# Octaveの安易な入門

横田博史

平成18年9月1日(金)

# 目 次

# まえがき

この文書は Octave の入門用に記述したものです. 基本的に私の WebPage [1] で公開している”Scilab を中心とした MATLAB クローン入門” と”Octave で file を利用する話”を基に書き直したものです.

基本的に Octave を中心とした話に変更していますが,MATLAB や SciLab でも使える話になっています.RLab でも使えるかもしれません. しかし,Yorick はやや勝手が違うので, あまり役にたたないかもしれません.

平成 18 年 9 月 9 日 (土) 横田博史

[1] http://www.bekkoame.ne.jp/ ponpoko/Math/MathIndex.html

# 第1章　MATLABクローンについて

## 1.1　MATLABクローン色々

数値行列を扱うアプリケーションとしては,The MathWorks,Inc. の MATLAB が非常に有名です. この MATLAB は C 風の処理言語を持ったアプリケーションで,Toolbox と呼ばれるライブラリや Simulink 等のアプリケーションでシステムの拡張が行えます. 2006 年に出荷された MATLABR2006a から従来の出荷形態が年 2 回に変更され,R2006a の様に R(elease),2006(年),a(期) の様な命名方式となりました.

この MATLAB は幅広く利用されており, 数値行列を扱うアプリケーションで, MATLAB の影響を強く受けたものは沢山あります. その中で, 特に MATLAB の影響が強く現われたアプリケーションとして以下のものが挙げられるでしょう.

MATLAB に似たシステム

- Octave
- RLaB
- SciLab
- Yorick

これらのシステムは全てグラフ表示が可能です. 又, 処理言語の構文や命令は C 風ですが, コンパイラ言語ではなく, 対話処理を行う言語です. 又, 変数は C の様に一々, その型を宣言する必要は無く, 自由気儘に使えます.

この中で Octave が最も MATLAB と互換性が非常に高く,MATLAB クローンと呼んでも差し支え無い程です. 但し, グラフ出力では MATLAB が高機能の為,gnuplot を用いる Octave ではどうしても見劣りすると同時に, 互換性も落ちます.

RLaB は MATLAB に似ていますが別の処理系です. 機能的には Octave と殆ど同じですが, MATLAB との互換性が低い分, 互換性の高い Octave と比較して損をしている面もあります.

SciLab は MATLAB や Octave と似ている点も多くありますが, その一方

で違いも多くあります. 機能は MATLAB と比べて見劣りしない程です. 特に,SciLab には MATLAB の Simulink に似た SCICOS があり, この SCICOS を用いてブロック線図の構築と解析が容易に行えます. 尚, この冊子で述べる行列の処理は SciLab でもほぼ同様に扱えます. 違う点は, 函数の定義の構文, ファイルの入出力といった所で目立ちます.

これが yorick になると, 行列の扱いには MATLAB に似た面もありますが,yorick は対話処理の可能な C と言える程の言語仕様となっており, その為に,MATLAB とは全く独自のソフトウェアになります. 但し, その処理は比較的高速で, その上, 簡単にアニメーションが作製出来るといった長所もあります.

ここでは MATLAB と最も互換性が高く, それ故に他のツールを使う上でも参考になる Octave について, 簡単な行列の処理, グラフ表示とファイルの扱いについて簡単に述べます.

尚, ここで述べる事は MATLAB でもほぼ同様の事が行えます.MATLAB と異なる個所についてはその旨を明記します. 但し, ここで比較対象の MATLAB は古い MATLAB 5.2 の為, 最新のものの場合は異っているかもしれない事を予め断っておきます.

# 第2章　Octaveのユーザーインターフェイスについて

## 2.1　古風なユーザーインターフェイス

Octave を立ち上げても,MATLABR2006a の様に専用のエディタを持った豪華なユーザーインターフェイスを持っていません.Octave は readline を用いた Emacs 風の行や履歴の編集が行える程度です. 入力に関しては履歴として保存されているので, Octave を終了しても或る程度, 履歴を使った処理も可能です. とは言え, 結果も保存される訳ではないので計算結果をちゃんと残しておきたければ,Emacs や TeXmacs といった外部アプリケーションを使う事になります.

TeXmacs の場合は Maxima と同様に, ファイルメニューの 挿入 から セッション を選び, その時に表示されるアプリケーション一覧から, Octave を選ぶと TeXmacs は Octave を起動します. 後は,Octave のプロンプトの直後に処理したい式を入れます. この TeXmacs をフロントエントとして利用すれば,Maxima や Macaulay2 の場合と同様に出力結果も綺麗にレンダリングしたものが表示されます．3 行 3 列の行列の逆行列の計算を TeXmacs 上で処理したものを図 2.1 に示しておきます.

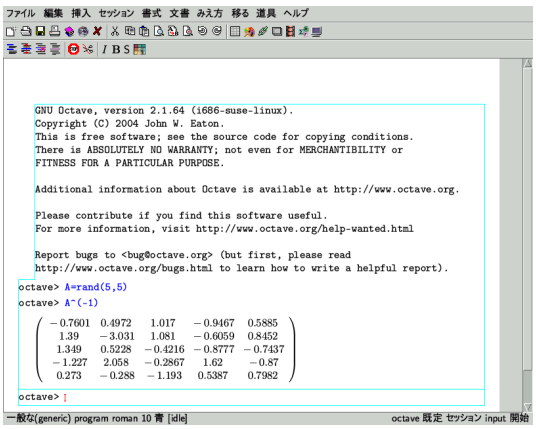


図 2.1: TeXmacs をフロントエンドとして利用

この様に, 綺麗に行列が表示されている事が判りますね. しかし,Maxima の

様に数式を直接扱う場合には綺麗に式が表示される事に十分な価値が認められますが,MATLAB で扱う対象は基本的に数値行列なので, そんなに重宝しないかもしれません. 実際, 大きな行列を表示してもそのままレンダリングして表示するので, 却って見難くなってしまいます.

## 2.2 Octave のヘルプ機能

Octave にはオンラインヘルプ機能があります. このオンラインヘルプの利用方法は単純に `help 〈命令〉` と入力するだけです. この時に表示される Help の内容は, 函数を定義するファイルの先頭に記述した注釈の部分を単純に表示しています. このファイルの拡張子は,m の為に,MATLAB や Octave では M-file と呼びます.

基本的に MATLAB/Octave の函数は M-file と一対一で対応します. 勿論,M ファイルの中に複数の函数を書込む事は構いませんが,help 機能が使えません. この help の機能は MATLAB でも同様です.

尚,Octave の help では M-file の先頭の注釈部分を表示するだけではなく, `help -i 〈事項〉` とする事で関連事項が調べられます.

例えば, 常微分方程式を解きたいがどの様な命令が使えるのか分らない場合, `help -i ode` と入力すれば Octave のオンラインマニュアルから, 常微分方程式を扱う命令に関する事項が表示されます.

因に,`help -i` の場合は,Texinfo を使って info ファイルを表示するものです. その意味では最近の Web Browser を用いるものと比較して古風なものです. その為,Octave をソースファイルからコンパイルして利用する場合には, 予め less や Texinfo といった GNU のアプリケーションを揃えておく必要があります.、

MATLAB の入門書があれば, 試しに MATLAB の命令を help で打ち込んで見ると面白いでしょう. 共通するものが非常に多い一方で, 意外な所で違っていたりします.

# 第3章　Octave初歩

この章では簡単な Octave の使い方の解説を行います.Octave は基本的に数値行列を処理するシステムです. 従って, 基本的に行列で構成された式なので,MATLAB/Octave 独特の行列の扱いに慣れてしまえば使うのも簡単です.

## 3.1　入力

Octave の四則演算は C と同様の数式で表現されます. 例えば,$1+2*3-4^2$ の数値計算を行いたい場合, `1+2*3-4^2` の様に入力します. 尚,Octave の演算子の大きな特徴として, 行列の演算が容易に行える様に工夫が凝らされています. その為,Octave の演算に関しては 3.3 節で詳細を述べる事とします.

Octave は入力行末にセミコロン `;` をつけておくと計算結果を表示しません. これは大きな行列データ処理や反復回数が多い処理で必須になります.

```
octave:1> 1+1;
octave:2> 1+1
ans = 2
```

入力可能なデータには, 整数, 浮動点小数といった数値, そして, 文字列があります. ここで, 文字列は引用符で括られた文字の列, 即ち,`'abc'` や`"efg"` です.

又,MATLAB/Octave ではリスト (=行ベクトル) と行列があります. リストはデータの列をコンマ (`,`) で区切ったものを大括弧 (`[]`) で括ったものです. リストは行ベクトルと同じものです. ここで数値と文字列が混在したリストは許容されません.

尚,`["123","abc"]` の様な文字列のリストはコンマで区切られた文字列が結合されて一つの文字列になります. 実際は文字列が文字のリストなので問題はありませんが, 表向きは大括弧 `[]` が消えてしまいます.

縦ベクトルと行列の書式は数値列, 或いは文字列をセミコロン `;` で区切ったものです.

これは SciLab,MATLAB,Octave でも同じ書式になります. 行列の入力例を以下に示しておきましょう.

```
ans =

  1  2  3
  4  5  6

octave:20> [1,2,3;4,5,6]'
ans =

  1  4
  2  5
  3  6

octave:21> [1,2,3]
ans =

  1  2  3

octave:22> [1;2;3]
ans =

  1
  2
  3
```

これらの例に示す様に MATLAB/Octave は行列やベクトルを視覚的に表示します. 尚, 元の区切は空行かコンマ",", 行の区切には";"を用います. 但し,";"に似ている":"は別の意味を持ちます. この":"の便利な利用方法は後の章で述べる事にします.

次に, 入力した行列 a の i 行 j 列の成分を `a(i,j)` で表現します. ここで i,j は数学の行列表記と同様に 1 から開始します. ベクトルや行列の元の設定は上記の例の様に一気に設定する方法と,`a(1,2)=1` の様に成分を一々指定して設定する方法があります. もし,`a` が未定義の場合に `a(1,2)=1` とすれば, `a` は 1 行 2 列の行列として生成されます. この時,1 行 2 列の成分 `a(1,2)` のみが 1 で他が零の行列になります. 次に,`a(3,2)=10` と続けて入力すれば, 今度は,`a` は 3 行 2 列の行列となり, 値が代入された個所を除く部分の値は全て零になります. この方法以外に行列をベクトルと看做して要素を指定する事も可能です. 具体的には,`a` が m 行 n 列の行列とすれば,`a(i,j)` は `a((j-1)*m+i)` として解釈出来ます. 但し, この場合は `a` が何等の方法で予め定義された場合に限ります. `a` が未定義であれば, この方式で `a` の各成分に値を設定すると,`a` はそのま

まベクトルになります. 尚, 行列の入力や部分の取り出しについては非常に効果的な方法があります. この方法に関しては後の節で詳細を述べます.

数値, 文字列, リスト, 行列の他に, これらのデータを混在して持つ事の可能な構造体も利用可能です. この構造体に関しては次の 3.2 節で詳細を述べます.

## 3.2 変数

Octave で扱える変数は先頭が通常のアルファベットで開始する文字の並びで, その中には Octave の演算子を含まないもの, 或いは,Octave の処理言語で用いられる文字列 (例えば,for,if 等) の予約語以外のものです. 例えば,`x` や `i_1` の様なものが変数として使えますが,`1x` や `x+y` の様なものは変数としては使えません.

次に, 変数への値の割当ては=で行います. 因に==は C と同様に等号の演算子になります.

```
a = 1.2000
octave:4> a
a = 1.2000
octave:5> c=pi
c = 3.1416
octave:6> s=a^2*pi
s = 4.5239
```

ところで,Octave や MATLAB では円周率 $\pi$ , 自然底 $e$ 及び純虚数 $i$ 等の数学定数が予め用意されていますが, その値は立ち上げ時に初期化ファイルを使って設定するものです.MATLAB と Octave では, それらの変数名 (実際は組込函数) は `pi`,`e`, `i` 等となっていますが, これらの変数値は利用者が簡単に変更出来てしまう代物です. 即ち, 予約語として保護されていません.

例えば,for 文を使う際に, 何気なく, 変数 i を内部変数として利用すると, 変数 i は純虚数 $i$ の値が予め設定されていたにも関わらず, 以下の様に呆気無く書換えられてしまいます.

```
octave:1> pi
pi = 3.1416
octave:2> i
i = 0 + 1i
octave:3> e
e = 2.7183
octave:4> i*i
```

```
ans = -1
octave:5> a=1+i;
octave:6> b=1-i;a*b
ans = 2
octave:7>
```

この様に,MATLAB や Octave では i,e,pi は予約語として特別に保護されていないのでうっかり利用しない様に注意しなければなりません (だから, 初期化ファイルで値が設定出来てしまうのです).

ここで利用者が定義した変数は who 命令を使うと表示されます. 更に,type 命令で変数が利用者が定義したものか, 組込のものかどうかが分ります.

以下に変数 i と pi の値を書換えている例を示します.

```
octave:1> who
octave:2> type i
i is a builtin function
octave:3> i=2
i = 2
octave:4> i*i
ans = 4
octave:5> type i
i is a user-defined variable
2
octave:6> type pi
pi is a builtin function
octave:7> pi=10
pi = 10
octave:8> i=sqrt(-1)
i = 0 + 1i
octave:9> pi=acos(-1)
pi = 3.1416
octave:10> who

*** local user variables:

i   pi

octave:11> type pi
pi is a user-defined variable
3.1416
```

```
octave:12> for i=1:10
> i*2;
> end;
octave:13> i
i = 10

octave:14> type i
i is a user-defined variable
10
```

この例では, 一番最初に who 命令を実行した時点では何も表示されませんでしたが, i や pi に値を入れてしまうと利用者定義の変数になってしまうので, 次に,who を実行すると利用者定義の変数として i と pi が表示されます. 実際, 変数の型が調べられる type 命令を使うと, これらの定数は書換えを行った為に利用者定義変数になっている事が分ります. 次の for の例では,i が最終的に 10 で置換えられている事を示しています. ここで,i は添字で,for 文等の反復処理で迂闊に利用する可能性が高い為, 変数の末尾に 1,2,3,$\cdots$ 等の数字を付けたり,ii の様に変数を全て二文字以上にする等の工夫をする方が安全です.

尚, 便利な定数に eps があります. この定数は微少値 (Ocatve の場合は eps=2.2204e-16, 但し, 計算機環境によっては異なるかもしれません) を表現しており, 零による割算を避ける為や,while 文等の反復処理で eps の何倍かに誤差が収まった場合に反復処理を停止する様に設定する等と何かと便利な定数です.

次に,MATLAB/Octave では構造体も利用可能です.C では変数の宣言が必要ですが, MATLAB/Octave では特に宣言する必要はありません.neko.name の様なピリオド . を使ってた変数に値を設定すれば自動的に構造体が設定されます.

```
octave:1> neko.namae="たま"
neko =
{
  namae = たま
}

octave:2> neko.weight="5kg"
neko =
{
  namae = たま
  weight = 5kg
}
```

```
octave:3> neko.age=2
neko =
{
  age = 2
  namae = たま
  weight = 5kg
}
```

この例では構造体 neko に値を設定しています. 構造体にすると, リストや行列とは異なり, 文字列と数値を混在する事が可能なります.

尚, 構造体の値は neko と入力して全てを表示させれば判りますが, これでは効率が良くありませんね.Octave では is_struct 命令 で変数が構造体かどうかを判定し,struct_elements 命令で構造体の構造が調べられます.

```
octave:4> neko
neko =
{
  age = 2
  namae = たま
  weight = 5kg
}

octave:5> is_struct(neko)
ans = 1
octave:6> struct_elements(neko)
ans =

age
namae
weight

octave:7>
```

`is_struct` は真の場合には 1 を返し, 偽の場合には 0 を返します. 基本的に MATLAB/Octave で真偽値は整数値の 1 が真,0 が偽を意味します. 尚,MATLAB の場合は命令の名前は似ていますが, 全く別の命令を用います. その為, 互換性に注意が必要です.

## 3.3 演算子

MATLAB,Octave で演算は C 等のプログラム言語と変わらない形で入力を行います. 例えば,`1+2*3+4^5+6/7` の様に入力すると,Octave/MATLAB は $1+2\cdot 3+4^5+\frac{6}{7}$ を計算します.

この事は数値だけではなく, リストや行列でも同様です. この他にリストや行列に特有の処理もあります.

以下に MATLAB/Octave の演算の一覧を示しましょう. 尚, この一覧では大文字を行列, 小文字をスカラーとしましょう.

MATLAB/Octave の演算子

| 演算 | 例 | 概要 |
|---|---|---|
| 和 (+) | A+B,a+b,A+b | 同じ大きさの行列, 行列とスカラーの和 |
| 差 (-) | A-B,a-b,A-b | 同じ大きさの行列, 行列とスカラーの差 |
| 積 (*) | A*B,a*b,a*B | 行列同士の行列積, 行列とスカラーの積 |
| 冪 (^) | a^b,A^b,a^B | 行列の冪 |
| 商 (/) | A/B,a/b,A/b | 行列同士の場合, 演算子右側を逆行列にして行列積を計算. A/B=A*B^(-1) と同値 |
| 商 (\) | A \ B,a \ b,a \ B | A \ 行列同士の場合, 演算子右側を逆行列にして行列積を計算. B=A^(-1)*B と同値 |
| 積 (.*) | A.*B,a.*b,a.*B | 同じ大きさの行列の成分毎の積, 行列とスカラーの積 |
| 冪 (.^) | A.^B,a.^b,A.^b, a.^B | 行列の冪 |
| 商 (./) | A./B,a./b,a./B | 同じ大きさの行列の成分毎の商 (/), 行列とスカラーの商 |
| 商 (.\ ) | A.\ B,a.\ b,a.\ B, A .\ b | 同じ大きさの行列の成分毎の商 (\), 行列とスカラーの商 |
| 転置 (') | A' | 行列の転置 |

尚, この表に入れませんでしたが,FORTRAN で用いられる冪演算子** も冪演算子^と同じ意味で使えます.

では, 実際に動作を確認してみましょう. 行列 A と行列 B は以下のものとします.

```
octave:1> A=[1,2;3,1]
A =

  1  2
  3  1

octave:2> B=[5,1;2,1]
B =

  5  1
  2  1
```

### 3.3.1 和 (+), 差 (-)

```
octave:3> A+1
ans =

  2  3
  4  2

octave:4> A-1
ans =

  0  1
  2  0

octave:5> A+B
ans =

  6  3
  5  2
```

演算子が和+と差-の場合, 行列同士の演算の場合は両者が同じ大きさでなければなりませんが, 行列とスカラーの場合は可能で, その場合, 各成分とスカラーの和や差になります.

### 3.3.2 積 (*) と積 (.*)

```
octave:9> A*B
ans =

   9   3
  17   4

octave:10> A.*B
ans =

  5  2
  6  1
```

積には二種類あり, 演算子*と演算子.*は二つの被演算子の内, どちらか一方がスカラーであれば, 結果は同値になりますが, 両方が行列の場合は意味が違います. 先ず, 演算子* は通常の行列の積となります. これに対し, 演算子.*は成分毎の積となります.

即ち, 行列 A=($A_{ij}$) と B=($B_{ij}$) に対し,A.*B=($A_{ij} * B_{ij}$) となります. この演算子.*の先頭の. が付く演算子は全て成分毎の演算を意味します.

### 3.3.3 冪 (^と.^)

```
octave:29> A.^2
ans =

  1  4
  9  1

octave:30> 2^A
ans =

  5.6453  4.3104
  6.4656  5.6453
```

通常の冪^のみ行列同士では利用出来ません.

### 3.3.4 商 (/), 商 (\)

```
octave:15> A/B
ans =

  -1.00000   3.00000
   0.33333   0.66667

octave:16> A\B
ans =

  -0.20000   0.20000
   2.60000   0.40000

octave:17> A*B^(-1)
ans =

  -1.00000   3.00000
   0.33333   0.66667
```

```
octave:18> A^(-1)*B
ans =

  -0.20000   0.20000
   2.60000   0.40000

octave:19> 2\B
ans =

  2.50000  0.50000
  1.00000  0.50000

octave:20> A/2
ans =

  0.50000  1.00000
  1.50000  0.50000
```

同じ大きさの行列に対しては,A/B=A*B^(-1),A \ B=A^(-1)*B と各々同値になります. 行列とスカラーの場合, スカラーを分母にする計算は可能ですが, 行列を分母にする計算, 例えば,B \ 2 や 2/A の計算はエラーになります.

尚, 前述の様に, 演算子** も^と同様に使えます.

```
octave:10> [1,2,3].**[2,3,1]
ans =

  1  8  3

octave:11> [1,2,3].^[2,3,1]
ans =

  1  8  3

octave:12>
```

## 3.4 行列の処理について

MATLAB/Octave で扱うデータは基本的に行列です. 処理言語仕様自体が効率良く行列を扱える様に工夫されています.

この節では行列を扱う事を目的とした幾つかの命令と行列の成分の取出し方, 及び演算の処理速度について述べます.

先ず, 行列の大きさは `size` 命令で返されます. 似たものに `length` 命令があり, こちらはベクトルに対してその長さを返します. 尚, 行列に対しては, その列数を返します.

```
octave:23> a=[1 2 3;4 5 6]
a =

  1  2  3
  4  5  6

octave:24> a(1,2)
ans = 2
octave:25> a(4)
ans = 5
octave:26> a(3)
ans = 2
octave:27> size(a)
ans =

  2  3

octave:28> length(a)
ans = 3
```

MATLAB/Octave では行列の一部の取出しが非常に容易に行えます. 例えば, ベクトル `a` の i 番目の成分は `a(i)`, 行列 `A` の i 行 j 列の成分は `A(i,j)` で得られます.MATLAB/Octave のベクトルや行列の成分の添字は,C の配列と異なり,1 から開始します.

ここまでは普通ですが,MATLAB クローン一般で優れているのはベクトルや行列の一部の取り出しが非常に容易に行える事です. 先ず, ベクトルに対し i 番目の成分から j( $\geq$ i) 番目の成分を持つベクトルは `a(i:j)` で得られます. 同時に, 行列 a の i 行目で構成される行ベクトルと j 列目で構成される列ベクトルは各々`a(i,:)` と `a(:,j)` で得られます.

実際にその様子を見てみましょう.

```
octave:32> a=[1:4;5:8;8:11]
a =

   1   2   3   4
   5   6   7   8
   8   9  10  11

octave:33> b=a(2,:)
b =

  5  6  7  8

octave:34> c=a(:,2)
c =

  2
  6
  9
```

a(2,:) で行列 a の第二行を行ベクトル (リスト) として取出しています. それに対し,a(:,2) では第二列を縦ベクトルとして取出しています.

この様に,MATLAB や Octave では, 記号":" を用いて行列同士の代入を簡略化する事が出来ます.

この記号":"を利用して, 行列を列或いは行ベクトルの集合と看做し, これらの成分を持つ行列を以下の例の方法で容易に生成出来ます.

```
octave:37> a=rand(4,4)
a =

  0.204195  0.372247  0.195707  0.529230
  0.063849  0.915721  0.857846  0.630308
  0.191641  0.602701  0.667216  0.162591
  0.261553  0.435798  0.732046  0.561905

octave:38> b=zeros(4,4)
```

```
b =

  0  0  0  0
  0  0  0  0
  0  0  0  0
  0  0  0  0

octave:39> b(:,2)=a(:,4)
b =

  0.00000  0.52923  0.00000  0.00000
  0.00000  0.63031  0.00000  0.00000
  0.00000  0.16259  0.00000  0.00000
  0.00000  0.56191  0.00000  0.00000
```

尚,a(:,1) で代入を行う場合, 代入する側と代入される側の":" で指定した
大きさが, 同じものでなければなりません.

```
octave:39> a
a =

  0.204195  0.372247  0.195707  0.529230
  0.063849  0.915721  0.857846  0.630308
  0.191641  0.602701  0.667216  0.162591
  0.261553  0.435798  0.732046  0.561905

octave:40> c=zeros(2,2)
c =

  0  0
  0  0

octave:41> c(:,2)=a(:,2)
error: a(i, j) = x: x must be a scalar or the number of elements in i must
error: match the number of rows in x and the number of elements in j must
error: match the number of columns in x
error: assignment failed, or no method for ‘matrix = matrix’
error: evaluating assignment expression near line 41, column 7
octave:41> a(:,2)=a(:,2)
```

```
error: a(i, j) = x: x must be a scalar or the number of elements in i must
error: match the number of rows in x and the number of elements in j must
error: match the number of columns in x
error: assignment failed, or no method for ‘matrix = matrix’
error: evaluating assignment expression near line 41, column 7

octave:41> a=a(:,4)
a =

  0.52923
  0.63031
  0.16259
  0.56191
```

この上の例に示す様に":"で示した行列が右辺と左辺の大きさが違う場合にエラーになります. 更に, 左辺が未定義の場合に":"を用いて成分を指定してもエラーになります. この場合には, 予め変数を定義しておく必要があります. 但し, 右辺と左辺の大きさが一致すれば問題はありません.

```
octave:43> d=zeros(10,2)
d =

  0  0
  0  0
  0  0
  0  0
  0  0
  0  0
  0  0
  0  0
  0  0
  0  0

octave:44> a
a =

  0.204195  0.372247  0.195707  0.529230
  0.063849  0.915721  0.857846  0.630308
  0.191641  0.602701  0.667216  0.162591
  0.261553  0.435798  0.732046  0.561905
```

```
octave:45> d([7:10],1)=a(:,1)
d =

  0.00000  0.00000
  0.00000  0.00000
  0.00000  0.00000
  0.00000  0.00000
  0.00000  0.00000
  0.00000  0.00000
  0.20419  0.00000
  0.06385  0.00000
  0.19164  0.00000
  0.26155  0.00000
```

この様に両者の大きさが異なる場合, 両辺に":"を用いて代入すればエラーになりますが, 両辺のサイズが同じになる様に添字を調整すれば, 代入を行う際に問題がありません.

又, 記号:を使って, 数値ベクトルの間隔を指定する事が可能です. この場合,[初期値:刻み幅:終値] で設定します. 例えば 1:5 は 1:1:5 と同じ意味で, 間隔が 1 の場合は真中の間隔指定は不要です. 尚,1:2:6 の場合は,6 を越えない値が設定される為, [1 3 5] が返されます.

```
octave:111>bb=[0:0.1:0.5]
bb =

  0.00000  0.10000  0.20000  0.30000  0.40000  0.50000
```

勿論, 禁断の for 文を用いて次の様に設定する事も出来ますが, 手間は逆に増えてしまいます. 更に,for 文を用いると速度の低下に繋がり易い問題があります. 実際に速度を比較してみましょう.

```
octave:116> t1=time;for i=1:100; bb(i,i)=0.01*i;end;t2=time;t2-t1
ans = 0.0047450
octave:117> t1=time;dd=[1:100];t2=time;t2-t1
ans =  4.9114e-05
```

この場合でも,for 文を使うと著しく速度が低下する事が分りますね.

そこで, 行列の定義でも可能な限り for 文の利用は避けたいものです.

## 3.5 特定の行列の生成

特定の行列に関しては, 容易に行列が生成出来る命令が幾つかあります. 先ず, 対角成分に決まった数値を設定する場合は diag 命令を用います.diag 命令では対角成分に設定する数値をベクトルで与えます. ここで, 対角成分を指定した列程ずらす事も, 与えられた行列の対角成分を抜き出す事も可能です.

対角成分が全て 1 で他が全て 0 の行列の生成は eye 命令を使います.

```
octave:118> diag([1,2])
ans =

  1  0
  0  2

octave:119> diag([1,2],1)
ans =

  0  1  0
  0  0  2
  0  0  0

octave:120> diag([1,2],2)
ans =

  0  0  1  0
  0  0  0  2
  0  0  0  0
  0  0  0  0

octave:121> diag([1,2],-2)
ans =

  0  0  0  0
  0  0  0  0
  1  0  0  0
  0  2  0  0
```

```
octave:122> a=rand(3,3)
a =

  0.58905  0.61873  0.63411
  0.19251  0.11602  0.18785
  0.54143  0.83113  0.83952

octave:123> diag(a)
ans =

  0.58905
  0.11602
  0.83952

octave:124> eye(3,2)
ans =

  1  0
  0  1
  0  0

octave:125> eye(2,3)
ans =

  1  0  0
  0  1  0
```

## 3.6 多項式の扱い

Octave や MATLAB では多項式を扱う事も可能です. 但し, 数式処理の様に直接多項式は扱えません.Octave では多項式の係数ベクトルの形で扱う事になります.

多項式の変換

$a_n x^n + a_{n-1} x^{n-1} + \cdot + a_1 x + a_0 \quad \Leftrightarrow \quad [a_n, a_{n-1}, \cdots, a_1, a0]$

この様に, 多項式は 1 変数のものに限定されます.

尚, 多項式同士の積は conv 命令で行い,deconv 命令で多項式の商と剰余を計算し, polyval 命令で多項式の変数に値を代入した結果を返します.

```
octave:2> conv([1,2],[1,-2])
ans =

   1   0  -4

octave:3> conv([1,2,2,1],[1,-2])
ans =

   1   0  -2  -3  -2

octave:4>  [aa,bb]=deconv([1,3,3,1],[1,1,1])
aa =

  1  2

bb = -1

octave:5> polyval([1,2,3,4],2)
ans = 26
```

この例では, 最初に `conv([1,2],[1,-2])` で 多項式 $(x+2)\cdot(x-2)$ の展開を計算しています. この結果は [1,0,-4] となり, $x^2x^0 - 4$ , 即ち, $x^2 - 4$ に対応します.`deconv` の例では $x^3+3x^2+3x+1$ を $x^2+x+1$ で割った時の商と剰余を計算しています. この結果は $x^3+3x^2+3x+1 = (x^2+x+1)\cdot(x+2)-1$ である事を示しています. この他にも多項式を扱う函数が幾つか存在しますが, どれも直接的な計算を行うものではなく, あくまでも多項式をリストで扱うものです.

この様に,Octave では多項式の処理は得意とは言い難い面があります. だからこそ, この様な処理で複雑なものは Maxima 等の数式処理に任せ, 逆に数値

行列の演算に徹するのが効率の面でも良いでしょう. 何でも一つの道具だけで済まそうとする事は便利そうに見えても思った程, 便利でも無く, 賢明な手段でもありません.

## 3.7 函数の定義と M-ファイル

Octave では直接, 函数定義を行う事も可能ですが,MATLABR2006a では函数定義を直接入力する事は出来ません.

基本的に MATLAB/Octave で函数定義は拡張子が.m のファイル (M-file) に記述します. この M-file がカレントディレクトリ, 或いは M-file が存在するディレクトリが path に含まれていれば, 自動的に M-file の読込んで実行します.

例えば以下の様に函数を記述します.

```
function [z]=nekoneko(x,y)
         if length(x)>length(y)
            z=x;
         else
            z=x./y;
         end;
```

このファイル名を nekoneko.m にしましょう. この nekoneko.m を置いたディレクトリをカレントディレクトリとした状態で texttttoctave と入力して Octave を起動しましょう. それから nekoneko(4,3) と入力してみましょう.Octave は自動的に函数の読込を行って, 計算を実行して結果を返します.

```
octave:1> nekoneko(4,3)
ans = 1.3333
octave:2>
```

Octave で函数を直接定義したい場合には,M-file の内容を記述します. 尚, 函数を定義する場合には end を多目に入力する事になります.

以下に Octave 上で直接函数を定義している様子を示します. ここで,>は Octave の入力待ちのプロンプトです.

```
octave:1> function [z]=nekoneko(x,y)
> if length(x)==length(y)
> z=x./y;
> else if length(x)>length(y)
```

```
> z=x;
> else
> z=y;
> end
> end
> end
octave:2> nekoneko([1:3],[3:-1:1])
ans =

  0.33333  1.00000  3.00000

octave:3> nekoneko([1:3],[3:-1:0])
ans =

  3  2  1  0

octave:4> nekoneko([1:5],[3:-1:0])
ans =

  1  2  3  4  5
```

函数の内容を見たい場合には type 命令を利用します. 実例を以下に示しておきます.

```
octave:6> type nekoneko
nekoneko is a user-defined function:

function z = nekoneko (x, y)
  if length (x) == length (y)
    z = x ./ y;
  else
    if length (x) > length (y)
      z = x;
    else
      z = y;
    endif
  endif
endfunction
octave:7>
```

尚,type で内容を見る事が可能なものは M-file や利用者定義の変数で, 組込函数は事が出来ません. 組込函数かどうかは who 命令で調べる事が可能です.

```
octave:8> who

*** currently compiled functions:

length    nekoneko

*** local user variables:

aa  bb

octave:9> type aa
aa is a user-defined variable
[ 1, 2, 3 ]
octave:10>
```

この様に,who を実行して表示される変数や函数には type 命令で内容を見られます. 尚, $\pi$ 等の数学的定数は組込函数として扱われています. その為,who 命令を実行しても表には出て来ません.

最初の help 命令でも述べた様に, 函数の help 等は M-file の先頭に注釈として記述します.M-file の註釈行は%で開始する行です. では先程の nekoneko.m に以下の注釈を入れて,help を実行してみましょう.

```
%  函数 nekoneko
%
%  こんな風に記述すると、help でこの注釈を
%  表示することが出来るよ (^^)v。
%

function [z]=nekoneko(x,y)
         if length(x)>length(y)
            z=x;
         else
            z=x./y;
         end;
```

に様に nekoneko.m を記述した場合, 以下の様になります.

```
octave:1> help nekoneko
nekoneko is the user-defined function from the file
/home/yokota/WebPage/Math/books/source/octave/nekoneko.m

 函数 nekoneko

 こんな風に記述すると、help でこの注釈を
 表示することが出来るよ (^^)v。


Additional help for built-in functions, operators, and variables
is available in the on-line version of the manual.  Use the command
‘help -i <topic>’ to search the manual index.

Help and information about Octave is also available on the WWW
at http://www.octave.org and via the help@octave.org
mailing list.
octave:2>
```

## 3.8 外部アプリケーションの起動命令

Octave には外部命令を実行する命令として,system 命令 と exec 命令の二種類があります. 因に MATLAB の場合は!を先頭に付けて外部の命令を起動させますが, 行末に; を追加すると外部命令のオプションとして判断されるので注意が必です. Octave では!は否定を表わす not の意味になるので,MATLAB と Octave で動作するプログラムを開発する場合には特に注意が必要になります.

exec 命令は正常に処理の実行が終了しすると Octave 自体を終了させてしまいますが,system 命令は, 実行終了後に Octave に戻るので, 通常は system 命令を用いると良いでしょう.

ここでは system 命令を用いた安易な例を以下で紹介しましょう.

先ず,system 命令で起動させるプログラムは外部のプログラムであれば何でも構いません. ここでは次のシェルスクリプトとします. このスクリプトはカレントディレクトリ上でファイルサイズのみをファイル x1 に出力するものです.

```
#!/bin/sh
ls -l | awk '{print $5}'>x1
```

次に,system 命令を用いるプログラム mike を以下に示します.

```
function x = mike ()
  system ("tama");
  load ("x1");
  x = sum (x1);
endfunction
```

この mike は先程のシェルスクリプト tama を system 命令で起動させて, 結果ファイルの x1 を読込んで, その総和を計算するものです. 尚,system 命令で実行する命令 tama は環境変数 path で設定されたディレクトリかカレントディレクトリ上にあれば大丈夫です. 尚, ディレクトリを `system("/usr/bin/tama")` の様に直接指定しても良いでしょう. 更に,system 命令で実行するプログラムが引数を必要とする場合, `system("tama mike")` の様に通常利用する命令を単に二重引用符"で括ります.

例えば, 次のシェルスクリプト pochi で指定したディレクトリ上のファイルサイズを x1 に出力します. 上述の tama との違いは二行目に`$1` が追加されている点です.

```
#!/bin/sh
ls -l $1 |  awk '{print $5}'>x1
```

この pochi に対し,mike を改良した kuro を以下に示します.

```
function x = kuro (wrd)
     evl=["pochi ",wrd];
     system (evl);
     load ("x1");
     x = sum (x1);
endfunction
```

この kuro では evl で文字列の結合を行っています. 例えば,wrd として"/usr"が与えられると,/usr 上のファイルサイズの総和を計算する事になりますが, evl では文字列"pochi "と word として与えられた"/usr"が結合された"pochi /usr"が代入されます. この値を system 命令で評価 (その為に evl を"で括っていません) して生成された x1 を用いて総和が計算されます.

この様に,system 命令を用いる事で外部プログラムを用いる事が可能となり, 全てを Octave 言語に書き直す手間を省いたり,Octave への外部プログラムの組込みに悩む前に, 安易にシステム動作の確認が行えるので重宝します.

尚,Octave には cd,ls 及び pwd といった命令が利用可能です. 働きも UNIX のそれと全く同じものです.

# 第4章　処理の高速化

## 4.1　行列処理の高速化の工夫

MATLAB や Octave は基本的に行列演算の個所は LAPACK 等の行列計算用のライブラリから函数を呼出して計算させるものです. その為, 対話的に処理を行える割には処理が速いという長所があります.

尚,MATLAB クローン一般でよく生じる事ですが, 処理言語の反復処理を利用して行列の成分を処理すると極端に処理速度が低下します.

勿論, 四則演算も対処方法を誤れば, 計算時間を無駄に消費しかねません. Octave でも和+, 差-, 積.*, 商/と冪^が処理速度の速い順になります. 殊に, 冪乗^ は他の演算と比べて極端に処理が遅い特徴があります. 例えば, a^2 や a^0.5 は a.*a と sqrt(a) の方が圧倒的に高速になります.

そこで, 乱数行列を生成する rand 命令を用いて 1000 行 1 列の行列を生成し,a.*a,a.^2 で成分の二乗を計算する例を示しましょう.

ここで, 返されている数値が処理時間です. この計算は私の計算機 (Pentium 3 1GHz の SuSE 9.3 環境の PC) で実行しています. 但し, この数値は絶対的なものではなく, 参考の為に示しています.

```
octave:9>  t1=time;a.*a;t2=time;t2-t1
ans = 0.00011706
octave:10>  t1=time;a.^2;t2=time;t2-t1
ans = 0.00026608
octave:11> t1=time;exp(2*log(a));t2=time;t2-t1
ans = 0.00070596
octave:12>  t1=time;a.*a.*a.*a.*a.*a.*a.*a.*a.*a.*a;t2=time;t2-t1
ans = 0.00032997
octave:13>   t1=time;a.^12;t2=time;t2-t1
ans = 0.00032210
octave:14>  t1=time;exp(12*log(a));t2=time;t2-t1
ans = 0.00071812
```

この様に二乗の計算でさえも 2 倍近く通常の積の方が冪よりも速い事が判ります. 又, 指数函数と対数函数の組み合わせのものと比べると通常の積は 20

倍近い事が判ります. 但し, 冪の指数が増えても冪や対数函数を用いたものでの差は, 処理時間に大きな差が生じないものの, 積に関しては, 回数が単純に増加する為に, 計算時間に大きな違いが出ており, 冪よりも遅くなる事が判るかと思います.

では, 冪乗を適当に区切って計算した場合はどうなるでしょうか. 今度は 9 乗の計算で 3 個づつに纏めて計算させたものも含めて比較してみましょう.

```
octave:15> t1=time;a.*a.*a.*a.*a.*a.*a.*a.*a;t2=time;t2-t1
ans = 0.00025606
octave:16> t1=time;b=a.*a.*a;b=b.*b.*b;t2=time;t2-t1
ans = 0.00020099
octave:17>  t1=time;b=a.^9;t2=time;t2-t1
ans = 0.00032377
octave:18> t1=time;exp(9*log(a));t2=time;t2-t1
ans = 0.00072813
```

この結果では冪と安易な積を比較すると安易な積の方が速いものの, 冪との差は随分と小さくなっています. これに対し, 三乗で纏めた積は安易な積よりも 1.5 倍程度高速になっています. この様に Octave では工夫次第で積による処理の方が速くなる事が判ります.

以上から, 冪乗も小分けして積で表現した方が速く, 指数が大きくない冪乗と log 函数は可能な限り避けた方が賢明だと言えます.

ここで冪乗に関しては指数の大きさによる面もある為, 注意が必要になりますが, 一般的に, 全体の計算量を削減したものの方が良好な結果を得易いと言えます. この様に全体の計算量を考慮せずに安易な処理を行うと, 逆に悪化する可能性が高くなります.

この様に積に関しても工夫の余地がある事が分りましたが,MATLAB 系の処理言語では安易に for 文を用いるだけで容易に処理速度を低下させる事が出来てしまいます. これは MATLAB 言語等では, 配列から指定された行列の成分を取り出してコピーし, そのコピーを用いて計算した結果を再び配列に戻す事を行っている為です. これに対し, 行列やベクトルの直接演算を行う場合は, 標準的な行列演算ライブラリを直接利用します. その為, 配列の取り出しやコピーの様な余計な手間が不要となるので, 下手にプログラムを組むよりは高速に処理が行えます.

## 4.2 ベクトルに対する並びの照合

MATLAB 風の処理言語では, ベクトルに対して並びの照合処理が可能です. この並びの照合を適用する事によって, 処理速度を引き起し易い for 文等の loop 文を使わずに, 見通しの良いプログラムを行う事が可能になります. この機能は数値データを扱う場合には非常に強力で,MATLAB 風言語の威力が発揮される個所でもあります.

この機能を利用すれば, 与えられたベクトルから適合するものがあるかどうかを検証する事が容易に行えます. 以下の例では与えられたベクトルから 2 に等しいものがあるかを検証し, その場所を `find` 命令を用いて探す処理を実行しています.

尚,Octave と MATLAB では真が 1, 偽が 0 と直接数値で返されます. この事を利用して, 行列の処理だけで for 文等を用いずに全てを処理する事が容易になっています.

```
octave:66> x=[1:5,5:-1:1]
x =

  1  2  3  4  5  5  4  3  2  1

octave:67> x==2
ans =

  0  1  0  0  0  0  0  0  1  0

octave:68> y=find(x==2)
y =

  2  9

octave:69> x(y)
ans =

  2  2
```

この例では 2 に等しいものを検出していますが,C と比べ, 非常に簡単な処理で 2 に等しい元の位置を見付け出しています. この検出は等号だけでは無く, 不等号に対しても適用が可能です.

```
octave:83> x=[1:5,5:-1:1]
x =

  1  2  3  4  5  5  4  3  2  1

octave:84> y=find(x>3)
y =

  4  5  6  7

octave:85> z=x>3
z =

  0  0  0  1  1  1  1  0  0  0

octave:86> z.*x
ans =

  0  0  0  4  5  5  4  0  0  0
```

この find 命令は与えられた行列で 0 と異なる成分の位置を返す命令です. Octave で行列は (i,j) で指定しなければならないので, 並びの照合の対象がベクトルでなければ x(find(x>3)) の様な処理はエラーになります. find(x>3) で返された列ベクトルは $x > 3$ で生成された行列を列ベクトルから構成されたものと看做しています. 具体的には m 行 n 列の行列の (i,j) 成分は $m \times n$ 個の成分の列ベクトルの $m(j-1)+i$ 番目の成分に対応します.

```
octave:5> aa=rand(5);
octave:6> bb=aa>0.5
bb =

  1  1  0  0  1
  1  1  0  1  0
  0  1  1  0  1
  1  0  0  1  0
  0  1  1  1  0
```

```
octave:7> find(bb)
ans =

   1
   2
   4
   6
   7
   8
  10
  13
  15
  17
  19
  20
  21
  23

octave:8>aa(find(bb))
error: single index only valid for row or column vector
error: evaluating index expression near line 8, column 1
octave:8>
```

この例では 0.5 より大となる行列 aa の成分をリストとして出力していますが, その結果をそのまま行列の成分として引渡すとエラーになる例になります.

MATLAB クローンでは,`y=x(x>;3)` の様に `find` 命令を利用せずに処理する事も可能です. この様に並びの照合を適用して処理の簡略化が行えます.

例えば, 上記の与えられたベクトルから 3 よりも大きな数値に対してのみ 2 倍する事も以下の様に簡単に出来てしまいます.

```
octave:89> x=[1:5,5:-1:1]
x =

  1  2  3  4  5  5  4  3  2  1

octave:90> y=find(x>3)
y =

  4  5  6  7
```

```
octave:91> for i=x(y)
> 2*i
> end
ans = 8
ans = 10
ans = 10
ans = 8
```

前述の様に,MATLAB クローンでは for を利用する事は薦められる事ではありません. 寧ろ, 次の方法で処理する方が美しく, 処理も速くなります.

```
octave:1> x=[1:5,5:-1:1]
x =

  1  2  3  4  5  5  4  3  2  1

octave:2> z=zeros(size(x));
octave:3> z(x>3)=2*x(x>3)
z =

   0   0   0   8  10  10   8   0   0   0
```

この処理の一例として $x \geq 0$ の場合は 2 ,$x < 0$ の場合に $-1$ を設定する場合は, 次の様に処理すれば無駄な for 文を使った反復処理なしで, 簡易に済ませられます.

```
octave:10> (x>=0)*2+(x<0)*(-1);
octave:11> tmp=(x>=0);
octave:12> tmp*2+(1-tmp)*(-1)
```

この方法の処理の変種を比較したものを以下に示します. この計算は Pentium 3 1GHz 相当の PC の結果です.

```
octave:105> x=rand(100000,1);
octave:106>  t1=time;(x>=0.5)*2+(x<0.5)*(-1);t2=time;t2-t1
ans = 0.042382
octave:107>  t1=time;tmp=(x>=0.5);tmp*2+(1-tmp)*(-1);t2=time;t2-t1
ans = 0.027326
```

```
octave:108> t1=time;tmp=(x>=0.5);tmp*2+tmp-1;t2=time;t2-t1
ans = 0.023695
octave:109> x;t1=time;for i1=[1:length(x)]
> if x(i1)>=0.5; x(i1)=x(i1)*2; else x(i1)=-x(i1);
> end;end;t2=time;t2-t1
ans = 8.4271
```

最初の例では $x \geq 0.5$ と $x \geq 0.5$ の二種類の並びの照合を実行しています. 二番目の例では 0.5 の並びの照合のみを行っていますが, ここで $-1$ の積を余計に実行しています. 三番目の例では二番目の例と似ていますが, 最後の積を予め実行したものです. そして, 最後の例は for 文を用いて, 一々同じ処理を繰返したものです. 最速のものと比較して実に 400 倍近くもの差が生じていますね. 結局, 行列の積では専門のライブラリが用いられているお陰で, 下手に for 文を用いるよりは効率的に計算が出来ている事が分ります. この理由からも, 安易な for 文の利用は出来るだけ避けて, 行列の演算に置換えられるものは徹底して置換えた方が無難な事が分ると思います.

尚, 並びの照合に関連して,MATLAB や Octave の便利な命令に any と all 命令があります.any 命令は行列データに零でない成分があれば 1 , 零行列であれば 0 を返します. これに対し,all は全ての成分が零でない場合のみ 1 を返し, 他は 0 となります. この命令を用いれば, 更に余計な処理を行わずに済みます.

```
octave:1> a=rand(4,5)-rand(4,5)
a =

  -0.671536   0.539990   0.205556   0.171495   0.276634
  -0.784795   0.585699  -0.274086  -0.448760   0.131415
  -0.072425   0.276092   0.355440  -0.257676   0.357314
   0.356246   0.061500  -0.318618   0.241485  -0.295221

octave:2> if any(a(:,1)>0)
> lst=find(a(:,1)>0);
> b=exp(a(lst,1));
> end;
octave:3> b
b = 1.4280
```

この様に any 命令を使うと, 添字集合として使うリストが空リストかどうかを心配する必要が無くなります. 但し, 全てが一致するかどうかは any 命令

だけでは判別出来ません. 全てが 1 の場合に 1 を返す命令として all 命令があります.

```
octave:11> all(a(:,1)==a(:,3))
ans = 0
octave:12> a(:,1)=a(:,3);
octave:13> all(a(:,1)==a(:,3))
ans = 1
octave:14>
```

この様に並びの照合を適用した数値処理は非常に強力で,for 文による反復処理無しに全てを簡潔に済ます事が可能になります.

# 第5章　グラフ表示

## 5.1　グラフ表示機能

Octave ではグラフ表示に gnuplot を用います．その為,gnuplot がインストールされていない環境ではグラフ表示が行えません.

基本的に, 二次元グラフの表示では Matlab との互換性の問題はあまりありませんが, 曲面表示になると,Octave では機能的にも劣っている為, 互換性は低くなります。

では, 簡単にグラフの表示について解説しましょう．グラフ表示機能で最も互換性が高いのが, 単純な二次元グラフの表示です．二次元表示を行う場合,plot 命令を用います.

plot 命令

plot(⟨$Y$ − データ ⟩, ⟨ 書式 ⟩, ···)
plot(⟨$X$ − データ ⟩, ⟨$Y$ − データ ⟩, ⟨ 書式 ⟩, ···)

ここで,plot 命令に Y 軸データ単体を与える場合, 行や列ベクトル, 或いは行列を与える事が出来ます. ここで,m 行 n 列の行列 A を与えた場合,plot 命令は,A の n 個の列ベクトルを描きます. この場合,X 軸は 1 から n の整数で,Y 座標は, その X 座標に対応する列ベクトルの成分となります.

次に,X と Y のデータを分けて描く事も可能です. この場合,X と Y は同じ大きさのベクトル, 或いは行列が設定可能です.

- 網目を入れたい場合

  グラフに網目を入れたければ, grid 命令を使います. この grid 命令は, 引数として文字列の”on”か”off”だけを取ります.on の場合に網目を入れ,off の場合には網目の表示を解除します.

- グラフの重ね描きをしたい場合

  色々なデータのグラフを重ね描きをしたい場合,hold 命令を使います．この hold 命令も grid 命令と同様に, 引数として文字列の`"on"`か`"off"`だけを取ります.on の場合は重ね描きを行い,off の場合には重ね描きを行わずに, グラフの更新を行います.

- グラフにラベルや表題を入れたい場合

  X 軸,Y 軸,Z 軸上にラベルを入れたければ,xlabel, ylabel,zlabel 命令を用います. 又, 上側に表題を入れたければ,title 命令を用います. これらの命令は一つの文字列を引数とします.

では, 実際に曲線を描いてみましょう.

```
octave:1> a1=[0:0.05:1]*2*pi;
octave:2> b1=sin(a1);
octave:3> plot(b1);
octave:4> plot(a1,b1);
octave:5> xlabel("X")
octave:6> ylabel("Y")
octave:7> title("sine curve");
octave:8> grid("on");
octave:9> hold("on");
octave:10> c1=cos(a1);
octave:11> plot(a1,c1);
octave:12> title("red:sin, green:cos");
octave:13> hold("off");
octave:14> plot(a1,b1,a1,c1);
octave:15> X=[a1;a1];
octave:16> Y=[b1;c1];
octave:17> plot(X,Y);
octave:18> plot(X',Y');
```

この例では正弦函数と余弦函数を描いています. 最初に配列 b1 を描くと, 図 5.1 に示す様に,X 軸には配列番号が振られます.

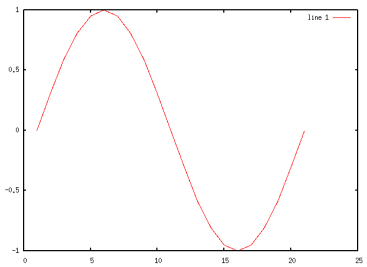


図 5.1: 正弦函数の描画 (Y 座標のみ)

次に, 配列 a1 と配列 b1 の対にすると, 図 5.2 に示す様に, 配列 a1 を X 軸, 配列 b1 を Y 軸の座標としてグラフを描きます.

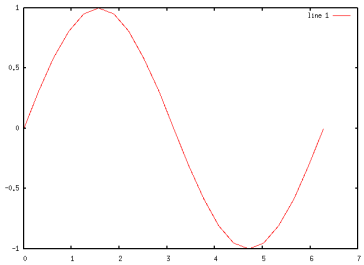


図 5.2: 正弦函数の描画

次に,xlabel と ylabel 命令で X 軸と Y 軸にラベルを入れ,title 命令でグラフの表題を入れてみたものが, 図 5.3 になります. 因に,xlabel 等を用いると, その都度, グラフは更新されます.

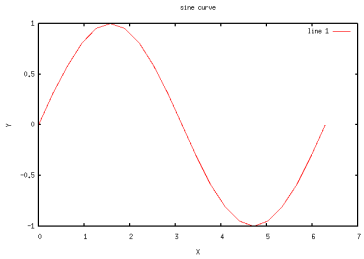


図 5.3: グラフにラベルと表題を追加

それから, grid(”on”) で網目を入れたものが図 5.4 になります.

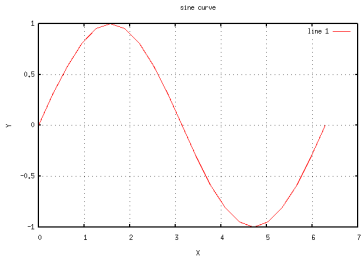


図 5.4: 網目を追加

複数のグラフを描く場合は, 二種類の方法があります. 一つは,hold 命令を使って, plot 命令を繰返して描く方法, もう一つは,plot 命令で一度に描く方法

です. plot 命令で一度に描く場合,`plot(a1,b1,a1,c1)` の様に X と Y の値の対を並べる方法と,X=[a1;a1] と Y=[b1;c1] の様に X の値と Y の値の行列を作って `plot(X',Y')` で一度に描く方法があります. どちらの方法でも図 5.5 の様なグラフが描かれます.

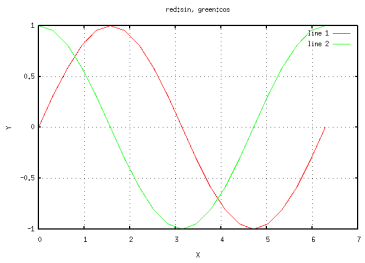


図 5.5: plot(X',Y') のグラフ

尚,`plot(X,Y)` とすると,plot 命令は列ベクトルで描こうとする為, 図 5.6 の様な意味不明のグラフが出るので, この点には注意しましょう.

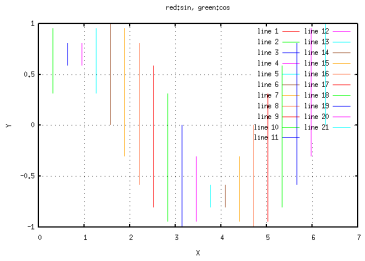


図 5.6: plot(X,Y) のグラフ

# 第6章　Octaveでfileを利用する話

この章では,Octave でファイルを利用する方法について簡単に説明します. その為, ここで説明する命令についても, その機能の一部のみを用いるだけで, 全てを活用しているとは限りません.

## 6.1　load 命令によるデータファイルの処理

最も基本的な, 数値行列を含むファイルの読込処理は load 命令で対処出来ます.

実際に試してみましょう. ここではファイル名を neko にして, 行列データは次の数値にしましょう.

ファイル neko の内容

```
1 2 3
4 5 6
```

では実際にファイル neko の読込を行いましょう. 読込は単純に `load neko` の様にファイル名を必要がればパス付きで指定するだけです.

```
octave:1> load neko
octave:2> neko
neko =

  1  2  3
  4  5  6
```

これで読込みが出来ました. この例で示す様に行列の名前はファイル名が割当てられていますね. では, 続けて, 同じ名前のファイルを読込んでみましょう. どうなりますか？

```
octave:3> load neko
warning: load: local variable name ‘neko’ exists.
warning: use ‘load -force’ to overwrite
```

```
error: load: unable to load variable ‘neko’
error: evaluating index expression near line 3, column 1
```

何か, 色々と文句を言わていますね.Octave では自動的に読込まれたファイル名と同じ名前の変数が既に利用されていると, この様にエラーが返されます. そして, 既存のデータは上書きされずに保存されます.

そこで,load 命令に-force オプションを付けてみましょう. ここでは,`load force neko` と入力してみましょう.

```
octave:3> load -force neko
octave:4> neko
neko =

  1  2  3
  4  5  6
```

今度はどうでしょう. 上書きされていますね.

では, 行列名はどの様な規則で付けられているのでしょう. 実は,neko と同じ内容のファイルで,neko.matrix という名前のファイルがあるので, 続けて読込んでみましょう.

```
octave:5>load neko.matrix
warning: load: local variable name ‘neko’ exists.
warning: use ‘load -force’ to overwrite
error: load: unable to load variable ‘neko’
error: evaluating index expression near line 5, column 1
octave:5> load -force neko.matrix
octave:6> who

*** local user variables:

neko

octave:7>
```

またエラーが出ていますね. 行列名は拡張子を外した名前の部分が使われるので, 変数名は,neko.matrix の場合も neko になります. そこで,load 命令に-force オプションが無かった為に, 変数の内容保護の為にエラーとなったのです. 最後に who と入力してみましょう.neko と返されていますね. これは利

用者定義の変数が neko しか無い事を意味します. この who は利用者が定義した変数を返す命令です.

数値行列ファイルの読込みが出来るなら, その逆はどうするのでしょう. 単純な数値行列データのファイルへの保存には save 命令が使えます.save 命令は先程の who 命令を実行して表示される利用者定義のデータ全ての指定ファイルへの保存と, 個別のデータの保存の両方が行え, その上, 既存ファイルへのデータの追加も行えます.

```
octave:1> a=rand(4,4);
octave:2> b=rand(3,1);
octave:3> save neko a
octave:4> save test
octave:5> who

*** local user variables:

a  b
```

この例では, 行列 a,b を各々rand 命令で生成した 4 行 4 列と 3 行 1 列の行列としています. `save neko a` で, 行列 a をファイル neko に保存し, `save test` とすると,`who` 命令で表示されるデータ全てをファイル test に保存しています. ここで, ファイル neko とファイル test の内容を確認しましょう.

- ファイル neko の内容

```
# Created by Octave 2.0.16, Thu May 10 08:21:44 2001
# name: a
# type: matrix
# rows: 4
# columns: 4
 0.590789258480072 0.222358718514442 0.876821994781494 0.949454307556152
 0.741063475608826 0.656238257884979 0.365377485752106 0.979949235916138
 0.395543217658997 0.417380422353745 0.444111585617065 0.857901215553284
 0.568090081214905 0.558982253074646 0.0379265695810318 0.475694209337234
```

- ファイル test の内容

```
# Created by Octave 2.0.16, Thu May 10 08:21:53 2001
# name: a
# type: matrix
# rows: 4
# columns: 4
 0.590789258480072 0.222358718514442 0.876821994781494 0.949454307556152
 0.741063475608826 0.656238257884979 0.365377485752106 0.979949235916138
 0.395543217658997 0.417380422353745 0.444111585617065 0.857901215553284
 0.568090081214905 0.558982253074646 0.0379265695810318 0.475694209337234
# name: b
# type: matrix
# rows: 3
# columns: 1
 0.628571033477783
 0.415022879838943
 0.216913774609566
```

この様に,save 命令で変数を指定すると, 変数に割当てられた値がファイルに保存され, 変数を指定しないと,who 命令で表示される変数に割当てられた値がファイルに保存されます.

既存ファイルにデータの追加する場合,save 命令に-append オプションを付けます. 例えば, 上記の test ファイルに行ベクトル $c$ を追加したければ, 以下の様に処理を行います.

```
octave:6> c=[1,2,3];
octave:7> save -append test c
```

ここで,$c$ を追加後のファイル test の内容を確認しておきましょう.

```
# Created by Octave 2.0.16, Thu May 10 08:21:53 2001
# name: a
# type: matrix
# rows: 4
# columns: 4
 0.590789258480072 0.222358718514442 0.876821994781494 0.949454307556152
 0.741063475608826 0.656238257884979 0.365377485752106 0.979949235916138
```

```
 0.395543217658997 0.417380422353745 0.444111585617065 0.857901215553284
 0.568090081214905 0.558982253074646 0.0379265695810318 0.475694209337234
# name: b
# type: matrix
# rows: 3
# columns: 1
 0.628571033477783
 0.415022879838943
 0.216913774609566
# name: c
# type: matrix
# rows: 1
# columns: 3
 1 2 3
```

ファイルの保存の場合は load 命令と異なり, 指定したファイルが存在していても, 無警告で処理が実行されます. 実際,test ファイルが存在する状態で test に行列 a と c を保存する例を以下に示しておきます.

```
octave:8> save test
octave:9> save test a c
octave:10>
```

save 命令で保存した行列 a と c を含むファイル test の中身を以下に示します.

```
# Created by Octave 2.0.16, Thu May 10 08:29:54 2001
# name: a
# type: matrix
# rows: 4
# columns: 4
 0.590789258480072 0.222358718514442 0.876821994781494 0.949454307556152
 0.741063475608826 0.656238257884979 0.365377485752106 0.979949235916138
 0.395543217658997 0.417380422353745 0.444111585617065 0.857901215553284
 0.568090081214905 0.558982253074646 0.0379265695810318 0.475694209337234
# name: c
# type: matrix
# rows: 1
# columns: 3
```

```
 1 2 3
```

尚,save で保存したデータは load 命令でそのまま読込めます.

```
octave:1> load test
octave:2> who

*** local user variables:

a  c

octave:3> a
a =

  0.590789  0.222359  0.876822  0.949454
  0.741063  0.656238  0.365377  0.979949
  0.395543  0.417380  0.444112  0.857901
  0.568090  0.558982  0.037927  0.475694

octave:4> c
c =

  1  2  3

octave:5>
```

これで, 最も基本的な数値行列のファイルの読込の方法を終えます.

MATLAB や Octave ではもっと複雑な処理が出来ます. 両方とも言語的にが C に似ている為,C 風にファイルの操作が出来ます. この機能を用いれば, 非常に複雑な形式のファイルの読込と書込も可能です. ところが,Octave の方が処理言語の機能が上なので,MATLAB でも利用可能なプログラムを構築しなければならい時には注意が必要になります. ここでは MATLAB との互換性に留意して記述を纏め,Octave 独自の機能は利用しない様にしています.

### 6.1.1 ファイルの Open と Close

ファイルの Open は fopen 命令を用います.

fopne 命令の構文

id = fopen(⟨ ファイル名 ⟩,⟨ 型 ⟩)

先ず,⟨ ファイル名 ⟩ には'neko' や'test/neko.dat' の様なディレクトリを含めたファイル名前を設定します. 次に,fopen 命令の ⟨ 型 ⟩ は以下のものを用います.

fopen の型

1. 'r' 読込み
2. 'w' 新規に書込み
3. 'a' 末尾に追加
4. 'r+' 読込み. 内容の更新も可
5. 'w+' 書込み. 内容の更新も可
6. 'a+' 末尾に追加. 内容の更新も可.

fopen 命令は整数値を返します. この値 id は以降のファイル操作で利用し, 操作するファイルの指定に利用します. 尚, ファイルが存在しない等のファイルのオープンでエラーを出した場合に-1 が返されるので, この返却値を利用してエラー処理が出来ます. 次に, ファイルの close は C と同様に fclocse 命令を使って,fclose(id) で指定したファイルを閉じます.

ファイル内部の移動ではポインタを用います. ファイルを開くとポインタはファイルの先頭に置かれていますが, ファイルの読込等を行うと下の方に移動して行きます. そこで, ファイルの先頭にポインタを動かす必要が出て来ると,frewind 命令を使います.

データの読込や書込では fgets,fputs,fscanf 等の命令が存在し, 基本的な処理は C と同様の命令が揃っています. ところが, 実際の機能は C の同名の命令とは微妙に異なるので注意が必要になります. 特に MATLAB との互換性を考慮すると,Octave の機能をフルに活用したプログラムはそのままでは使えず, 修正が必要になってしまいます. これは入力と出力書式の指定で顕著です.MATLAB では基本的に行単位で書式が固定され, 微妙な調整が行えません.

その為, 一行に整数, 文字列, 浮動小数が混在する場合, 入出力の書式を指定して読込む方式は避け,fgets で一行を stream として取込んで,stream を分解して sscanf で形式の変換を行う事を薦めます. この方法に関しては次の節で説明しましょう.

## 6.2 データの読み込み

ファイルのデータ読込みは fgets,fscanf 等の命令が使えます. 尚,Octave の fscanf 命令は上述の様に MATLAB のものと比べ機能が強化されており, その上,C-flag を立てる事によって C 言語の fscanf と同じ利用が可能です. ところが,MATLAB では書式が行毎で数値や文字単位ではない為,Octave 専用のプログラムになってしまうので, MATLAB との互換性を重視したプログラムでは, 文字, 数値が混在する行を扱う場合, fgets 命令を使って一行を stream として取り込み, その stream を文字列照合や長さで分割したものに対して sscanf を用いて形式の変換を行う方が, 処理は繁雑になるかもしれないが安全です.

以下に単純な実例を示しましょう. ここでの例では単純な数値行列データに日本語を含めた文字列を含むものです.

neko.txt ファイルの内容

```
1 2 3
3 2 1
はい,1 2 3 ある日森の中, 熊さんに出逢った.
```

このファイルは 1 行と 2 行が数値ですが,3 行目は数値と文字が混在しています. このファイル ('neko.txt') を Octave で開き,fgets を用いて一行づつ読み込む様子を以下に示しましょう.

```
octave:43> fid=fopen('neko.txt','r');
octave:44> L1=fgets(fid)
L1 = 1 2 3

octave:45> L2=fgets(fid)
L2 = 3 2 1

octave:46> L3=fgets(fid)
L3 = はい,1 2 3 ある日森の中, 熊さんに出逢った.

octave:47> frewind(fid)
ans = 0
octave:48> L4=fgets(fid)
L4 = 1 2 3
```

この例では, ファイルの内容参照を行う為に,fopen 命令 の型の指定で,'r' を指定しています. 一般的には,'r' か'r+' でファイルを開きます. ところで, 間

違って'w' や'w+' を指定すると, 直ちにファイルが更新され, 以前の内容が消去された状態となるので注意が必要です.

fgets 命令は例で示す様にファイルの先頭から一行づつ読込みを行います. ここで, ファイルの先頭にポインタを移動させる為には, frewind 命令を使います.

ところで,fgets で返される値のか型は実は文字列型です. この例で, 一見するとベクトルにしか見えない L1 は文字列型です. 実際に,L1 の和を計算しようとすると以下のエラーが出ます.

```
octave:56> L1+L1
error: invalid conversion from string to real matrix
error: invalid conversion from string to real matrix
error: evaluating assignment expression near line 56, column 3
octave:56> size(L1)
ans =

  1  6

octave:57> L1
L1 = 1 2 3
```

この様に, テキストファイルデータを fgets 命令で読込むと,fgets か返す値は全て文字列型になってしまいます. そこで, 必要に応じて型の変換を行わなければなりません. 型の変換は C と同様に sscanf 命令を用います.

sscanf の型の指定

| | | |
|---|---|---|
| $\%d$ | $\Rightarrow$ | 整数型データに変換 |
| $\%f$ | $\Rightarrow$ | 浮動小数点型データに変換 |
| $\%s$ | $\Rightarrow$ | 文字列型データに変換 |

以下に, 文字列 L1=1 2 3 と L2=3 2 1 を sscanf 命令を使って型の変換を行った実例を示します.

```
octave:51> a11=sscanf(L1,'%d')
a11 =

  1
  2
  3

octave:52> a12=sscanf(L1,'%s')
a12 = 123
octave:53> a12=sscanf(L1,'%f')
a12 =

  1
  2
  3

octave:54> a11=sscanf(L1,'%d')
a11 =

  1
  2
  3

octave:55> a12=sscanf(L2,'%d')
a12 =

  3
  2
  1

octave:56> a11+a12
ans =

  4
  4
  4
```

尚,d の場合,Octave では自動的に列ベクトルになります. ところが, MATLAB では行ベクトルになります. これは Octave が列ベクトルを基準として

いる傾向がある為でしょう. この点は Octave と MATLAB の両方で動作するプログラムを作成する際には, 特に注意が必要な個所の一つです.

この様にファイルが文字列か数字列の何れかで構成されている場合,fgets 命令で行毎読んで,sscanf で形式の変換を一度に行えば良い事になります. ところが,L3 の様に数と文字が混合している場合は厄介です.L3 の様に意地の悪い代物は, 悪意を持って例として示しているので仕方がありませんが, 次のファイル tama の様な形式は非常に普通でしょう.

```
# No. Flag  Value
1     t      10
2     f     -10
3     t      20
4     f     -20
```

この様なデータの場合,sscanf で一気に変換する事は意味が無く, 本来なら書式指定で各々を変換するのが本来の姿です. ところが,MATLAB には与えられたストリームを一気に変換する事しか出来ません.

先ず, ファイル tama を開き, 安易に一行づつ sscanf で整数型 ('%d') や文字列型 ('%s') へと一気に変換した例を示しましょう.

```
octave:73> fid2=fopen('tama','r')
fid2 = 3
octave:74> tama1=fgets(fid)
tama1 = # No. Flag  Value

octave:75> tama2=fgets(fid)
tama2 = 1     t      10

octave:76> tama3=fgets(fid)
tama3 = 2     f     -10

octave:77> tama4=fgets(fid)
tama4 = 3     t      20

octave:78> tama5=fgets(fid)
tama5 = 4     f     -20

octave:79> st1=sscanf(tama1,'%d')
st1 = [](0x1)
octave:80> st1=sscanf(tama2,'%d')
```

```
st1 = 1
octave:81> st1=sscanf(tama3,'%d')
st1 = 2
octave:82> st1=sscanf(tama4,'%d')
st1 = 3
octave:83> st1=sscanf(tama5,'%d')
st1 = 4
octave:84> st1=sscanf(tama1,'%s')
st1 = #No.FlagValue
octave:85> st1=sscanf(tama2,'%s')
st1 = 1t10
octave:86> st1=sscanf(tama3,'%s')
st1 = 2f-10
octave:87> st1=sscanf(tama4,'%s')
st1 = 3t20
octave:88> st1=sscanf(tama5,'%s')
st1 = 4f-20
```

この様に,sscanf による変換では形式に対応しない個所以降の列は除外されてしまいます. 例えば,'%d' で整数型に変換する個所を見て頂くと判りますが, 二列目の flag に当たって変換が途中で終了してしまい,flag の前の No. に相当する数のみが返されています. 更に, ファイル先頭行の文字列は変換が出来ずに空行が返されています. その上, ストリームを一気に文字列に変換した場合, 空行やタブは省略されています. 実際,`2 f -10` が `2f-10` に変換されたりしていますね.

この様に MATLAB の sscanf,fscanf の互換性のある方式ではこれ以上の処理が出来ません. そこで,Octave で sscanf 命令に C フラグを立てると C と同様に複雑な形式のファイルにも対応出来る様になります.

```
octave:94> [s1,s2,s3,s4]=sscanf(tama1,'%s %s %s %s','C')
s1 = #

s2 = No.

s3 = Flag

s4 = Value

octave:95> [n1,flg,n3]=sscanf(tama2,'%d %s %d','C')
n1 = 1
```

```
flg = t

n3 = 10
```

この様に Octave であれば, より C 風に sscanf や fscanf を利用出来ますが, この C フラグを立ててしまうと今度は MATLAB では利用出来ません. この場合, スマートではありませんが, 次の様にストリームを分けて対処すれば互換性に問題が生じる事もなく, 上手に処理が行えます.

```
octave:96> find(tama2=='t' | tama2=='f')
ans = 7
octave:97> n1=sscanf(tama2(1:6),'%d')
n1 = 1
octave:98> n2=sscanf(tama2(8:length(tama2)),'%d')
n2 = 10
octave:99> flg=sscanf(tama2(7),'%s')
flg = t
```

ここで, 最初に find を用いている個所はストリーム tama2 からフラグ値の t か f のどちらかがある個所を求めて, その個所から前後に分けて整数に変換しています. ここで, 記号'—' は Octave/MATLAB の論理和になります.

同様にコメント行かどうかは'#' がストリームに存在するかどうか検証するだけで出来てしまいます. だから, 全てを一旦文字列に変換し, 先頭が'#' であるかを判別する様にすれば良い事になります.

尚, 重要な事に Octave と MATLAB の両方では, 扱う行列データは文字か数値のみしか許容されず, 両者が混在した行列を扱う事は出来ない事です.

この様に行列データに制約があるので, 文字と数値が混在する表の扱いでは, 色々な行列データを準備する必要がある様に思えます.

Octave と MATLAB の双方に C の構造体と同様のデータ構造を用いる事が可能なので, そちらを使うと多くの行列を利用する必要が無くなってしまいます. その上,MATLAB と Octave で構造体を利用する場合は, 他の変数と同様に予め宣言する必要は無く, 以下の例の様に直接利用しても構いません.

実際に Octave で構造体を用いた例を示しましょう.

```
octave:102> [neko.n1(1),neko.flg(1),neko.n2(1)]=sscanf(tama2,'%d %s %d','C')
neko.n1 = 1

neko.flg = t
```

```
neko.n2 = 10

octave:103> [neko.n1(2),neko.flg(2),neko.n2(2)]=sscanf(tama3,'%d %s %d','C')
neko.n1 = 2

neko.flg = f

neko.n2 = -10
```

この例ではsscanfで変換したデータをneko.n1,neko.flg,neko.n2に割当てています. ここで各々の配列は数値と文字列になっている事に注意して下さい.

MATLAB と Octave ではこの様に構造体を用いて文字と数値が混在したデータを一括して扱えます. ここで, データが構造体かどうかは直接データ名を入力する事でも判別可能ですが, この場合, 全てのデータが一度に表示されるので,Octave の場合, is_struct 命令で構造体かどうかを判定し, struct_elements 命令で構造体の構造が調べられます. 但し,MATLAB の場合は命令の名前は似ているが, 全く別の命令を用います. その為, 互換性に注意が必要です.

```
octave:104> neko
neko =
{
  n2 =

     10
    -10

  flg =

t
f

  n1 =

    1
    2
}

octave:107> is_struct(neko)
ans = 1
```

```
octave:108>

octave:109> struct_elements(neko)
ans =

n2
flg
n1
```

この様にMATLABとの互換性を考慮すると, 泥臭い処理が必要になりますが, fgetsとsscanf命令を上手く用いれば, 通常のファイルの処理も出来ます.

## 6.3 ファイルの更新とデータの追加の例

fopenの指定でファイルの更新, 例えば, 内容の入れ替えや, データの追加が行えます. 先ず, ファイルの更新では,'w' を指定しましょう. こうすると, 既存のファイルの内容は消去され, もしも, 存在しない場合には, 指定された名前のファイルが新規に生成されます. 又, 既存のファイルを利用する場合,'a' を指定すると, 既存のファイルのデータに続けて書込めます.

次に, 例として与えられた行列データをフラグに沿って指定されたファイルに追加したり, 置き換えるプログラムを示しましょう.

```
function [err]=appendDATA(fname,mv,flg)
        err   = 0;
        % ファイル名の設定.
        vfname=[fname,'.vdt'];
        % フラグ flg==0 であれば上書き, それ以外は末尾にデータを
        % 追加する.
        if flg==0
                vfp = fopen(vfname,'w');
        else
                vfp = fopen(vfname,'a');
        end;

        % データの書き込み
        [m,n]=size(mv);
        if m>0
                if flg==0
                        fprintf(vfp,' %d ',mv(1,:));
```

```
                fprintf(vfp,'\n');
        end;
        for k=[2:m]
                fprintf(vfp,'%22.15e',mv(k,:));
                fprintf(vfp,'\n');
        end;
else
        err=1;
end;
fclose(vfp);
```

この例では, ファイル名は修飾子".vdt"を抜いた型で与える. 又, ファイルの先頭行はカラムを分類する為に整数としており, 2 行目以降が実際のデータで, 行列 mv もその様な形式となっています. このデータ行の出力並びは fprintf 命令の'%22.15e' で指定したもので, この書式の設定は C や FORTRAN のそれと同じ形式になります. 更に, この処理を"%データの書き込み"と註釈を入れた個所から下で行っています.

以上の様に Octave のファイル処理では, 命令やオプションがほぼ C に似た仕様となっているので,MATLAB との互換性を考慮しなければ, C 風に記述して C オプションを立てておけば便利です. 更に, 扱うデータを数値行列にすれば非常に苦労も少なく, 非常に気楽にファイル操作が行える仕様となっています.

しかし,MATLAB との互換性を考慮すれば,MATLAB で書式指定を伴う処理では行単位で行われるので, 一行に文字と数値が混在データファイルを扱う場合には工夫が必要になります. その上,Octave の独自の機能に頼ると MATLAB で動作しないプログラムとなる可能性が非常に高くなってしまうので注意しましょう.

# 索 引