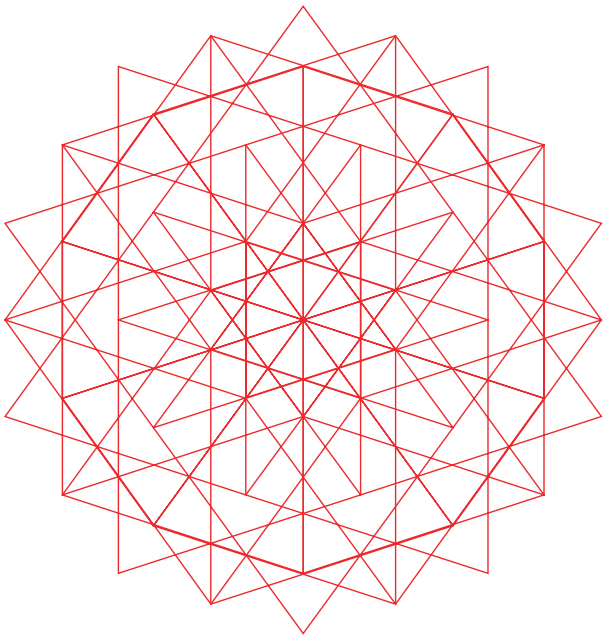
Octave を使いましょう



川上　博

Oct 2004

目次

第 I 部

Octave の基本

第 1 章

行列の入力と加工

Octave は，MATLAB と同様に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．

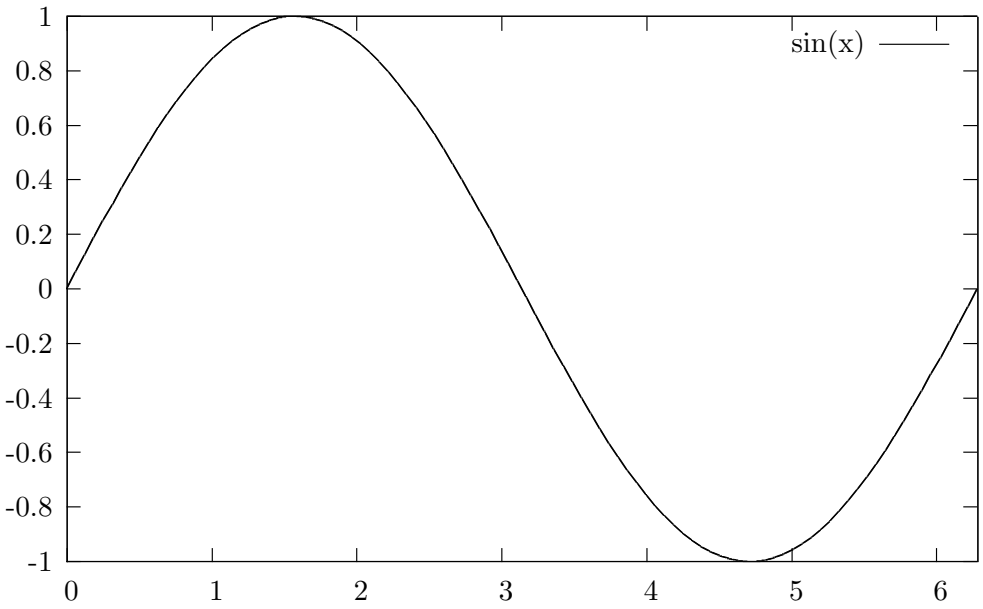


図 1.1　正弦波のグラフ.

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

【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．

2. j は虚数単位 $\sqrt{-1}$ です．i, j, I, J はいずれも虚数単位に予約された定数です．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．

3. 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．

4. A と B の入力後，何が得られましたか．そう

I like Solving our problems with Octave.

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．Octave では，すべての入力データをアレー (array) 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，Octave 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) の定義をしておきましょう．数 (scalar) 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[*1]．2 次元配列になっています．

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

[*1] 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．Octave 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．

表 1.1　Octave 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	$a = 5.5,\ b = 2 * pi + 3j$
スカラー定数	円周率	pi
	虚数単位	$i,\ j,\ I,$ または J
	数でない	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

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．

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

Octave 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．

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．

```
> V=vander([1 2 3 4 5])
```

次の行列が得られます．

$$V = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 16 & 8 & 4 & 2 & 1 \\ 81 & 27 & 9 & 3 & 1 \\ 256 & 64 & 16 & 4 & 1 \\ 625 & 125 & 25 & 5 & 1 \end{bmatrix}$$

この行列を

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

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

  ```
  > V( 2, 3 )
  ```

  とします．4 が答えになったはずです．

  ```
  > V( 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．

> V(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

> V([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．

> V(2, :)

最後の行をみるには

> V(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．

> V(2 : 4, :)

では，こんなのはいかがでしょう．

> V(4 : 2, :)

答えは，[ ](0 × 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．

> V(: , 3)

最後の列をみるには

> V(: , end)

が使えます．

- たとえば 3 列目と 4 列目をみるには

> V(: , 3 : 4)

また，2 行目の 3 列目と 4 列目をみるには

> V(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 W に代入するには，

> W=V(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．

  > V(:)

  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．

  > V=vander([1 2 3 4 5])
  > V(2, :)=[ ]

- たとえば，2 行目と 3 行目を取り除く．

  > V=vander([1 2 3 4 5])
  > V(2 : 3, :)=[ ]

- たとえば，3 列目を取り除く．

  > V=vander([1 2 3 4 5])
  > V(:, 3)=[ ]

- たとえば，2 列目と 3 列目を取り除く．

  > V=vander([1 2 3 4 5])
  > V(:, 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．

  > V=vander([1 2 3 4 5])
  > V(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．

  > V=vander([1 2 3 4 5])
  > V([1,5],:)=V([5,1],:)

- 行を入れ替えて取り出す.

  > V=vander([1 2 3 4 5])
  > V([2, 5,1],:)

- 行の置き換え.

  > V=vander([1 2 3 4 5])
  > W=[1 2 3 4 5]
  > V(2, :)=W

- 行の置き換え.

  > V=vander([1 2 3 4 5])
  > W=[1 2 3 4 5]
  > V(2, :)=W

- 対応する行と列を小行列 N で置き換える.

  > V=vander([1 2 3 4 5])
  > W=[1 2; 3 4]
  > V(2:3, 3:4)=W

- 対応する行と列を小行列 N で置き換える.

  > V=vander([1 2 3 4 5])
  > W=[1 2; 3 4]
  > V([1, 3], [3, 4])=W

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　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 1.2　Octave の行列操作のための関数

Octave 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

第 2 章

行列の演算

いよいよ行列演算の練習です．Octave 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．Octave 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．Octave 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．Octave 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	Octave 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	$A \otimes B = [a_{pq} B]$	$\mathrm{kron}(A,\, B)$
逆行列の積	$AX = B \Rightarrow X = A^{-1} B$	$A \backslash B$
(連立方程式の解)	$XA = B \Rightarrow X = B A^{-1}$	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　Octave の初等関数

	Octave 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

Octave 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．

> V=vander([1 2 3 4 5])
> sum(V)
> trace(V)
> sum(V')
> sum(rot90(V))

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

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

が容易に確かめられますから，方程式

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

[*1] アレー (array) は Octave 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　Octave の行列関数

	Octave 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

の解はとても制限されたものになるでしょう.

よく使われる固有値と固有ベクトルは次のように計算できます.

> [a b]=eig([1 2;3 4])

```
a =
  -0.82456  -0.41597
   0.56577  -0.90938

b =
  -0.37228   0.00000
   0.00000   5.37228
```

a 行列の 2 本の列が 2 つの固有ベクトルを，b 行列の対角要素が対応する固有値となっています.

> -0.37228*[-0.82456;0.56577]-[1 2;3 4]*[-0.82456;0.56577]

を確かめるといいでしょう.

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));
   > axis([0 2*pi -1 1])
2. > t=0 : pi/30 : 2*pi ;
   > plot(cos(t), sin(t))
   > axis([-1 1 -1 1])
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．

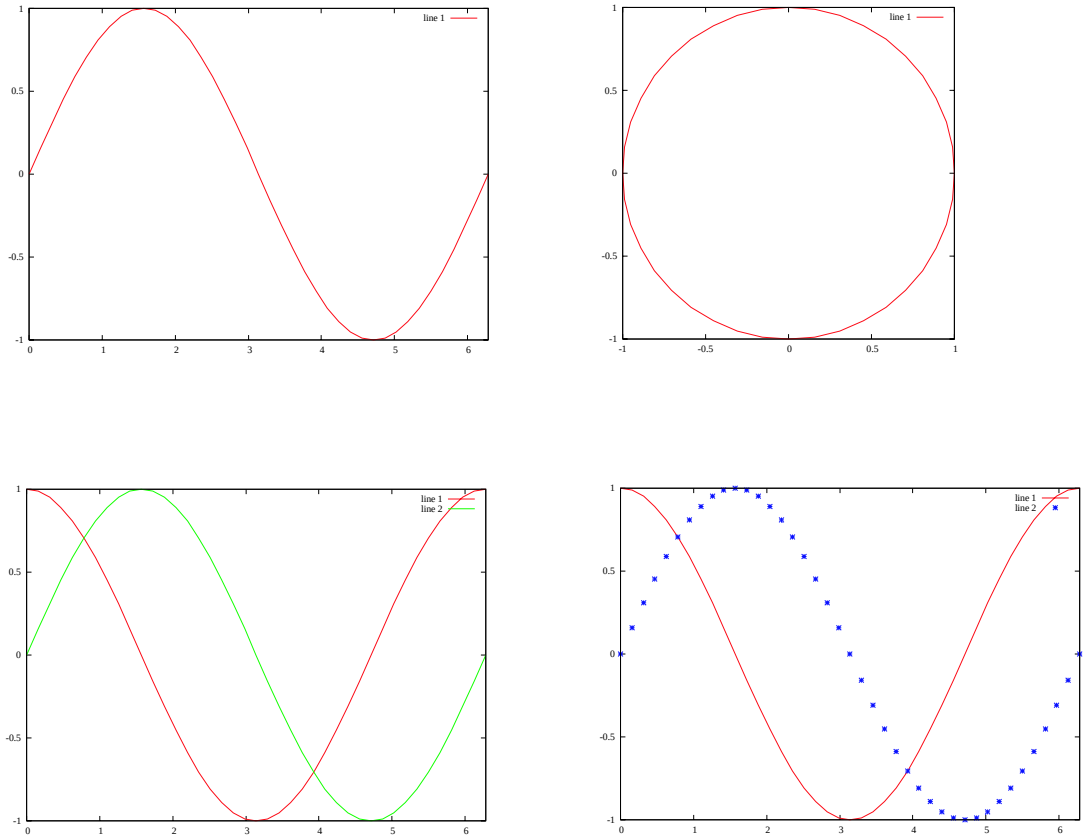


図 3.1　例 3.1.

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．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. $y = 2\sin(t) + \cos(4t)$ と $y = -2\cos(t) + \sin(3t),\ 0 < t < 2\pi$
4. $x = \sin(t)$ と $y = \cos(t) + \sin(3t),\ 0 < t < 2\pi$

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
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

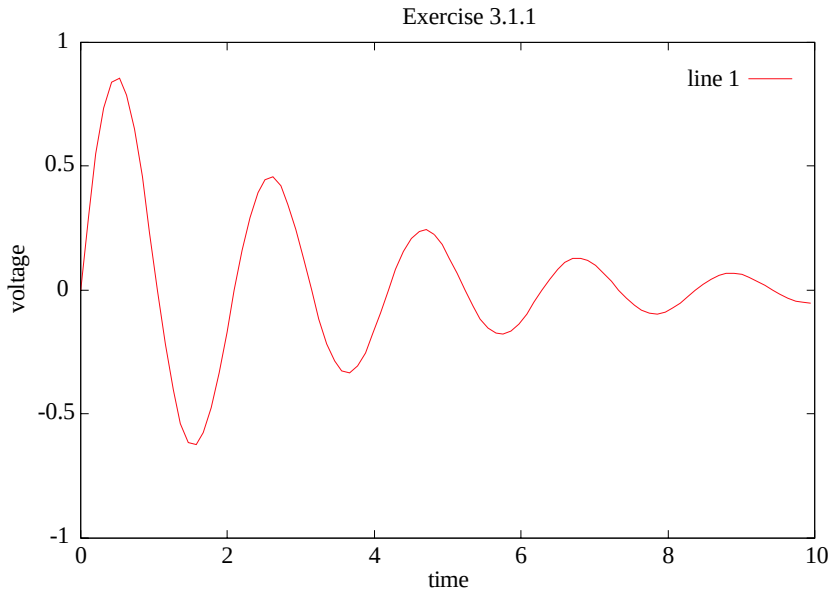


図 3.2　練習 3.1.

なお，グラフの「枠」を正方形にするには

> gset size square

とします．通常は gset size nosquare になっています．また，

> gset title ”Exercise 3.1.1”
> gset xlabel ”time”
> gset ylabel ”voltage”

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

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
clg または clearplot	グラフを消す

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．

```
> grid on
```

とすると「網目」がかかります．grid off で網目は除かれます．また，描いたグラフを消去するには

```
> clg
```

とします[*1]．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

【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(t, sin(2*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．

[*1] clg を一度実行しても図を消去できないようです．もう一度 clg を実行するとうまく消去できます．

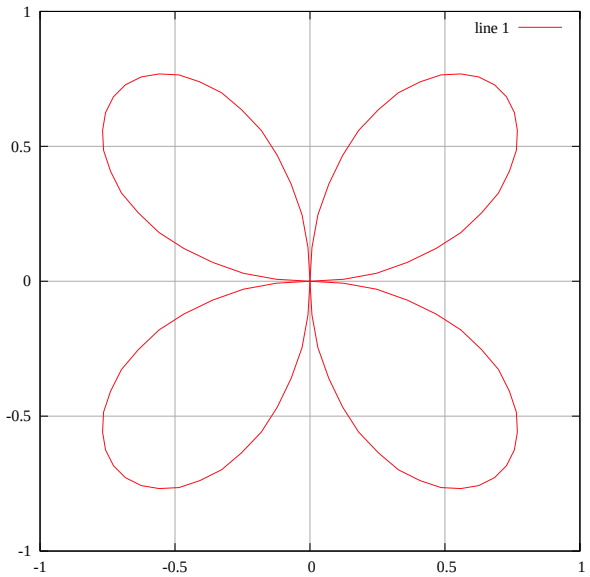


図 3.3　極座標表示 $r = \sin(2\theta)$.

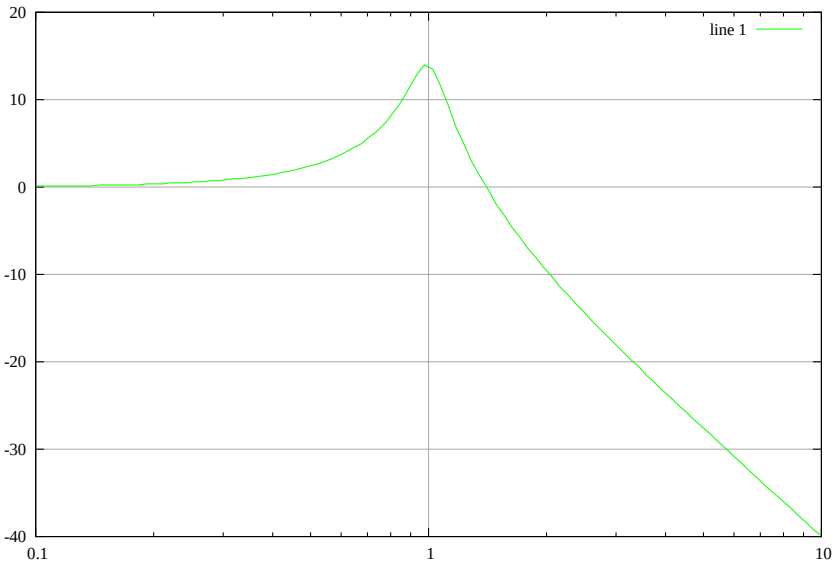


図 3.4　例 3.3： Bode 線図（振幅）.

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．

【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　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[*2]．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

[*2] 元の画面に返すには oneplot() を実行します．

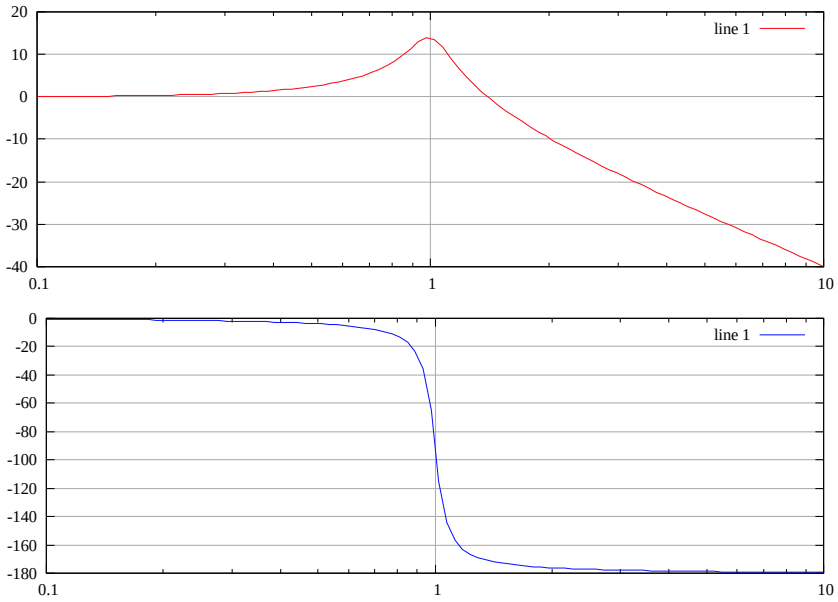


図 3.5　ゲイン線図と位相線図.

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 おかしいですって．では，こうしましょう．

> axis([0.1,10,-180,0])

3.1.5　Gnuplot の命令を使う

これまで描いてきたグラフィックスの命令は，基本的に Matlab と互換性のある命令でした．Octave には，実は Gnuplot が内蔵されていて，作図は Gnuplot の基本命令 plot と splot を，それぞれ gplot と gsplot として使用可能としています．前者 gplot は 2 次元グラフィックスで，これまでに使った plot 命令と同じです．また，後者の gsplot は 3 次元グラフィックスの基本命令です．こちらは次節で説明します．

Matlab と Octave，そして Gnuplot の 3 者の関係が入り交じって Octave のグラフィックス環境はややこしくなっています．それだけ，色々な命令を使って作図可能だということです．Octave では，

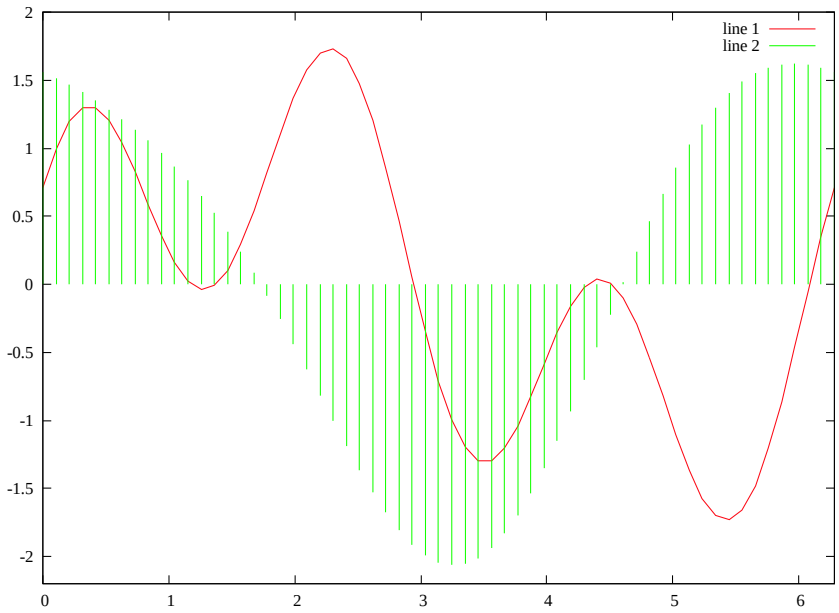


図 3.6　gplot による周期関数の表示.

次のルールに従って Gnuplot の命令を使うことができるようです.

- Gnuplot の命令に，前置詞 g を付けると，Octave の命令として使うことができる．たとえば，Gnuplot の set 命令は，Octave では gset 命令としてまったく同じ働きをする命令として使用できる.

gplot 命令を使って次の 2 つの周期関数を描てみよう.

$$x_1(t) = \sin t + \sin(3t + \frac{\pi}{4}),\ \ x_2(t) = 1.8\cos(t) - 0.3cos(2t - \frac{\pi}{5})$$

> t=[0 : pi/30 : 2*pi]';
> data=[t, sin(t)+sin(3*t+pi/4), 1.8*cos(t)-0.3*cos(2*t-pi/5)];
> gplot [0:2*pi][-2.2:2] data with lines, data using 1:3 with impulses

gplot 命令は 2D グラフィックスの基本命令です.

gplot [表示領域] プロット用データ行列（データ個数 × n 列（t, x1, x2,··· データ））使い方の指示のように使います.

最後に，作図用データの行列の型について注意しておきましょう.

- plot 命令のデータは，行ベクトルでも列ベクトルでもよい.
- gplot 命令は列ベクトルデータのみを処理する.

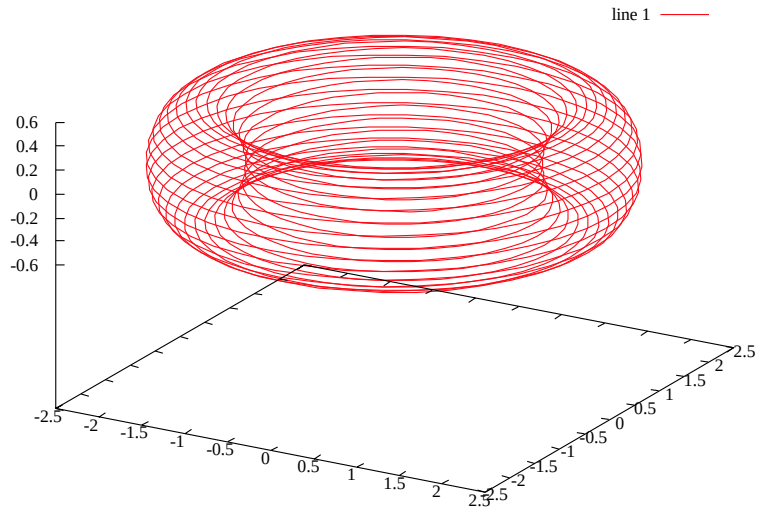


図 3.7　3 次元表示 トーラス.

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　gsplot 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);
> p=[x; y; z]';
> gsplot p
```

gsplot 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

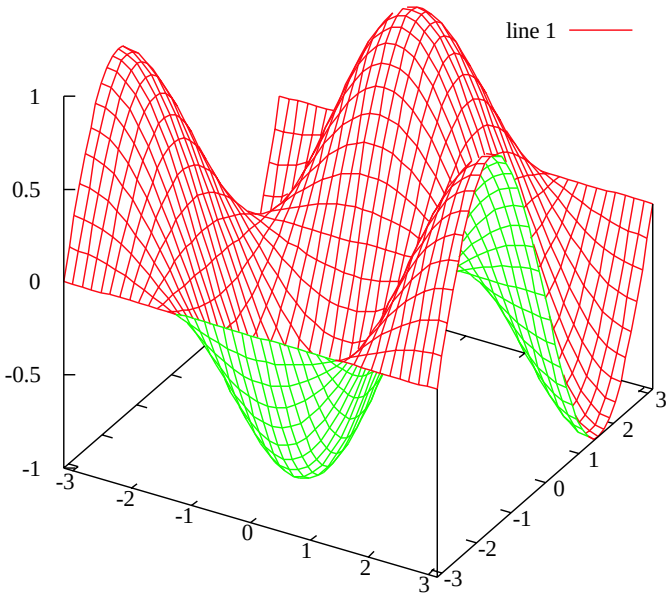


図 3.8　あみめ表示 $z = \sin x \cos x$.

す.

gsplot プロット用データ行列（データ個数 × 3 列（x,y,z データ））

のように使います.

gshow view

とすれば，現在の視点が表示されます．デフォルトで

```
octave:1>
        view is 60 rot_x, 30 rot_z, 1 scale, 1 scale_z
```

となっています．変更は

gset view 45,45

などとします.

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.

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

【解説】関数を定義してワイヤーフレームで曲面を描きます.

> t=-pi:pi/20:pi;
> x=t; y=t;
> z=sin(x')*cos(y);

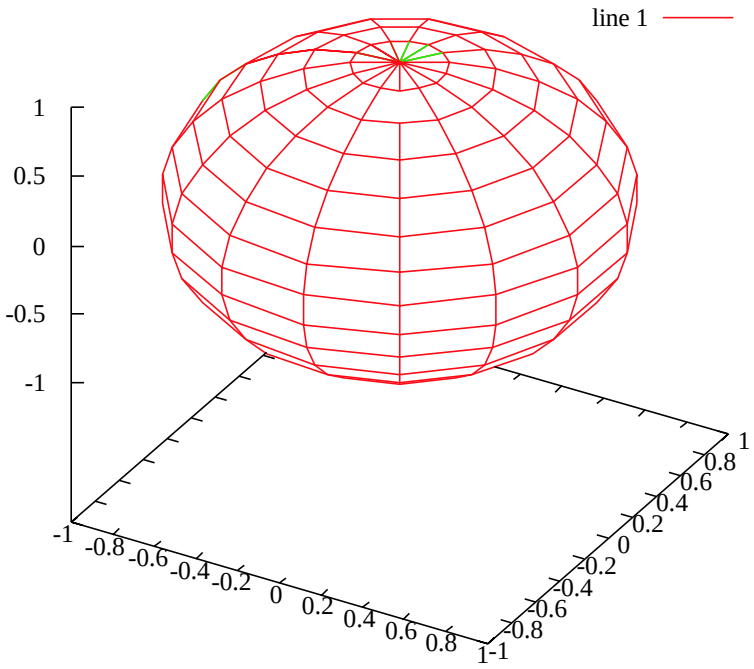


図 3.9　3 次元表示 球.

> mesh(x,y,z)
> axis([-pi pi -pi pi -1 1])
> gset ticslevel 0

3.2.3　パラメータ付けされた曲面を描く

【例 3.6】

球面を描いてみましょう

> 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));
> mesh(x,y,z);

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 Octave のプログラム

Octave には，2 種類のプログラムの扱いがあります．1 つは Octave 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 Octave 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 Octave 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[*1]．このことから，Octave 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 Octave 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．

[*1] これは，明らかに Matlab との互換性を持たせるためであると思われます．

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．適当なエディターを使ってプログラムを作ってください．

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．Octave 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 Octave 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;
endfor
wa
```

このプログラムは繰り返し命令「for　endfor」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;
endfor
%
```

```
% Of course you can write shortly as follows:
% a=sum([1:n])
%
endfunction
```

関数のプログラムは最初の行に

```
function a=wa3(n)
  関数の内容
endfunction
```

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．Matlab の関数と互換性を保つために endfunction は省略してもよいそうです．

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);
endif
endfunction
```

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　endfor」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　endwhile」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；
endfor
```

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；
endwhile
```

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;
endwhile
endfunction
```

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；
  endif
  ```
- ```
  if  論理式
    処理の本体 1；
  else
    処理の本体 2；
  endif
  ```
- ```
  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；
  endif
  ```

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
  printf("Cannot calculate!!\n");
endif
endfunction
```
```

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 endswitch」文を使うと良いでしょう．

> help -i 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;
endfunction
```

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;
endfunction
```

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];
endfunction
```

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];
endfunction
```

5. 表示のスクリプト

```
function ST()
% show graphics:save as "ST.m"
global U X Path deg
gset size square;
plot(X(1, :), X(2, :));
endfunction
```

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);
  endfor
  R(18);
endfor
ST;
```

7. クモの巣スクリプト No. 2

```
TI;
for i=1:10
  for j=1:10
    F(1);
    R(108);
  endfor
  R(36);
endfor
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);
  branch(length/2, level -1);
  R(-90);
  branch(length/2, level -1);
  R(45);
  F(-length);
endif
endfunction
```

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;
endfunction
```

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 Nov. 4, 2004
% - 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
```

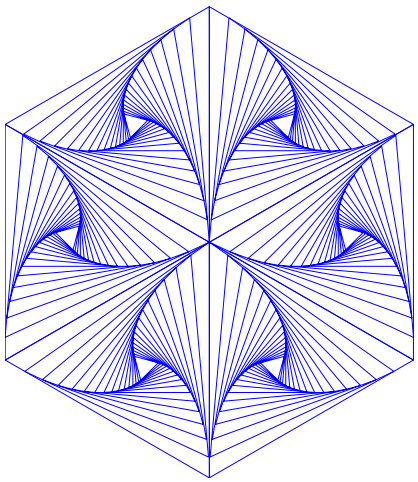


図 4.1　タートルグラフィックスで描いた三角形の集合.

```
    R(60*i+60);
  endif
endfor
ST(1);
endfunction
% ------------------------------------------------------------
% - 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);
endfor
endfunction

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);
endfor
endfunction

% ------------------------------------------------------------
% - 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;
endfunction

function ST(q)
% show turtle
global U X Path deg parity
gset size square;
plot(X(1,:),X(2,:));
endfunction

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;
endfunction

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

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

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

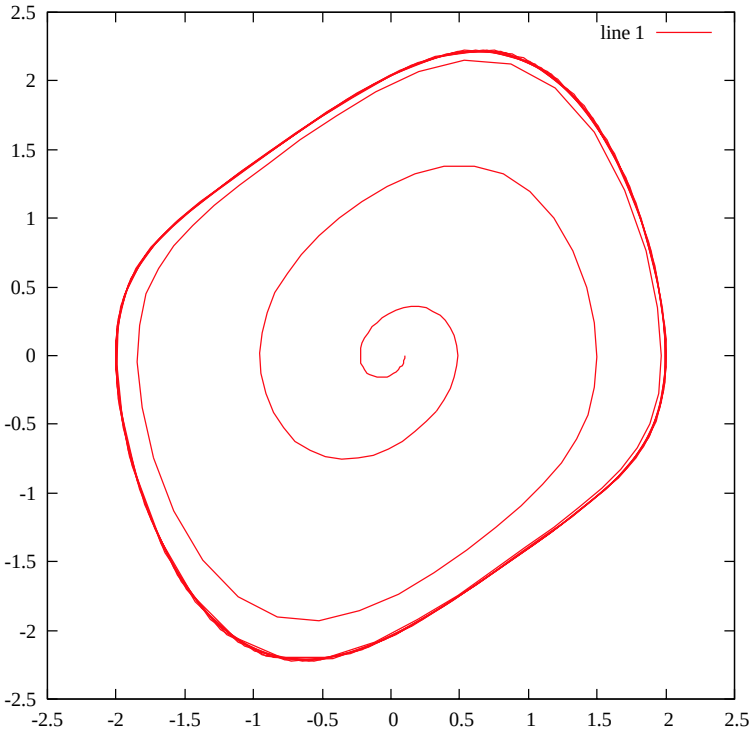


図 4.2　ファン・デア・ポール方程式の相平面図.

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

Octave には常微分方程式を解いてくれる関数が用意されています．これを使って最も短いプログラムを作ってみましょう．「lsode」という関数を使います．初期値と積分する時間間隔を引数に与えると，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 : pi/20 : 20*pi];
x0=[0.1; 0.0];
x=lsode("myvdP", x0, t0);

gset size square
plot(x(:, 1), x(:, 2));
% 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(x, t)
% van der Pol equation
% save as myvdP.m
dx=[x(2); 0.5*(1-x(1)*x(1))*x(2)-x(1)];
endfunction
% 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使って波形を表示することにしましょう．次のプログラムがそれです．

```
function vdp2()
% van der Pol equation: the second essay.
% save as vdP2.m
t0=[0 : pi/20 : 20*pi];
x0=[0.1; 0.0];
x=lsode("myvdP", x0, t0);
% wave forms of x(t) and x-dot(t)
subplot(2,1,1);
axis([0, 20*pi, -2.5, 2.5]);
plot(t,x(:, 1), 'r');
subplot(2,1,2);
axis([0, 20*pi, -2.5, 2.5]);
plot(t,x(:, 2), 'r');
% end of program
endfunction
```

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．

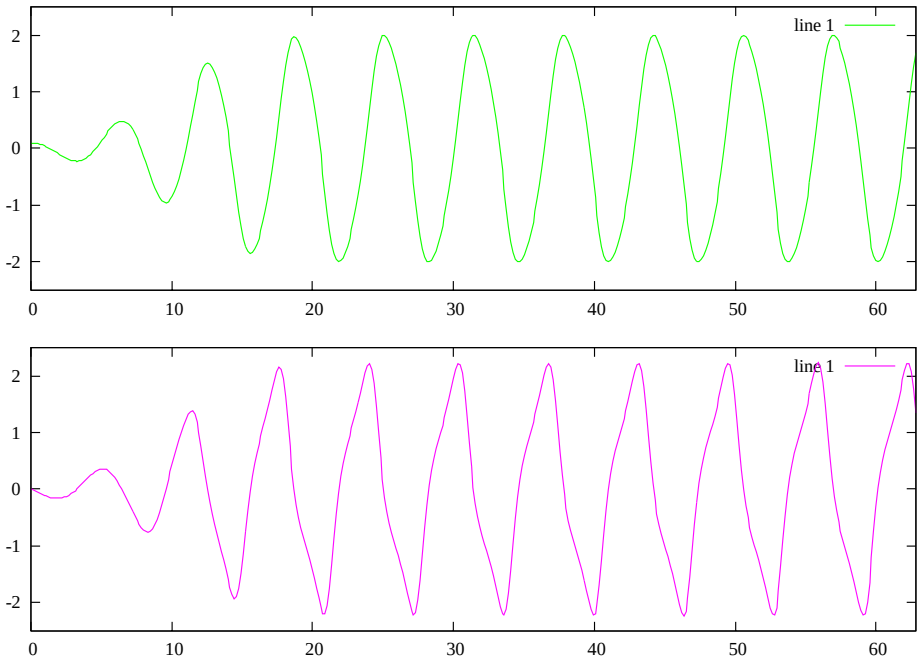


図 4.3　ファン・デア・ポール方程式の発振波形.

4.5.3　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．

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

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

[*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．

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

作ったプログラムは以下の通りです.

```
function Rossler()
% main program
t=0; tmax=200*pi; h=0.1; x=[0.0 3.0 0.0];
y=x;
while t<tmax
  [t,x]=RK(t,x,h);
  y=[y; x];
endwhile

gset data style line;
gset para;
gset size square;
gset ticslevel 0;
gsplot y;
endfunction
%end of main

% Rossler equation
function xdot=fn(t,x)
xdot=zeros(1,3);
xdot(1)=-(x(2)+x(3));
xdot(2)=x(1)+0.35*x(2);
xdot(3)=0.4*x(1)+x(1)*x(3)-4.5*x(3);
endfunction

% 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;
endfunction
```

ここでは，積分公式を自分で書いてみました．Runge-Kutta の 4 段公式を使いました.

一つ注意することは，計算結果のデータの作り方です．この場合 3 列の行ベクトルを時間ステップ毎にに作って gplot のデータとすることにしました．考えてみると，微分方程式の変数は，列ベクトル，行ベクトル，あるいは $m \times n$ 行列となってもいいわけです．利用する場合のデータ構造を加味して積分するといいでしょう.

4.6　よりよいプログラムを書くために

Octave 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 endfor」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，Octave 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);
endfor
```

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);
endfor
```

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 Octave 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！