y_0
$\partial\phi_1/\partial x_0$	x_3	$\dot{x}_3 = x_4$	1
$\partial\phi_2/\partial x_0$	x_4	$\dot{x}_4 = Px_3 - kx_4$	0
$\partial\phi_1/\partial y_0$	x_5	$\dot{x}_5 = x_6$	0
$\partial\phi_2/\partial y_0$	x_6	$\dot{x}_6 = Px_5 - kx_6$	1
$\partial\phi_1/\partial B$	x_7	$\dot{x}_7 = x_8$	0
$\partial\phi_2/\partial B$	x_8	$\dot{x}_8 = Px_7 - kx_8 + \cos(t)$	0
$\partial^2\phi_1/\partial x_0^2$	x_9	$\dot{x}_9 = x_{10}$	0
$\partial^2\phi_2/\partial x_0^2$	x_{10}	$\dot{x}_{10} = Px_9 - kx_{10} + Qx_3^2$	0
$\partial^2\phi_1/\partial x_0\partial y_0$	x_{11}	$\dot{x}_{11} = x_{12}$	0
$\partial^2\phi_2/\partial x_0\partial y_0$	x_{12}	$\dot{x}_{12} = Px_{11} - kx_{12} + Qx_3x_5$	0
$\partial^2\phi_1/\partial y_0^2$	x_{13}	$\dot{x}_{13} = x_{14}$	0
$\partial^2\phi_2/\partial y_0^2$	x_{14}	$\dot{x}_{14} = Px_{13} - kx_{14} + Qx_5^2$	0
$\partial^2\phi_1/\partial x_0\partial B$	x_{15}	$\dot{x}_{15} = x_{16}$	0
$\partial^2\phi_2/\partial x_0\partial B$	x_{16}	$\dot{x}_{16} = Px_{15} - kx_{16} + Qx_3x_7$	0
$\partial^2\phi_1/\partial y_0\partial B$	x_{17}	$\dot{x}_{17} = x_{18}$	0
$\partial^2\phi_2/\partial y_0\partial B$	x_{18}	$\dot{x}_{18} = Px_{17} - kx_{18} + Qx_3x_7$	0

where $P = -c_1 - c_3x_1^3$ and $Q = -6c_3x_1$. The parameter λ is selected as B.

特性方程式は次式となる。

$$g(\mu) = \det(DT(x) - \mu I_n) = 0 \tag{6.9}$$

これより，

1. 接線分岐 (tangent bifurcation) は，式 (6.9) の一つの根が $\mu = 1$ となるパラメータで生じる。したがって，式 (6.8) と $g(1) = 0$ を (x, λ) に関して Newton 法で解けばよい。
2. 周期倍分岐 (period doubling bifurcation) は，式 (6.9) の一つの根が $\mu = -1$ となるパラメータで生じる。したがって，式 (6.8) と $g(-1) = 0$ を (x, λ) に関して Newton 法で解けばよい。

これらの連立方程式を解く際に現れるヤコビ行列の計算には，式 (6.3) の初期値に関する第一変分方程式，第二変分方程式，およびパラメータに関する第一変分方程式を用いるとよい。式 (6.2) に関するこれらの方程式と必要な初期値を表 6.1 に示した。

6.3　Phase Portrait をみる：DemoPL

Phase Portrait をみるプログラムは，使い勝手がいいように色々な要求のあるプログラムです。簡潔さも大切です。それに，1987 年当時とコンピュータ環境は様変わりしてしまっています。当時はキー割り込みで画面を制御することが主流でしたが，今は GUI でマウスと画面のボタンが制御の本流です。とは言っても，キー操作でパラメータを変化させるなどキー割り込みも捨てがたい魅力を持っています。そんなわけで，次に示すプログラムではキーの割り込みを使うことにしました。

```
function DemoPL
% main program
global hndl1 hndl2 huex huey x X bu hf B DB B0 DB0 k dk ifl
global huek hueB hueB0

B=0.3; DB=0.01; B0=0.0; DB0=0.01; k=0.1; dk=0.01;ifl=1;
bu=-1; m=64; h=2.0*pi/m;
x=[0.0; 0.0]; X=[x x];

InitGUI;
InitG(x);

set(gca,'ButtonDownFcn','bdf')
set(gca,'Userdata',1);
set(hf,'KeyPressFcn','KeyInt');

while get(gca,'Userdata')==1,
   set(hndl1,'xdata',X(1,1),'ydata',X(2,1));
   cx=num2str(X(1,1)); cy=num2str(X(2,1));
   set(huex,'String',cx); set(huey,'String',cy);
   ck=num2str(k); cB=num2str(B); cB0=num2str(B0);
   set(huek,'String',ck); set(hueB,'String',cB); set(hueB0, 'String',cB0);
   drawnow;
   for j=0:m-1
         if bu==1,
            gcpnt;
            break;
```

```
          end;
       t=j*h;
       [t, x]=RK(t,x,h);
      X=[x X(:,1)];
      if ifl==1,
         set(hndl2,'xdata',X(1,1:2),'ydata',X(2,1:2));
      end;
   end
end
closereq;
%end of main

% Duffing's equation
function dy=Duf(t,y)
global B B0 k
dy=[y(2); -k*y(2)-y(1)*y(1)*y(1)+B0+B*cos(t)];

% Runge Kutta method
function [t, z]=RK(t,z,h)
   f1=Duf(t,z);
   f2=Duf(t+h/2,z+h*f1/2);
   f3=Duf(t+h/2,z+h*f2/2);
   f4=Duf(t+h,z+h*f3);
   t=t+h;
   z=z+h*(f1+2*(f2+f3)+f4)/6;

% Graphics initialization
function InitGUI
global hndl1 hndl2 huex huey hf
global huek hueB hueB0
hf=figure('Units','Normalized','Position',[.1 .2 .7 .7]);
set(hf,'DefaultUicontrolUnits','Normalized');
set(hf, 'Pointer','crosshair');
hd1=uicontrol(hf,'Style','Pushbutton','String','Close','Value',0,...
     'Position',[0.8,0.1,0.1,0.1],'Callback','set(gca,''Userdata'',-1)');
hd2=uicontrol(hf,'Style','Pushbutton','String','Cls','Value',0,...
     'Position',[0.8,0.22,0.1,0.1],'Callback','delete(line)');

hutx=uicontrol(hf,'Style','Text','String','x-axis:',...
   'Position',[.8 .87 .08 .04],'HorizontalAlignment','Center');
huex=uicontrol(hf,'Style','Edit','String','',...
   'Position',[.8 .84 .1 .05],'HorizontalAlignment','Left');

huty=uicontrol(hf,'Style','Text','String','y-axis:',...
   'Position',[.8 .77 .08 .04],'HorizontalAlignment','Center');
huey=uicontrol(hf,'Style','Edit','String','',...
   'Position',[.8 .74 .1 .05],'HorizontalAlignment','Left');

hutk=uicontrol(hf,'Style','Text','String','k:',...
   'Position',[.8 .63 .08 .04],'HorizontalAlignment','Center');
huek=uicontrol(hf,'Style','Edit','String','',...
   'Position',[.8 .6 .1 .05],'HorizontalAlignment','Left');

hutB=uicontrol(hf,'Style','Text','String','B:',...
```

```
   'Position',[.8 .53 .08 .04],'HorizontalAlignment','Center');
hueB=uicontrol(hf,'Style','Edit','String','',...
   'Position',[.8 .5 .1 .05],'HorizontalAlignment','Left');

hutB0=uicontrol(hf,'Style','Text','String','B0:',...
   'Position',[.8 .43 .08 .04],'HorizontalAlignment','Center');
hueB0=uicontrol(hf,'Style','Edit','String','',...
   'Position',[.8 .4 .1 .05],'HorizontalAlignment','Left');

% Graphics initialization
function InitG(x)
global hndl1 hndl2 huex huey
  hndl1=line('color','r','marker','.','markersize',20,'erase','none',...
   'xdata',[],'ydata',[]);
  hndl2=line('color','b','linestyle','-','linewidth',1,'erase','none',...
   'xdata',[],'ydata',[]);
  set(gca,'Position',[0.02 0.1 .8 .8]);
  axis([-2 2 -2 2]);
  box on;
  title('Phase Portrait');
  xlabel('x(t)');
  ylabel('x-dot');
  grid on;
  axis square;
  figure(1);

  function gcpnt
  global x X bu
  a=get(gca,'CurrentPoint');
  x=[a(1,1); a(1,2)];
  X=[x x];
  bu=-1;

  function bdf
  global bu
  bu=1;

% Key interrupt function
function KeyInt;
global hf B DB B0 DB0 k dk ifl
a=get(hf,'CurrentCharacter');
switch a
    case 'D'
        ifl=1-ifl;
    case 'b'
        B=B-DB;
    case 'B'
        B=B+DB;
    case 'c'
        B0=B0-DB0;
    case 'C'
        B0=B0+DB0;
    case 'k'
        k=k-dk;
```

```
        case 'K'
            k=k+dk;
        otherwise
            %disp('other key is pressed');
    end
```

6.4　固定点・周期点を求める：DufFix

周期解を求め，その安定性を吟味することは非線形システムの解析で最も基本的な事柄です。プログラムの一例を示しておこう。

```
function DufFix
% main program
global B B0 k
global IterMax ErrMax epsx
B=0.3; DB=0.01; B0=0.0; DB0=0.01; k=0.1; dk=0.01;
L=1; m=128;
x=[-0.32; 0.04; 1.0; 0.0; 0.0; 1.0];
IterMax=10; ErrMax=100.0; epsx=0.00001;

while 1,
    [s_ind, iterN, x, chara]=Fixed(L, m, x);
    switch s_ind
        case 0
            % convergence
            disp([s_ind iterN B B0 x(1) x(2) chara]);
        case 1
            % divergence
            disp([s_ind 'States are divergent']);
            break;
        case 2
            % so many iterations
            disp([s_ind 'So many iterations in the Newton method']);
            break;
        otherwise
            % impossible
            disp(['This case never occurs']);
    end
    B0=B0+DB0;
end
% end of main program

% Find fixed point or L-periodic point
function [s_ind, iterN, x, chara]=Fixed(L,m,x)
global IterMax ErrMax epsx
s_ind=0; dist=1.0; iterN=1; p=zeros(2,1); chara=[];
while dist>epsx,
    iterN=iterN+1;
    p(1)=x(1); p(2)=x(2);
    x=NewtonR(L,m,x);
    ddx=x(1)-p(1); ddy=x(2)-p(2);
    dist=(abs(x(1)-p(1))+abs(x(2)-p(2)))/2;
```

```
    if dist>ErrMax
        s_ind=1;% divergence
        break;
    end
    if iterN>IterMax
        s_ind=2; % sorry iterating too much
        break;
    end
end
% characteristic multipliers
[V, W]=eig([x(3) x(4); x(5) x(6)]);
chara=[W(1,1) W(2,2)];

% Newton's iteration
function x=NewtonR(L,m,x)
p=x(1:2); A=zeros(2,2);
x=sysvar(L,m,x);
A(1,1)=x(3)-1.0;
A(1,2)=x(5);
A(2,1)=x(4);
A(2,2)=x(6)-1.0;
x(1:2)=p+A\[p(1)-x(1); p(2)-x(2)];

% Poincare map of states
function x=sysvar(L,m,x)
x(3)=1.0; x(4)=0.0; x(5)=0.0; x(6)=1.0;
h=2.0*pi/m;
for kkk=1:L
     for j=0:m-1
       t=j*h;
       [t, x]=RK(t,x,h);
     end
end

% Runge Kutta method
function [t, z]=RK(t,z,h)
   f1=Duf(t,z);
   f2=Duf(t+h/2,z+h*f1/2);
   f3=Duf(t+h/2,z+h*f2/2);
   f4=Duf(t+h,z+h*f3);
   t=t+h;
   z=z+h*(f1+2*(f2+f3)+f4)/6;

% Duffing's equation
function dy=Duf(t,y)
global B B0 k
dy=zeros(6,1); c1=0.0; c3=1.0;
P= -c1-3.0*c3*y(1)*y(1) ;
dy(1)=y(2);
dy(2)=-k*y(2)-y(1)*y(1)*y(1)+B0+B*cos(t);
dy(3)=y(4);
dy(4)=P*y(3)-k*y(4);
dy(5)=y(6);
dy(6)=P*y(5)-k*y(6);
```

6.5　分岐パラメータの計算：DemoBF

周期解が分岐するパラメータを追跡するプログラムをつくる。プログラムの構造は DufFix と同じです。解く方程式の数が増え，ヤコビ行列の計算のための微分方程式の数も増えています。

```
function DufBf
% main program
global B B0 k
global IterMax ErrMax epsx

L=1; m=128; IterMax=20; ErrMax=100.0; epsx=0.00001;

% A0 initial guess
%B=0.46; DB=0.01; B0=0.0; DB0=0.01; k=0.2; dk=0.01;
%x=[-0.654; 0.224; 1.0; 0.0; 0.0; 1.0;0 ; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0];
%mu=1.0;

B=3.0; DB=0.01; B0=0.4988; DB0=0.01; k=0.2; dk=0.01;
x=[1.9632; -2.42; 1.0; 0.0; 0.0; 1.0;0 ; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0];
mu=-1.0;

while 1,
    [s_ind, iterN, x, chara]=Fixed(L, m, x, mu);
    switch s_ind
        case 0
            % convergence
            disp([iterN B B0 x(1) x(2) chara]);
        case 1
            % divergence
            disp([s_ind 'States are divergent']);
            break;
        case 2
            % so many iterations
            disp([iterN 'So many iterations in the Newton method']);
            break;
        otherwise
            % impossible
            disp(['This case never occurs']);
    end
    B0=B0+DB0;
end
% end of main program

% Find fixed point or L-periodic point
function [s_ind, iterN, x, chara]=Fixed(L,m,x, mu)
global B B0 k
global IterMax ErrMax epsx
s_ind=0; dist=1.0; iterN=1; p=zeros(2,1); chara=[];
while dist>epsx,
    iterN=iterN+1;
    p(1)=x(1); p(2)=x(2); p(3)=B;
    x=NewtonR(L,m,x,mu);
```

```
    ddx=x(1)-p(1); ddy=x(2)-p(2);ddz=(B-p(3))/B;
    dist=(abs(x(1)-p(1))+abs(x(2)-p(2))+abs(ddz))/3;
    if dist>ErrMax
        s_ind=1;% divergence
        break;
    end
    if iterN>IterMax
        s_ind=2; % sorry iterating too much
        break;
    end
end
% characteristic multipliers
[V, W]=eig([x(3) x(4); x(5) x(6)]);
chara=[W(1,1) W(2,2)];

% Newton's iteration
function x=NewtonR(L,m,x,mu)
global B B0 k
p=[x(1);x(2);B]; A=zeros(3,3);
x=sysvar(L,m,x);
A(1,1)=x(3)-1.0;
A(1,2)=x(5);
A(1,3)=x(7);
A(2,1)=x(4);
A(2,2)=x(6)-1.0;
A(2,3)=x(8);
A(3,1)=mu*(-x(9)-x(12));
A(3,2)=mu*(-x(11)-x(14));
A(3,3)=x(15)*(x(6)-mu)-x(16)*x(5)+(x(3)-mu)*x(18)-x(4)*x(17);
ppp=p+A\[p(1)-x(1); p(2)-x(2);-mu*mu+mu*(x(3)+x(6))-x(3)*x(6)+x(4)*x(5)];
x(1)=ppp(1); x(2)=ppp(2); B=ppp(3);

% Poincare map of states
function x=sysvar(L,m,x)
x(3)=1.0; x(4)=0.0; x(5)=0.0; x(6)=1.0;
x(7)=0.0; x(8)=0.0; x(9)=0.0; x(10)=0.0;
x(11)=0.0; x(12)=0.0; x(13)=0.0; x(14)=0.0;
x(15)=0.0; x(16)=0.0; x(17)=0.0; x(18)=0.0;

h=2.0*pi/m;
for kkk=1:L
     for j=0:m-1
       t=j*h;
       [t, x]=RK(t,x,h);
     end
end

% Runge Kutta method
function [t, z]=RK(t,z,h)
   f1=Duf(t,z);
   f2=Duf(t+h/2,z+h*f1/2);
   f3=Duf(t+h/2,z+h*f2/2);
   f4=Duf(t+h,z+h*f3);
   t=t+h;
```

```
   z=z+h*(f1+2*(f2+f3)+f4)/6;

% Duffing's equation
function dy=Duf(t,y)
global B B0 k
dy=zeros(6,1); c1=0.0; c3=1.0;
P= -c1-3.0*c3*y(1)*y(1) ;
Q=-6.0*c3*y(1);
dy(1)=y(2);
dy(2)=-k*y(2)-y(1)*y(1)*y(1)+B0+B*cos(t);
dy(3)=y(4);
dy(4)=P*y(3)-k*y(4);
dy(5)=y(6);
dy(6)=P*y(5)-k*y(6);
dy(7)=y(8);
dy(8)=P*y(7)-k*y(8)+cos(t);
dy(9)=y(10);
dy(11)=y(12);
dy(13)=y(14);
dy(15)=y(16);
dy(17)=y(18);
dy(10)=P*y(9)-k*y(10)+Q*y(3)*y(3);
dy(12)=P*y(11)-k*y(12)+Q*y(3)*y(5);
dy(14)=P*y(13)-k*y(14)+Q*y(5)*y(5);
dy(16)=P*y(15)-k*y(16)+Q*y(3)*y(7);
dy(18)=P*y(17)-k*y(18)+Q*y(5)*y(7);
```

第 7 章

周期解の分岐計算：自律系の場合

7.1 はじめに

はじめに，次のなめらかな自律系を考える：

$$\frac{dx}{dt} = f(x,\ \lambda) \tag{7.1}$$

ここに x は状態，λ は系のパラメータとする：

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in R^n, \quad \lambda = \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_m \end{bmatrix} \in R^m \tag{7.2}$$

$t = 0$ で初期値 x_0 を持つ式 (7.1) の解を

$$x\,(t) = \varphi(t,\ x_0),\ x_0 = \varphi(0,\ x_0) \tag{7.3}$$

と書くことにする．

さて，式 (7.1) が周期 L の周期解を持つとし，この周期解に対する局所断面[*1] (local cross section) Π を考え，Poincaré 写像 T を

$$T : \Pi \to \Pi; \quad x \mapsto \varphi\left(L\left(x\right),\ x\right) \tag{7.4}$$

とする．

T の微分[*2] DT がどのような形で計算できるかを考えよう．この公式を用いて，周期解の特性乗数が計算できるようになる．また，周期解そのものを数値計算する際にも役立てることができる．

[*1] 局所断面とは，余次元 1 (codimension 1) の局所的に定義された多様体のことである。

[*2] 写像 T の微分は，色々な記法で表現される。DT はヤコビ行列と考えると $\dfrac{\partial T}{\partial x}$ のようにも書ける。しかし，後者の記法では Π から Π への写像という点があいまいになる。

7.2　局所断面上の Poincaré 写像の性質

局所断面 (cross section) Π がスカラー関数 $g : R^n \to R$ によって

$$g(x) = 0 \tag{7.5}$$

で与える（選んでおく）ものと仮定しよう：

$$\Pi = \{x \in R^n \mid g(x) = 0\} \tag{7.6}$$

Π が局所断面となっていることから

$$\frac{\partial g}{\partial x} f \neq 0 \tag{7.7}$$

である．

今，点 $x_0 \in \Pi$ を初期値とする解が $t = L(x_0)$ に再び Π と交わる点を x_1 とする：

$$x_1 = x(L(x_0)) = \varphi(L(x_0),\, x_0) \tag{7.8}$$

式 (7.7) を x_0 で微分して次式を得る．

$$DT(x_0) = \frac{\partial x_1}{\partial x_0} = \frac{\partial \varphi}{\partial x_0} + \frac{\partial \varphi}{\partial t}\,\frac{\partial L}{\partial x_0} = \frac{\partial \varphi}{\partial x_0} + f\frac{\partial L}{\partial x_0} \tag{7.9}$$

一方，

$$g(x_1) = g(\varphi(L(x_0),\, x_0)) = 0 \tag{7.10}$$

を x_0 で微分して，次の関係式を得る．

$$\frac{\partial g}{\partial x}\left(\frac{\partial \varphi}{\partial x_0} + f\,\frac{\partial L}{\partial x_0}\right) = 0 \tag{7.11}$$

したがって，

$$\frac{\partial L}{\partial x_0} = -\frac{1}{\frac{\partial g}{\partial x} f}\,\frac{\partial g}{\partial x}\,\frac{\partial \varphi}{\partial x_0} \tag{7.12}$$

となる．これを式 (7.9) に代入して，次式を得る．

$$DT(x_0) = \frac{\partial \varphi}{\partial x_0} - \frac{1}{\frac{\partial g}{\partial x} f}\, f\,\frac{\partial g}{\partial x}\,\frac{\partial \varphi}{\partial x_0} = \left[I_n - \frac{1}{\frac{\partial g}{\partial x} f}\, f\frac{\partial g}{\partial x}\right]\,\frac{\partial \varphi}{\partial x_0} \tag{7.13}$$

さて Π の局所座標として $R_{n-1} = \{(u_1, \cdots, u_{n-1})\}$ を

$$x_1 = u_1,\; x_2 = u_2,\; \cdots,\; x_{n-1} = u_{n-1},\; x_n = h(u_1,\, u_2,\, \cdots,\, u_{n-1}) \tag{7.14}$$

によって R_n に埋め込んだとし，この埋め込み写像を ι とする：

$$\iota : R^{n-1} \to R^n; \quad u \mapsto x = \iota(u) = (u_1,\, \cdots,\, u_{n-1},\, h(u_1,\, \cdots,\, u_{n-1})) \tag{7.15}$$

また $\Pi \to R^{n-1}$ は射影:

$$p : R^n \to R^{n-1}; \quad x \mapsto p\,(x) = (x_1,\, \cdots,\, x_{n-1}) \tag{7.16}$$

によって誘導される写像としよう．このとき局所座標 R^{n-1} 上での Poincaré 写像 χ は

$$\chi : R^{n-1} \to R^{n-1}; \quad u \mapsto \chi\,(u) = pT\iota(u) = u_1 \tag{7.17}$$

となる．そこで

$$\begin{aligned} D\chi\,(u_0) &= Dp\,(x_1)\,DT\,(x_0)\,\iota\,(u_0) = [I_{n-1}\vdots 0]DT(x_0)\begin{bmatrix} I_{n-1} \\ Dh \end{bmatrix} \\ &= [I_{n-1}\vdots 0]DT(x_0)\begin{bmatrix} \begin{bmatrix} I_{n-1} \\ \cdots\cdots \end{bmatrix} \\ -\frac{\frac{\partial g}{\partial u}}{\frac{\partial g}{\partial x_n}} \end{bmatrix} \end{aligned} \tag{7.18}$$

となる．ここに

$$g\,(u_1,\, u_2,\, \cdots,\, u_{n-1},\, h\,(u_1,\, u_2,\, \cdots,\, u_{n-1})) = 0$$

の関係より，$h(u)$ の微分が

$$Dh(u) = -\frac{1}{\frac{\partial g}{\partial x_n}}\,\frac{\partial g}{\partial u} \tag{7.19}$$

となる関係を用いた．

したがって，χ の微分である式 (7.18) は次式となる．

$$D\chi(u) = [I_{n-1}\vdots 0]\left[I_n - \frac{1}{\frac{\partial g}{\partial x}\cdot f} f \cdot \frac{\partial g}{\partial x}\right]\,\frac{\partial \varphi}{\partial x_0}\,\begin{bmatrix} \begin{bmatrix} I_{n-1} \\ \cdots\cdots \end{bmatrix} \\ -\frac{\frac{\partial g}{\partial u}}{\frac{\partial g}{\partial x_n}} \end{bmatrix} \tag{7.20}$$

7.2.1　例 1：2 次元自律系の場合

$$\begin{aligned} \frac{dx}{dt} &= f_1\,(x,\ y) \\ \frac{dy}{dt} &= f_2\,(x,\ y) \end{aligned} \tag{7.21}$$

を考える．Poincaré 断面として

$$g\,(x,\ y) = y = 0 \tag{7.22}$$

を選んだ場合を取り上げよう．

$$\frac{\partial g}{\partial x} = [0,\ 1]\,,\ \frac{dg}{dx} = \frac{\partial g}{\partial x} + \frac{\partial g}{\partial y}Dh = 0 \ \Rightarrow\ Dh = -\frac{\partial g}{\partial x} = 0 \tag{7.23}$$

より

$$D\chi(u) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & -\dfrac{f_1}{f_2} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dfrac{\partial \varphi}{\partial x_0} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & -\dfrac{f_1}{f_2} \end{bmatrix} \dfrac{\partial \varphi}{\partial x_0} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{7.24}$$

となる．

いま，基本解を

$$\frac{\partial \varphi}{\partial x_0} = \begin{bmatrix} \psi_{11} & \psi_{12} \\ \psi_{21} & \psi_{22} \end{bmatrix} \tag{7.25}$$

とすると，式 (7.25) は次式となる．

$$D\chi\left(u\right) = \psi_{11} - \frac{f_1}{f_2}\psi_{21} \tag{7.26}$$

7.2.2　例 2：3 次元自律系の場合

$$\begin{aligned} \frac{dx}{dt} &= f_1\left(x,\ y,\ z\right) \\ \frac{dy}{dt} &= f_2\left(x,\ y,\ z\right) \\ \frac{dz}{dt} &= f_3\left(x,\ y,\ z\right) \end{aligned} \tag{7.27}$$

を考える．Poincaré 断面として

$$g\left(x,\ y,\ z\right) = y = 0 \tag{7.28}$$

が選べる場合を考えよう．Π は $(x,\ z)$ 平面となるので $(x,\ z)$ を局所座標にとる．

$$f_2\left|_{\Pi}\right. \neq 0 \tag{7.29}$$

である．

$$\begin{aligned} &g\left(x,\, y,\, z\right) = y = 0 \\ &y = h\left(x,\, z\right) = 0 \\ &f\left|_{\Pi} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}\right. \end{aligned} \tag{7.30}$$

ただし $f_2 \neq 0$ を仮定した.

$$
\begin{aligned}
Dh &= -\begin{bmatrix} \dfrac{\partial g}{\partial x} & \dfrac{\partial g}{\partial z} \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix}, \qquad \frac{\partial g}{\partial x} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \qquad \frac{\partial g}{\partial x} \cdot f\,|_{\Pi} = f_2 \\
f\,|_{\Pi} \cdot \frac{\partial g}{\partial x} &= \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & f_1 & 0 \\ 0 & f_2 & 0 \\ 0 & f_3 & 0 \end{bmatrix} \\
I_3 - \frac{1}{f_2} f \frac{\partial g}{\partial x} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & \dfrac{f_1}{f_2} & 0 \\ 0 & 1 & 0 \\ 0 & \dfrac{f_3}{f_2} & 0 \end{bmatrix} = \begin{bmatrix} 1 & -\dfrac{f_1}{f_2} & 0 \\ 0 & 0 & 0 \\ 0 & -\dfrac{f_3}{f_2} & 1 \end{bmatrix} \\
D\chi(u,\, v) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & -\dfrac{f_1}{f_2} & 0 \\ 0 & 0 & 0 \\ 0 & -\dfrac{f_3}{f_2} & 1 \end{bmatrix} \begin{bmatrix} \dfrac{\partial \varphi}{\partial x_0} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -\dfrac{f_1}{f_2} & 0 \\ 0 & -\dfrac{f_3}{f_2} & 1 \end{bmatrix} \dfrac{\partial \varphi}{\partial x_0} \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}
\end{aligned}
\tag{7.31}
$$

いま，基本解を

$$
\frac{\partial \varphi}{\partial x_0} = \begin{bmatrix} \psi_{11} & \psi_{12} & \psi_{13} \\ \psi_{21} & \psi_{22} & \psi_{23} \\ \psi_{31} & \psi_{32} & \psi_{33} \end{bmatrix} \tag{7.32}
$$

とすると，式 (7.31) は次式となる.

$$
D\chi(u,\, v) = \begin{bmatrix} \psi_{11} - \dfrac{f_1}{f_2}\psi_{21} & \psi_{13} - \dfrac{f_1}{f_2}\psi_{23} \\ \psi_{31} - \dfrac{f_3}{f_2}\psi_{21} & \psi_{33} - \dfrac{f_3}{f_2}\psi_{23} \end{bmatrix} \tag{7.33}
$$

【注意 1】$x = 0$ を局所断面として選んだ場合は同様に計算して次式を得る.

$$
D\chi(u,\, v) = \begin{bmatrix} \psi_{22} - \dfrac{f_2}{f_1}\psi_{12} & \psi_{23} - \dfrac{f_2}{f_1}\psi_{13} \\ \psi_{32} - \dfrac{f_3}{f_1}\psi_{12} & \psi_{33} - \dfrac{f_3}{f_1}\psi_{13} \end{bmatrix} \tag{7.34}
$$

【注意 2】$z = 0$ を局所断面として選んだ場合は同様に計算して次式を得る.

$$
D\chi(u,\, v) = \begin{bmatrix} \psi_{11} - \dfrac{f_1}{f_3}\psi_{31} & \psi_{12} - \dfrac{f_1}{f_3}\psi_{32} \\ \psi_{21} - \dfrac{f_2}{f_3}\psi_{31} & \psi_{22} - \dfrac{f_2}{f_3}\psi_{32} \end{bmatrix} \tag{7.35}
$$

7.2.3　例 3：4 次元自律系の場合

$$\begin{aligned} \dot{x}_1 &= f_1(x_1,\, x_2,\, x_3,\, x_4) \\ \dot{x}_2 &= f_2(x_1,\, x_2,\, x_3,\, x_4) \\ \dot{x}_3 &= f_3(x_1,\, x_2,\, x_3,\, x_4) \\ \dot{x}_4 &= f_4(x_1,\, x_2,\, x_3,\, x_4) \end{aligned} \tag{7.36}$$

を考える．Poincaré 断面として

$$g(x) = x_4 = 0 \tag{7.37}$$

が選べる場合を考えよう．断面上で

$$f_4(x_1,\, x_2,\, x_3,\, x_4) \neq 0 \tag{7.38}$$

とする．

$$g(u,\ h(u)) = 0 \tag{7.39}$$

より

$$Dg(u,\ h(u)) = \frac{\partial g}{\partial u} + \frac{\partial g}{\partial x_4}\frac{\partial h}{\partial u} = 0 \tag{7.40}$$

したがって

$$\begin{aligned} &Dh(u) = -\frac{\partial g}{\partial u} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix},\ \frac{\partial g}{\partial x} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \\ &I_4 - \frac{1}{f_4} f \frac{\partial g}{\partial x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \frac{1}{f_4}\begin{bmatrix} 0 & 0 & 0 & f_1 \\ 0 & 0 & 0 & f_2 \\ 0 & 0 & 0 & f_3 \\ 0 & 0 & 0 & f_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & f_1/f_4 \\ 0 & 1 & 0 & f_2/f_4 \\ 0 & 0 & 1 & f_3/f_4 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned} \tag{7.41}$$

これより次式を得る．

$$D\chi(u_1,\ u_2,\ u_3) = \begin{bmatrix} \psi_{11} - \dfrac{f_1}{f_4}\psi_{41} & \psi_{12} - \dfrac{f_1}{f_4}\psi_{42} & \psi_{13} - \dfrac{f_1}{f_4}\psi_{43} \\ \psi_{21} - \dfrac{f_2}{f_4}\psi_{41} & \psi_{22} - \dfrac{f_2}{f_4}\psi_{42} & \psi_{23} - \dfrac{f_2}{f_4}\psi_{43} \\ \psi_{31} - \dfrac{f_3}{f_4}\psi_{41} & \psi_{32} - \dfrac{f_3}{f_4}\psi_{42} & \psi_{33} - \dfrac{f_3}{f_4}\psi_{43} \end{bmatrix} \tag{7.42}$$

7.3　Rossler 方程式を解析する

7.3.1　Rössler 方程式の周期解と局所断面上の Poincaré 写像の微分

Rossler 方程式の周期解を計算しよう．プログラムコードを示す前に，問題を整理しておこう．まず，解析する方程式は Rossler 方程式：

$$\begin{aligned} \frac{dx}{dt} &= -y - z = f_1 \\ \frac{dy}{dt} &= x + ay = f_2 \\ \frac{dz}{dt} &= bx - cz + xz = f_3 \end{aligned} \tag{7.43}$$

を考える．

局所断面として

$$g\,(x,\ y,\ z) = y = 0 \tag{7.44}$$

が選べる場合を考えよう．Π は $(x,\ z)$ 平面となるので $(x,\ z)$ を局所座標にとる．

$$f_2\,|_\Pi \neq 0 \tag{7.45}$$

である．

Π 上に Poincaré 写像 T の固定点があったと仮定し，この点を $[u_1,\ 0,\ u_3]$ とする．また，この固定点に対応する周期解の周期を L としよう．このとき，式 (7.43) の解を用いて次の固定点方程式を得る．

$$\begin{aligned} F_1(L, u_1, u_3) &= \varphi_1(L, u_1, 0, u_3) - u_1 = 0 \\ F_1(L, u_1, u_3) &= \varphi_2(L, u_1, 0, u_3) = 0 \\ F_1(L, u_1, u_3) &= \varphi_3(L, u_1, 0, u_3) - u_3 = 0 \end{aligned} \tag{7.46}$$

式 (7.46) を Newton 法によって解くことを考えよう．このため，式 (7.46) を固定点の周りで Taylor 展開し，定数項と線形項を残すと次式を得る．

$$F + Jh = 0 \tag{7.47}$$

ここに

$$F = \begin{bmatrix} F_1(L, u_1, u_3) \\ F_2(L, u_1, u_3) \\ F_3(L, u_1, u_3) \end{bmatrix},\quad h = \begin{bmatrix} \Delta L \\ \Delta u_1 \\ \Delta u_3 \end{bmatrix},\quad J = \begin{bmatrix} f_1 & \dfrac{\partial \varphi_1}{\partial x_1} - 1 & \dfrac{\partial \varphi_1}{\partial x_3} \\ f_2 & \dfrac{\partial \varphi_2}{\partial x_1} & \dfrac{\partial \varphi_2}{\partial x_3} \\ f_3 & \dfrac{\partial \varphi_3}{\partial x_1} & \dfrac{\partial \varphi_3}{\partial x_3} - 1 \end{bmatrix} \tag{7.48}$$

したがって，式 (7.47) に従って，周期 L と座標 $u_1,\ u_3$ に関する Newton の繰り返し公式を構成す

るとよい．すなわち，

$$\begin{bmatrix} L^{(n+1)} \\ u_1^{(n+1)} \\ u_3^{(n+1)} \end{bmatrix} = \begin{bmatrix} L^{(n)} \\ u_1^{(n)} \\ u_3^{(n)} \end{bmatrix} - J^{-1} \begin{bmatrix} \Delta L^{(n)} \\ \Delta u_1^{(n)} \\ \Delta u_3^{(n)} \end{bmatrix} \tag{7.49}$$

なお，ヤコビ行列 J の要素は変分方程式の解を用いて次のように計算できる．

$$\frac{d}{dt}\begin{bmatrix} \dfrac{\partial \varphi_1}{\partial x_1} \\ \dfrac{\partial \varphi_2}{\partial x_1} \\ \dfrac{\partial \varphi_3}{\partial x_1} \end{bmatrix} = \begin{bmatrix} 0 & -1 & -1 \\ 1 & a & 0 \\ b+\varphi_3 & 0 & \varphi_1 - c \end{bmatrix} \begin{bmatrix} \dfrac{\partial \varphi_1}{\partial x_1} \\ \dfrac{\partial \varphi_2}{\partial x_1} \\ \dfrac{\partial \varphi_3}{\partial x_1} \end{bmatrix}, \quad \begin{bmatrix} \dfrac{\partial \varphi_1}{\partial x_1} \\ \dfrac{\partial \varphi_2}{\partial x_1} \\ \dfrac{\partial \varphi_3}{\partial x_1} \end{bmatrix}_{t=0} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{7.50}$$

$$\frac{d}{dt}\begin{bmatrix} \dfrac{\partial \varphi_1}{\partial x_3} \\ \dfrac{\partial \varphi_2}{\partial x_3} \\ \dfrac{\partial \varphi_3}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 0 & -1 & -1 \\ 1 & a & 0 \\ b+\varphi_3 & 0 & \varphi_1 - c \end{bmatrix} \begin{bmatrix} \dfrac{\partial \varphi_1}{\partial x_3} \\ \dfrac{\partial \varphi_2}{\partial x_3} \\ \dfrac{\partial \varphi_3}{\partial x_3} \end{bmatrix}, \quad \begin{bmatrix} \dfrac{\partial \varphi_1}{\partial x_3} \\ \dfrac{\partial \varphi_2}{\partial x_3} \\ \dfrac{\partial \varphi_3}{\partial x_3} \end{bmatrix}_{t=0} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{7.51}$$

また，周期解に対する局所断面上での特性乗数は式 (7.33) を使って計算できる．

7.3.2　固定点を求めるプログラム：RosFix

```
function RosFix
% main program for finding periodic solutions of Rossler's equation
global a b c
global IterMax ErrMax epsx
a=0.6; da=0.01; b=0.5; db=0.01; c=1.6; dc=0.02;
%L=10.9;
L=5.65; % the period
m=512;
%x=[1.13;0.0;2.64;1.0;0;0;0;0;1];
x=[1.44;0.0;1.76;1.0;0;0;0;0;1];
IterMax=10; ErrMax=100.0; epsx=0.00001;

while 1,
    [s_ind, iterN, L, x, chara]=Fixed(L, m, x);
    switch s_ind
        case 0
            % convergence
            disp([iterN a L x(1) x(3) chara]);
        case 1
            % divergence
            disp([s_ind 'States are divergent']);
            break;
        case 2
            % so many iterations
            disp([iterN 'So many iterations in the Newton method']);
            break;
        otherwise
```

```
            % impossible
            disp(['This case never occurs']);
    end
    a=a+da;
end
% end of main program

% Find fixed point: Poincare section is fixed as x(2)=0
function [s_ind, iterN, L, x, chara]=Fixed(L,m,x)
global a b c
global IterMax ErrMax epsx
s_ind=0; dist=1.0; iterN=1; p=zeros(3,1); chara=[];
while dist>epsx,
    iterN=iterN+1;
    p(1)=L; p(2)=x(1); p(3)=x(3);
%    disp([L x(1) x(2) x(3) dist]);
    [L, x]=NewtonR(L,m,x);
    ddx=x(1)-p(2); ddy=x(3)-p(3); ddz=L-p(1);
    dist=(abs(ddx)+abs(ddy)+abs(ddz))/3;
    if dist>ErrMax
        s_ind=1;% divergence
        break;
    end
    if iterN>IterMax
        s_ind=2; % sorry iterating too much
        break;
    end
end
% characteristic multipliers
[V, W]=eig([x(4)+x(3)*x(5)/x(1), x(7)+x(3)*x(8)/x(1);
    x(6)-(b*x(1)-c*x(3)+x(1)*x(3))*x(5)/x(1),...
      x(9)-(b*x(1)-c*x(3)+x(1)*x(3))*x(8)/x(1)]);
chara=[W(1,1) W(2,2)];

% Newton's iteration
function [L, x]=NewtonR(L,m,x)
global a b c
p=[L; x(1); x(3)]; A=zeros(3,3);
x=sysvar(L,m,x);
A(1,1)=-x(3);
A(2,1)=x(1);
A(3,1)=b*x(1)-c*x(3)+x(1)*x(3);
A(1,2)=x(4)-1.0;
A(2,2)=x(5);
A(3,2)=x(6);
A(1,3)=x(7);
A(2,3)=x(8);
A(3,3)=x(9)-1.0;
ppp=p+A\[p(2)-x(1); -x(2); p(3)-x(3)];
L=ppp(1); x(1)=ppp(2); x(2)=0.0; x(3)=ppp(3);

% Poincare map of states
function x=sysvar(L,m,x)
x(4)=1.0; x(5)=0.0; x(6)=0.0; x(7)=0.0; x(8)=0.0; x(9)=1.0;
```

```
h=L/m;
for j=0:m-1
   t=j*h;
   [t, x]=RK(t,x,h);
end

% Runge Kutta method
function [t, z]=RK(t,z,h)
   f1=fn(t,z);
   f2=fn(t+h/2,z+h*f1/2);
   f3=fn(t+h/2,z+h*f2/2);
   f4=fn(t+h,z+h*f3);
   t=t+h;
   z=z+h*(f1+2*(f2+f3)+f4)/6;

% Rossler's equation
function dy=fn(t,y)
global a b c
dy=zeros(9,1);
dy(1)=-y(2)-y(3);
dy(2)=y(1)+a*y(2);
dy(3)=b*y(1)-c*y(3)+y(1)*y(3);
dy(4)=-y(5)-y(6);
dy(5)=y(4)+a*y(5);
dy(6)=(b+y(3))*y(4)+(y(1)-c)*y(6);
dy(7)=-y(8)-y(9);
dy(8)=y(7)+a*y(8);
dy(9)=(b+y(3))*y(7)+(y(1)-c)*y(9);
```

第 8 章

Switched Dynamical Systems with Moving Border

8.1 Formation of the problem

8.1.1 Definition of two dynamical systems and a border manifold B

We define two dynamical systems on R^n:

$$\frac{dx}{dt} = f\left(x,\ \lambda_1\right), \quad x \in R^n \tag{8.1}$$

and

$$\frac{dy}{dt} = g\left(y,\ \lambda_2\right), \quad y \in R^n \tag{8.2}$$

where $\lambda_1,\ \lambda_2$ are the system parameters with appropriate dimensions.

We denote the solution $x(t)$ of Eq. (8.1) as

$$x\left(t\right) = \varphi\left(t,\ t_0,\ x_0\right) \tag{8.3}$$

with the initial condition:

$$x\left(t_0\right) = \varphi\left(t_0,\ t_0,\ x_0\right) = x_0 \tag{8.4}$$

Similarly the solution $y(t)$ of Eq. (8.2) is defined as:

$$y\left(t\right) = \psi\left(t,\ t_0,\ y_0\right) \tag{8.5}$$

with the initial condition:

$$y\left(t_0\right) = \psi\left(t_0,\ t_0,\ y_0\right) = y_0 \tag{8.6}$$

By switching two flows of $x(t)$ and $y(t)$ appropriately, we would like to control the resultant flow on R^n. To this end, let us define a codimension-one manifold B by the pre-image of the function β:

$$B = \{x \in R^n :\ \beta\left(x,\ t\right) = 0\} \tag{8.7}$$

where

$$\beta : R^n \times R \to R; \quad (x,\ t) \mapsto \beta\,(x,\ t) \tag{8.8}$$

The submanifold B will be temporally called the border manifold. We also assume that B is time varing periodically with period L:

$$\beta\,(x,\ t+L) = \beta\,(x,\ t)\,,\ \forall x \in B,\ \forall t \in R \tag{8.9}$$

8.1.2 Switching action on the border B

Let us construct a flow $z(t)$ starting from the initial point $P \in R^n$ at $t = 0$ and terminating at the final point $Q \in R^n$ at $t = L$. Switching action between two flows will be designed to occur whenever our resultant flow meets the border manifold B. In $[0,\ L]$ the number of switching instances is assumed as n_L. As an example of $z(t)$, let the initial flow is defined by Eq. (8.1) , and the switching instances are

$$0 < \tau_1 < \tau_2 < \cdots < \tau_{n_L} < L \tag{8.10}$$

Now we can define the total trajectory $z(t)$ as follows:

1. At the first inteval $0 \le t \le \tau_1$ we have:

$$z(t) = x\,(t) = \varphi\,(t,\ 0,\ P) \quad 0 \le t \le \tau_1 \tag{8.11}$$

2. At the time $t = \tau_1$ the point $x(\tau_1)$ gives the initial point $y\,(\tau_1) = P_1 = x(\tau_1)$ for the flow $y(t)$. The the solution enters into the seecond inteval $\tau_1 \le t \le \tau_2$.

$$z(t) = y\,(t) = \psi\,(t,\ \tau_1,\ P_1) \quad \tau_1 \le t \le \tau_2 \tag{8.12}$$

3. Now we find the initial point $P_2 = y(\tau_2,\ \tau_1,\ P_1)$ for the flow $x(t)$. The solution is obtained by

$$z(t) = x\,(t) = \varphi\,(t,\ \tau_2,\ P_2) \quad \tau_2 \le t \le \tau_3 \tag{8.13}$$

4. Similarly, we can define the trajectory $z(t)$ consecutively at every interval $\tau_k \le t \le \tau_{k+1} \quad k = 3, \cdots, n_L - 1$, and we obtain the solution in the final interval with the initial point P_{n_L-1}. Note that in the final interval the flow is defined by Eq. (8.1) (or Eq. (8.2)) , if n_L is even(or odd). Hence the final point Q is determined by the final flow at $t = L$.

Since the border B moves periodically, we can define our resultant flow $z(t)$ in every interval $[0,\ L]$. Hence $z(t)$ can be defined in $t \in [0,\ \infty)$.

Under these circumstances we would like to investigate the dynamic behaviors of the total system.

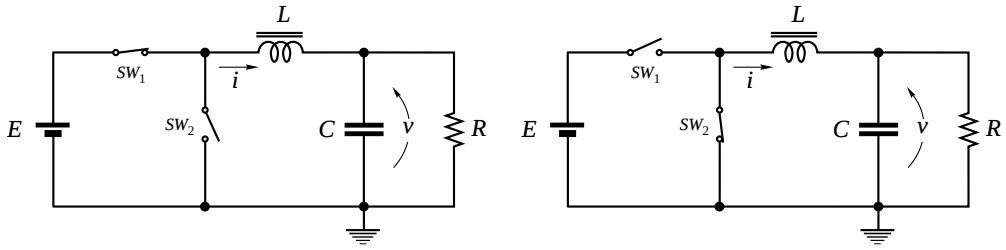


図 8.1　Linear resonant circuits. (a) with a voltage source E. (b) without E.

8.1.3　A simple example

As a simple example, we formulate the dynamics in the case of linear systems with a linear time varing border.

Let us consider linear circuits shown in Fig. 8.1 .

The circuit equations are given by:

$$\begin{aligned} C\frac{dv}{dt} &= i - \frac{v}{R} \\ L\frac{di}{dt} &= -v + E \end{aligned} \tag{8.14}$$

for the circuit (a) and

$$\begin{aligned} C\frac{dv}{dt} &= i - \frac{v}{R} \\ L\frac{di}{dt} &= -v \end{aligned} \tag{8.15}$$

for the circuit (b).

For the border movement, we define

$$\beta\left(v,\ t\right) = av - b\left(t\right) - c = 0 \tag{8.16}$$

As concrete examples of $b(t)$ we illustrate two cases:

$$b\left(t\right) = \cos\frac{2\pi}{L}t \tag{8.17}$$

and

$$b\left(t\right) = \frac{1}{L}t, \quad t \in [0,\ L] \tag{8.18}$$

Note that the latter ramp function is discontinous at $t = 0$ or $t = L$. At this discontinous instance the border $v = \dfrac{1}{a}b\left(t\right) + \dfrac{c}{a}$ jumps from $v = \dfrac{1+c}{a}$ to $v = \dfrac{c}{a}$.

This is the limiting case ($\varepsilon \to 0$) of the function

$$b(t) = \begin{cases} \dfrac{1}{L-\varepsilon} t, & t \in [0,\ L-\varepsilon] \\ -\dfrac{1}{\varepsilon}(t-L), & t \in [L-\varepsilon,\ L] \end{cases} \tag{8.19}$$

This discontinuity causes the change of the number of switching n_L, that is n_L becomes $n_L - 1$ in the discontinous case.

8.2 The Poincaré map and periodic solution

8.2.1 Definition of the Poincaré map

As the border movement is periodical, we can use this periodicity for defining the Poincaré map under the suitable condition. To define the Poincaré map, the vector field at $t = 0$ and $t = L$ must be the same. Therefore, the number of switching n_L has to be even when $\beta(x,\ t)$ is continous in time. Under this condition our total system is invariant at every instance of time: $t = kL,\ k = 0,\ 1,\ \cdots$. Hence we can define the Poincaré mapping T as follows:

$$T : R^n \to R^n; \quad x(0) \mapsto x(L) \tag{8.20}$$

8.2.2 Periodic solution

the case of continuous border

For a moment we assume that the Poincaré map is well defined and $n_L = 2$. Assuming also the existence of a periodic solution, we would like to calculate this periodic solution. See Fig. 8.2 .

Following the solution illustrated in Fig. **??**, we have the relations which give the periodic solution:

$$\begin{aligned} x_1 &= \varphi(\tau_1,\ 0,\ x_0) \\ x_2 &= \psi(\tau_2,\ \tau_1,\ x_1) \\ x_0 &= \varphi(L,\ \tau_2,\ x_2) \end{aligned} \tag{8.21}$$

with the conditions:

$$\begin{aligned} \beta(x(\tau_1),\ \tau_1) &= 0 \\ \beta(x(\tau_2),\ \tau_2) &= 0 \end{aligned} \tag{8.22}$$

These relations contains $3n + 2$ unknown variables ($x_0,\ x_1,\ x_2,\ \tau_1,\ \tau_2$). The number of relations is also $3n + 2$. Hence we can solve these relations.

the case of ramp border

In particular, in the case of the ramp border Eq. (8.18) we can choose $t = 0$ at the discontinuous instance. See Fig. 8.3 . In this case we have the periodicity conditions:

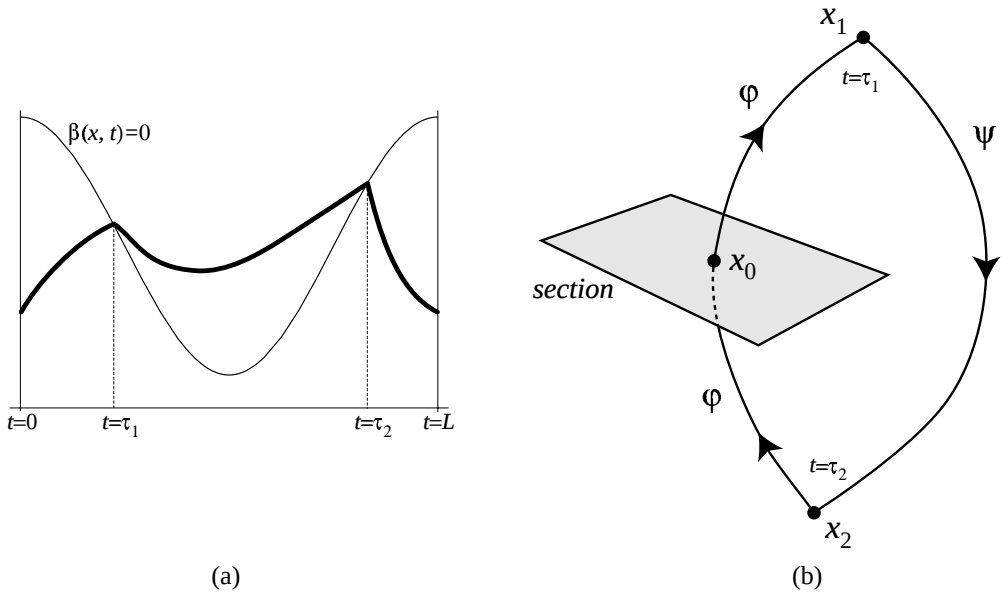


図 8.2　A simple periodic solution. (a) Intersection between the border and the solution . (b) Poincaré section.

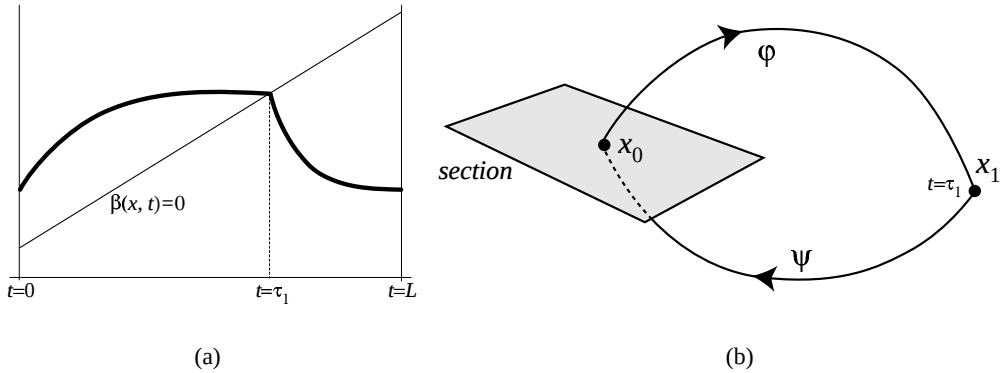


図 8.3　A periodic solution with discontionous border. (a) Intersection between the border and the solution . (b) Poincaré section.

$$\begin{aligned} x_1 &= \varphi\left(\tau_1,\ 0,\ x_0\right) \\ x_0 &= \psi\left(L,\ \tau_1,\ x_1\right) \\ \beta\left(\varphi\left(\tau_1,\ 0,\ x_0\right),\ \tau_1\right) &= 0 \end{aligned} \tag{8.23}$$

Or equivalently, we have

$$\begin{aligned} x_1 &= \varphi\left(\tau_1,\ 0,\ x_0\right) \\ x_0 &= \psi\left(L-\tau_1,\ 0,\ x_1\right) \\ \beta\left(x_1,\ \tau_1\right) &= 0 \end{aligned} \tag{8.24}$$

Assume for a moment the border function as:

$$\beta(x,\ t) = a \cdot x - b(t) - c = 0, \quad t \in [0,\ L] \tag{8.25}$$

differential of periodic solution

As an example of the differential of the Poincaré map, we calculate the derivative of the following relations:

$$\begin{aligned} x_1 &= \varphi(\tau_1(x_0),\ 0,\ x_0) \\ x_2 &= \psi(L - \tau_1(x_0),\ 0,\ \varphi(\tau_1(x_0),\ 0,\ x_0)) \\ \beta(\varphi(\tau_1(x_0),\ 0,\ x_0),\ \tau_1(x_0)) &= 0 \end{aligned} \tag{8.26}$$

That is, we would like to obtain the derivative: $\dfrac{\partial x_2}{\partial x_0}$.

Firstly, from the first relation of Eq. (8.26)

$$\frac{\partial x_1}{\partial x_0} = \frac{\partial \varphi}{\partial t}\frac{\partial \tau_1}{\partial x_0} + \frac{\partial \varphi}{\partial x_0} = f(\varphi)\frac{\partial \tau_1}{\partial x_0} + \frac{\partial \varphi}{\partial x_0} \tag{8.27}$$

Secondary, from the third relation of Eq. (8.26)

$$\begin{aligned} \frac{\partial \beta}{\partial x_0} &= \frac{\partial \beta}{\partial x}\frac{\partial x_1}{\partial x_0} + \frac{\partial \beta}{\partial t}\frac{\partial \tau_1}{\partial x_0} \\ &= \frac{\partial \beta}{\partial x}\left(f(\varphi)\frac{\partial \tau_1}{\partial x_0} + \frac{\partial \varphi}{\partial x_0}\right) + \frac{\partial \beta}{\partial t}\frac{\partial \tau_1}{\partial x_0} = 0 \end{aligned} \tag{8.28}$$

Hence we have

$$\frac{\partial \tau_1}{\partial x_0} = -\frac{1}{\frac{\partial \beta}{\partial x}f(\varphi) + \frac{\partial \beta}{\partial t}}\frac{\partial \beta}{\partial x}\frac{\partial \varphi}{\partial x_0} \tag{8.29}$$

Lastly, we obtain

$$\begin{aligned} \frac{\partial x_2}{\partial x_0} &= \frac{\partial \psi}{\partial t}\left(-\frac{\partial \tau_1}{\partial x_0}\right) + \frac{\partial \psi}{\partial x_1}\frac{\partial x_1}{\partial x_0} \\ &= -g(\psi)\frac{\partial \tau_1}{\partial x_0} + \frac{\partial \psi}{\partial x_1}\left(f(\varphi)\frac{\partial \tau_1}{\partial x_0} + \frac{\partial \varphi}{\partial x_0}\right) \\ &= \left\{-g(\psi) + \frac{\partial \psi}{\partial x_1}f(\varphi)\right\}\frac{\partial \tau_1}{\partial x_0} + \frac{\partial \psi}{\partial x_1}\frac{\partial \varphi}{\partial x_0} \\ &= \frac{\partial \psi}{\partial x_1}\frac{\partial \varphi}{\partial x_0} + \frac{1}{\frac{\partial \beta}{\partial x}f(\varphi) + \frac{\partial \beta}{\partial t}}\left\{g(\psi) - \frac{\partial \psi}{\partial x_1}f(\varphi)\right\}\frac{\partial \beta}{\partial x}\frac{\partial \varphi}{\partial x_0} \end{aligned} \tag{8.30}$$

In the 2-dimensional case with the border Eq. (8.25), our formula becomes as follows:

$$\begin{aligned}\frac{\partial\beta}{\partial x_0} &= \frac{\partial\beta}{\partial x}\frac{\partial x_1}{\partial x_0}+\frac{\partial\beta}{\partial t}\frac{\partial\tau_1}{\partial x_0}\\ &= a\frac{\partial x_1}{\partial x_0}-b'(\tau_1)\frac{\partial\tau_1}{\partial x_0}=a\left(f\frac{\partial\tau_1}{\partial x_0}+\frac{\partial\varphi}{\partial x_0}\right)-b'(\tau_1)\frac{\partial\tau_1}{\partial x_0}\\ &= \{a_1f_1+a_2f_2-b'(\tau_1)\}\frac{\partial\tau_1}{\partial x_0}+a\frac{\partial\varphi}{\partial x_0}=0\end{aligned} \tag{8.31}$$

Hence

$$\begin{aligned}\frac{\partial\tau_1}{\partial x_0} &= \frac{1}{b'(\tau_1)-a_1f_1-a_2f_2}\begin{bmatrix}a_1 & a_2\end{bmatrix}\begin{bmatrix}\frac{\partial\varphi_1}{\partial x_{10}} & \frac{\partial\varphi_1}{\partial x_{20}}\\ \frac{\partial\varphi_2}{\partial x_{10}} & \frac{\partial\varphi_2}{\partial x_{20}}\end{bmatrix}\\ &= \frac{1}{b'(\tau_1)-a_1f_1-a_2f_2}\begin{bmatrix}a_1\frac{\partial\varphi_1}{\partial x_{10}}+a_2\frac{\partial\varphi_2}{\partial x_{10}} & a_1\frac{\partial\varphi_1}{\partial x_{20}}+a_2\frac{\partial\varphi_2}{\partial x_{20}}\end{bmatrix}\end{aligned} \tag{8.32}$$

$$\begin{aligned}\frac{\partial x_1}{\partial x_0} &= \left\{-g(\psi)+\frac{\partial\psi}{\partial x_1}f(\varphi)\right\}\frac{\partial\tau_1}{\partial x_0}+\frac{\partial\psi}{\partial x_1}\frac{\partial\varphi}{\partial x_0}\\ &= \frac{\partial\psi}{\partial x_1}\frac{\partial\varphi}{\partial x_0}+\begin{bmatrix}-g_1(\psi)+\frac{\partial\psi_1}{\partial x_{11}}f_1(\varphi)+\frac{\partial\psi_1}{\partial x_{21}}f_2(\varphi)\\ -g_2(\psi)+\frac{\partial\psi_2}{\partial x_{11}}f_1(\varphi)+\frac{\partial\psi_2}{\partial x_{21}}f_2(\varphi)\end{bmatrix}\frac{\partial\tau_1}{\partial x_0}\\ &= \frac{\partial\psi}{\partial x_1}\frac{\partial\varphi}{\partial x_0}+\frac{1}{b'(\tau_1)-a_1f_1-a_2f_2}\begin{bmatrix}-g_1(\psi)+\frac{\partial\psi_1}{\partial x_{11}}f_1(\varphi)+\frac{\partial\psi_1}{\partial x_{21}}f_2(\varphi)\\ -g_2(\psi)+\frac{\partial\psi_2}{\partial x_{11}}f_1(\varphi)+\frac{\partial\psi_2}{\partial x_{21}}f_2(\varphi)\end{bmatrix}\times\\ &\quad\begin{bmatrix}a_1\frac{\partial\varphi_1}{\partial x_{10}}+a_2\frac{\partial\varphi_2}{\partial x_{10}} & a_1\frac{\partial\varphi_1}{\partial x_{20}}+a_2\frac{\partial\varphi_2}{\partial x_{20}}\end{bmatrix}\end{aligned} \tag{8.33}$$

where we put

$$\begin{gathered}x_0=\begin{bmatrix}x_{10}\\ x_{20}\end{bmatrix},\ x_1=\begin{bmatrix}x_{11}\\ x_{21}\end{bmatrix},\ f=\begin{bmatrix}f_1\\ f_2\end{bmatrix},\ g=\begin{bmatrix}g_1\\ g_2\end{bmatrix}\\ \frac{\partial\varphi}{\partial x_0}=\begin{bmatrix}\frac{\partial\varphi_1}{\partial x_{10}} & \frac{\partial\varphi_1}{\partial x_{20}}\\ \frac{\partial\varphi_2}{\partial x_{10}} & \frac{\partial\varphi_2}{\partial x_{20}}\end{bmatrix},\ \frac{\partial\psi}{\partial x_1}=\begin{bmatrix}\frac{\partial\psi_1}{\partial x_{11}} & \frac{\partial\psi_1}{\partial x_{21}}\\ \frac{\partial\psi_2}{\partial x_{11}} & \frac{\partial\psi_2}{\partial x_{21}}\end{bmatrix}\end{gathered} \tag{8.34}$$

8.2.3 Numerical method

Now let us consider our numerical method for solving Eq. (8.24) under the border function Eq. (8.25). We have to find the roots of the following equations:

$$\begin{aligned} F_1(x_0,\ x_1,\ \tau_1) &= x_1 - \varphi(\tau_1,\ 0,\ x_0) = 0 \\ F_2(x_0,\ x_1,\ \tau_1) &= x_0 - \psi(L-\tau_1,\ 0,\ x_1) = 0 \\ F_3(x_0,\ x_1,\ \tau_1) &= a \cdot x_1 - b(\tau_1) - c = 0 \end{aligned} \tag{8.35}$$

These equations can be solved by Newton's method. To this end we must calculate the Jacobian matrix of Eq. (8.26) , which is obtained as:

$$\begin{bmatrix} \frac{\partial F_1}{\partial x_0} & \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial \tau_1} \\ \frac{\partial F_2}{\partial x_0} & \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial \tau_1} \\ \frac{\partial F_3}{\partial x_0} & \frac{\partial F_3}{\partial x_1} & \frac{\partial F_3}{\partial \tau_1} \end{bmatrix} = \begin{bmatrix} -\frac{\partial \varphi}{\partial x_0}(\tau_1,\ 0,\ x_0) & I_n & -f(\varphi(\tau_1,\ 0,\ x_0),\ \lambda) \\ I_n & -\frac{\partial \psi}{\partial x_1}(L-\tau_1,\ 0,\ x_1) & g(\psi(L-\tau_1,\ 0,\ x_1),\ \lambda) \\ 0 & a & -b'(\tau_1) \end{bmatrix} \tag{8.36}$$

8.3 Phase Portrait をみる：DemoPL

8.4 固定点・周期点を求める：DufFix

8.5 分岐パラメータの計算：DemoBF

付録 A

起動と終了

1. Windows の起動方法
   コンピュータ本体とディスプレイの電源を入れる．数秒後に
   F1··· dos
   F2··· dos
   F3··· BSD
   と表示されるので，すばやく F1 キーを押す．間違えて別のシステムが起動したときは，起動後に Ctrl + Alt + Del を押して再起動する．その後すばやく F1 キーを押す．Windows が立ち上がるとパスワードを聞いてくるのでそのまま，リターンキーを押す．
2. MATLAB の起動方法
   Windows の画面右下に MATLAB へのショートカット アイコンが表示されているので，マウスの左ボタンでダブルクリックする．
3. MATLAB の終了方法
   MATLAB のコマンドラインから
   ≫ exit
   と入力すると終了する．
4. Windows の終了方法
   左下の スタート をクリックし，Windows の終了 を選択し，「コンピュータの電源を切れる状態にする」を選択し，「はい」をクリックする．
5. 計算機の終了方法
   ディスプレイの電源を切る．

付録 B

レポート

この授業の評価は次の項目で行います.

1. 以下にあげる課題についてレポートの評価
2. 講義の出席回数

レポートの提出のない場合は，単位を出すことができません.

B.1　レポートの課題

レポートの課題は次の 3 項目です．以下の節にあるレポートの書き方，例を参考にして，できるだけ簡潔で論理的なレポートを作ってください.

レポートの課題

1. タートルグラフィックス (Turtle graphics: TG) について説明せよ．また，自分のオリジナルな図形を描くプログラムを示し，描いた図形について解説せよ.
2. 次の要領で開ループ伝達関数 $G_0(s)$ を作って，問題に答えよ.
   (a) あなたの生年月日は？　昭和 a 年 b 月 c 日
   (b) あなたの開ループ伝達関数を次式として下さい.

$$G_0(s) = \frac{s+a}{(s+b)(s+c)}$$

   (c) あなたの開ループ伝達関数 $G_0(s)$ のステップ応答曲線を求めて下さい.
   (d) あなたの開ループ伝達関数 $G_0(s)$ のボード線図を求めて下さい.
   (e) あなたの開ループ伝達関数 $G_0(s)$ のナイキスト線図を求めて下さい.
3. 最後の問題は，あなた自身で作って下さい．これまでに習ったどの科目からでも好きな問題を作って，それを解くプログラムを作成し，解答下さい.

B.2　レポートの書き方

レポートは，$A4$ 用紙を使って下さい．表題は「システム解析レポート」として下さい．そのあと，所属学科，学年，出席番号，氏名及び提出年月日を書いて下さい．名前のないレポートは，採点できません．注意して下さい．

ワードプロセッサによりレポートを作成しても結構です．この場合，書式の設定，論文の推敲，文章の保存といった編集作業が容易となるでしょう．プリンターで印刷する場合は，各頁のレイアウトは，上下左右のマージンを適当に取って下さい．左端は綴じるためすこしマージンを広くするとよいでしょう．マージンとしては上: 3.5cm, 下: 2.5cm, 左: 3cm, 右: 2cm 程度を標準とします．各頁には最初を 1 頁として，ヘッダの部分に頁番号を付けて下さい．

あとは，各自の創意工夫によってすばらしいレポートが作成できることを期待します．レポートは何よりも他人に自分の伝えたい情報を提供するために作る資料です．日本語は，簡潔にかつ分かりやすく書きましょう．技術レポートは，これに加えて論理的に組み上げることも大切です．できれば友達に読んでもらって，何度か推敲するとよいでしょう．

B.3　とある学生のレポートの例

システム解析レポート

電気電子工学科 4 年 123 三島　三郎

提出日：1998 年 8 月 23 日

B.3.1　課題 1: タートルグラフィックスについて

タートルグラフィックスとは「ディスプレー画面の中央に仮想のタートル（亀）が上向きにいると考え，この亀を自由に動かすことによって，その軌跡である線図を描くこと」のためのプログラムのことである．

プログラムは，極めて簡単である．次の 4 つの数行プログラムからなる．

1. 初期化ルーチン：TI
2. 回転ルーチン：$R(a)$
3. 前進ルーチン：$F(a)$
4. 表示ルーチン：ST

これらは，講義の時に先生から示された．

そこで，たとえば「正 3 角形を描く」ということであれば，「1 歩進んで 120 度回転する動作を 3 回繰り返すとよい」ので，次のようにプログラムすればよい．

- % draw a triangle by step by step method
  TI;
  F(1); R(120); F(1); R(120); F(1);
  ST

- % draw a triangle by using for end statement
  TI;
  for i=1:3
  F(1);
  R(120);
  end
  ST

さて，私の課題であるが，今回は 3 角形の習作を試みる．正 3 角形を縮めながら描いてこれを基本図形とし，基本図形を更に組み合わせることによって，ある種の対称性を持つ図形を描くことにする．まず，次の 2 つの関数を定義する．

```
function A(a)
% draw a triangle
for i=1:3
  F(a);
  R(120);
end
%
function B(a,b)
% draw b number of triangles
A(a);
R(-120);
c=a;
for i=1:b
  R(125);
  c=0.956*c;
  F(c);
  R(120);
  c=0.864*c;
  F(c);
  R(120);
  F(c);
```

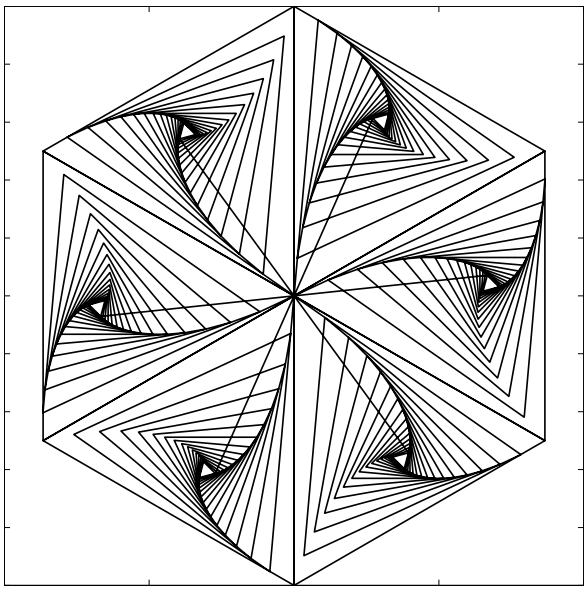


図 B.1　タートルグラフィックで描いた図形.

```
end
```

これを 6 個組み合わせて図形を求める.

```
% draw triangles
TI;
for i=1:6
  B(1,15);
  H([0; 0]);
  R(60*i);
end
```

ここでタートルを最初のホームポジションに帰す関数 $H(a)$ を次のように定義した.

```
function H(a)
% return to the home position
global X U Path
U=[0;1];
Path=a;
X=[X Path];
```

この図を図 B.1 に示しておいた.

最後に，このプログラムの問題点というか「バグ」について述べる．ホームポシションに帰す部分で直線を引いてしまう．これを消去する工夫をすべきである．今回は MATLAB のどのような関数を

使ったらいいのか分からなかったのでバグ付きのままにせざるを得なかった．今後の課題である．

B.3.2　課題 2：制御の問題

私の生年月日は昭和 51 年 8 月 23 日である．したがって私の伝達関数は，次式となる．

$$G_0(s) = \frac{s+51}{(s+8)(s+23)}$$

したがって分母は

$$D(s) = (s+8)(s+23) = s^2 + 31s + 184$$

である．これより次のプログラムによってステップ応答を得る．

```
% step response of the open transfer function
num=[0 1 51];
den=[1 31 184];
step(num, den)
```

この計算結果を図 B.2 に示した．

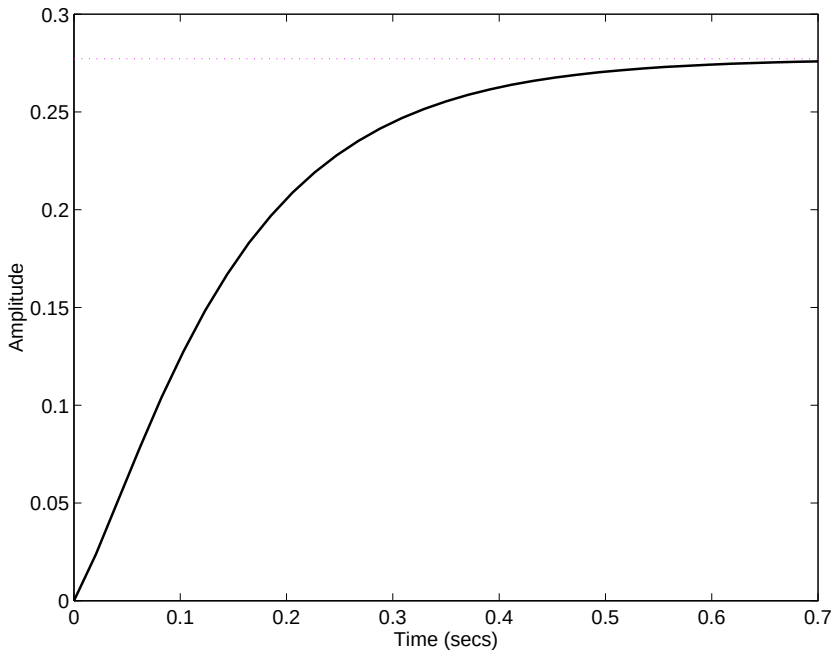


図 B.2　ステップ応答曲線．

つぎに，ボード線図を計算する．

```
% bode diagram
num=[0 1 51];
den=[1 31 184];
bode(num, den)
```

この計算結果を図 B.3 に示した．

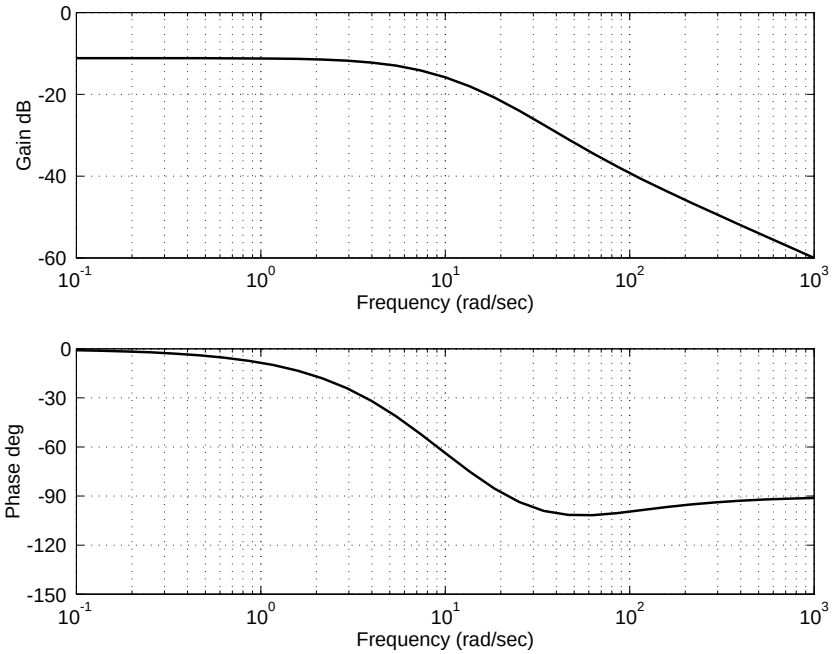


図 B.3　ボード線図.

最後に，ナイキスト線図を計算する.

```
% bode diagram
num=[0 1 51];
den=[1 31 184];
nyquist(num, den)
```

この計算結果を図 B.4 に示した.

B.3.3　課題 3：自由課題

この課題は皆さん各自の個性を発揮する課題です．とある学生の課題の取り組みはここでは省略します．回路や制御からおもしろい問題を探して解くことにして下さい.

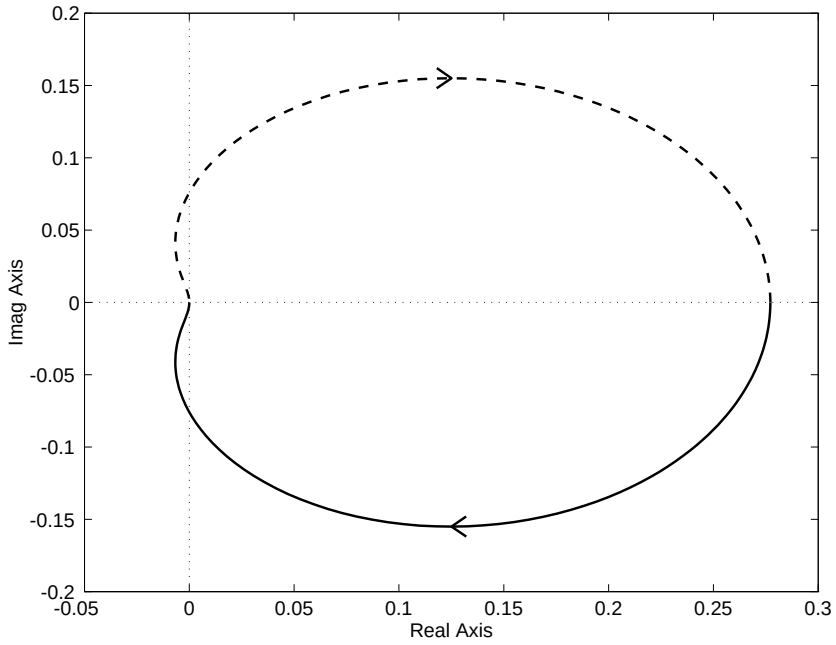

0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
Imag Axis
-0.05
0
0.05
0.1
0.15
0.2
0.25
0.3
Real Axis

図 B.4　ナイキスト線図.